Diagnostic Reasoning with Structural Analysis and Constraint Programming for Quality Improvement of Business Process Management Systems

Diana Borrego Núñez

dianabn@us.es

Supervised by
Prof. Dr. María Teresa Gómez López
Prof. Dr. Rafael Martínez Gasca

Thesis Dissertation submitted to the Department of Computer Languages and Systems of the University of Sevilla in partial fulfilment of the requirements for the degree of Ph.D. in Computer Science.

(Thesis Dissertation)

A Eduardo.

Acknowledgements

I would like to thank all those people who contributed in many ways to the success of this study. It is a pleasure to thank those who made this thesis possible, supporting and helping over the last few years.

I wish to thank, first and foremost, my advisors María Teresa Gómez López and Rafael Martínez Gasca, who shared with me a lot of their expertise and research insight. This thesis would not have been possible unless their continuous support of my Ph.D study and research, for their patience, motivation, enthusiasm, and knowledge.

I would like to show my gratitude to the Quivir Research Group members, and especially to my little cluster, for providing a stimulating and friendly environment in which to share and discuss ideas.

I am indebted to Los Simpáticos Investigadores for helping me get through the difficult times, and for all the emotional support.

I cannot find words to express my gratitude to my family, who has always stood by me and dealt with all of my absence from many family occasions with a smile.

And last but not least, I owe my deepest gratitude to Eduardo. Thank you for everything.

Agradecimientos

Me gustaría dedicar unas palabras de agradecimiento a toda esa gente que ha contribuido, de muchas distintas formas, al éxito de este trabajo. Es para mí un placer agradecer a quienes han hecho posible esta tesis mediante su apoyo y ayuda durante estos últimos años.

En primer lugar, desearía dar las gracias a mis directores de tesis María Teresa Gómez López y Rafael Martínez Gasca, quienes han compartido conmigo su profundo entendimiento y habilidad en la investigación. El desarrollo de esta tesis no habría sido posible sin su contínuo apoyo, paciencia, motivación, entusiasmo, y conocimiento.

Me gustaría mostrar mi agradecimiento a los miembros del Grupo de Investigación Quivir, y en especial a mi pequeño cluster, por proporcionar un entorno estimulante y cordial donde compartir y discutir nuestras ideas.

Quiero mostrar mi gratitud a Los Simpáticos Investigadores por ayudarme en los malos momentos, y por todo el apoyo emocional.

No puedo encontrar palabras para expresar el agradecimiento a mi familia, que siempre ha estado ahí, y ha comprendido mis ausencias en ocasiones familiares con una sonrisa.

Y por último, debo mi más profundo agradecimiento a Eduardo. Gracias por todo.

Abstract

Everyday more and more complex and critical processes of organizations' services and operations are automated by using business process management systems. Thereby, there exists a growing interest in improving the quality of these processes in order to ensure the reachability of business goals and, consequently, for organizations to become more competitive. To this end, in the current Thesis Dissertation, diagnosis techniques are applied at different stages of the business process lifecycle in order to identify and isolate functional faults both at design-time and runtime. The main contributions presented in this dissertation are: (1) diagnosis of the correctness of business process models which include semantic information to describe the behaviour of the processes; (2) decision about the better places to monitor the processes in order to diagnose possible functional faults at runtime; and (3) diagnosis of the abnormal behaviour of activities at runtime for correctly designed processes.

More precisely, in a first place, we propose a fully automated approach for diagnosing correctness of semantic business process models referencing both the control flow and data flow perspectives, in which the semantics of activities is specified with pre and postconditions. The control flow and data flow perspectives of the models are modelled in an integrated way using Artificial Intelligence techniques, enabling the verification and diagnosis of two kinds of correctness of complex semantic business process models in an efficient way.

Secondly, the diagnosability problem in business processes is addressed. Diagnosability analysis aims to determine whether observations available during the execution of a system are sufficient to monitor and precisely locate the source of a problem. Therefore, one of the aims of this Thesis is to determine a test-point allocation to obtain sufficient observable data in the data flow to allow the discrimination of faults for a later diagnosis process. This objective is formally stated in terms of an optimization problem, and reaches a given objective function depending on the business policy, under a set of constraints due to the availability of test-point resources within the domain. Three test-point allocation strategies are proposed to ensure the best business process diagnosability.

And finally, in order to identify and isolate functional faults at runtime, this Thesis Dissertation provides two techniques to diagnose the abnormal behaviour of the activities by means of an analysis of the trace of process instances, depending on the information available on the model: (1) if the model counts on business rules defining the behaviour and semantics of the process, we propose to automate the diagnosis of the abnormal executions by taking into account the involved activities and their business compliance rules, and using these rules to determine the activities which are not working in accordance with the contract defined in the model; and (2) if the model does not include any semantic information, or the available information cannot be modelled as a contract, this Thesis Dissertation provides a proposal to diagnose faulty activities in business-to-business collaborations using choreography structural analysis.

Resumen

Cada día más empresas tienden a automatizar los procesos más complejos y críticos de sus servicios y operaciones mediante el uso de sistemas de gestión de procesos de negocio. En consecuencia, existe un creciente interés en mejorar la calidad de estos procesos, para asegurar que se alcanzan los objetivos de negocio y hacer que las empresas sean más competitivas. Para cumplir con este objetivo, es necesario detectar cuándo y porqué un proceso no cumple sus objetivos de negocio, y que esto se haga de la forma más eficiente posible, haciendo así a las empresas competitivas. Con este fin, en la presente memoria de Tesis se aplican técnicas de diagnosis en diferentes fases del ciclo de vida de los procesos de negocio para, de esta forma, identificar y aislar fallos funcionales tanto en tiempo de diseño como de ejecución de los procesos. Las principales aportaciones presentadas en la presente memoria de tesis comprenden: (1) diagnosis de la corrección del modelo de los procesos mediante el uso de la descripción semántica de su comportamiento; (2) decisión acerca de dónde es mejor realizar la monitorización de los procesos de negocio para así poder diagnosticar posibles errores de funcionamiento en tiempo de ejecución; y (3) diagnosis del comportamiento incorrecto de actividades en tiempo de ejecución para procesos correctamente diseñados.

Más concretamente, en primer lugar, se propone un enfoque totalmente automático para diagnosticar la corrección de modelos semánticos de procesos de negocio. Dichos procesos reflejan tanto la perspectiva de flujo de control como la de flujo de datos, en los que la semántica de las actividades se especifica mediante el uso de pre y postcondiciones. Ambas perspectivas son modeladas de forma integrada mediante el uso de técnicas de Inteligencia Artificial, permitiendo la verificación y diagnosis de dos tipos de corrección de modelos complejos de procesos de negocio de manera eficiente.

En segundo lugar, se trata el problema de la diagnosticabilidad de los procesos de negocio. El objetivo del análisis de la diagnosticabilidad es determinar si las observaciones disponibles durante la ejecución de un sistema son suficientes

para monitorizar y localizar el origen de un problema de forma precisa. Por lo tanto, uno de los objetivos de esta Tesis es determinar la colocación de puntos de test para obtener los suficientes datos observables en el flujo de datos que permitan la discriminación entre fallos en un proceso de diagnosis posterior. Este objetivo se describe formalmente como un problema de optimización, para alcanzar una determinada función objetivo dependiendo de la política de negocio, bajo un conjunto de restricciones derivadas de la disponibilidad de puntos de test. Se proponen tres estrategias de colocación de puntos de test para asegurar la mejor diagnosticabilidad de procesos de negocio.

Y por último, para identificar y aislar fallos en tiempo de ejecución, esta memoria de Tesis proporciona dos técnicas para la diagnosis del comportamiento operacional de las actividades a partir de los datos de entrada y salida, mediante el análisis de las trazas de las instancias del proceso, dependiendo de la información disponible en el modelo: (1) si el modelo cuenta con un contrato que define la semántica del proceso, se propone la automatización de la diagnosis de las ejecuciones anómalas de los procesos. Esto se llevará a cabo teniendo en cuenta la actividades participantes y sus reglas de negocio, y utilizando dichas reglas para determinar las actividades cuyo comportamiento no es acorde al contrato definido en el modelo; y (2) si el modelo no include ninguna información semántica, o bien la información disponible no puede modelarse como un contrato, la presente memoria de Tesis proporciona un enfoque para diagnosticar actividades de comportamiento incorrecto en colaboraciones entre procesos mediante el análisis estructural de la coreografía.

Contents

Contents **xii**

List of Figures **xvi**

List of Tables **xvii**

I Preface 1

1 Introduction **3**

1.1 Research Context and Motivation . . . 3
1.2 Contributions . . . 7
1.3 Structure of this Dissertation . . . 12

II Background 15

2 Business Processes **17**

2.1 Concepts . . . 17
2.2 Business Process Model . . . 19
2.2.1 Control Flow Perspective . . . 19
2.2.2 Data-Flow Perspective . . . 20
2.3 Background to Business Process Analysis . . . 22
2.3.1 Verification Analysis Methods . . . 24
2.3.2 Validation Analysis Methods . . . 25
2.3.3 Performance Analysis Methods . . . 26
2.3.4 Discussion . . . 27

3 Diagnosability and Model-Based Fault Diagnosis 31
3.1 Diagnosability 32
3.2 Model-Based Diagnosis 35
3.2.1 FDI: Analytical Redundancy Approach 37
3.2.2 DX: Logical Diagnosis Approach 40
3.3 Distributed Diagnosis 41
3.3.1 Decentralized Approaches 42
3.3.2 Distributed Approaches 44

4 Constraint Programming 49
4.1 Basic Concepts in CSP Modelling 50
4.1.1 Consistency 51
4.1.2 Search Algorithms 51
4.1.3 Constraint Optimization Problem 51
4.1.4 Integer Programming 52
4.2 Overconstrained Constraint Satisfaction Problems 52
4.3 Business Data Constraints and CP 55

III Contribution I: Verification of Semantic Business Process Models 59

5 Diagnosing Semantic BP Models 61
5.1 Introduction 61
5.1.1 Motivation 61
5.1.2 Contribution 62
5.1.3 Illustrative Example 64
5.2 Workflow Data Graphs 69
5.2.1 Definitions 69
5.2.2 Analysis of the Diagnosis Problem 71
5.3 Diagnosis of Workflow Data Graphs 75
5.3.1 Combined IP and CSP Model 76
5.3.2 Detecting Basic Data-Flow Errors 79
5.3.3 Algorithm for May-correctness 81
5.3.4 Algorithm for Must-correctness 83
5.3.5 Diagnosing the Motivating Example 84
5.4 Implementation and Empirical Evaluation 86
5.4.1 Implementation 86

5.4.2 Experimental Design 89
5.4.3 Performance Results 89
5.5 Summary 90

IV Contribution II: Diagnosability in Business Processes 93

6 Improving the diagnosability in BP models 95
6.1 Introduction 95
6.1.1 Motivation 96
6.1.2 Contribution 97
6.1.3 Illustrative Example 98
6.2 Diagnosability of Business Processes 100
6.2.1 Definitions 100
6.2.2 Objectives for the Improvement of the Diagnosability of a Business Process 101
6.3 Test-Point Allocation Methodology 103
6.3.1 Improving Diagnosability Using Constraint Programming 105
6.3.2 Objective 1: Maximization of the Number of Clusters with the Allocation of a Fixed Number of Test Points. . . 108
6.3.3 Objective 2: Allocation of the Minimum Number of Test Points in Order to Obtain a Fixed Number of Balanced Clusters. 110
6.3.4 Objective 3: Minimization of the Number of Test Points to be Allocated in Order to Obtain Clusters with a Maximum Number of Activities. 111
6.4 Implementation and Empirical Evaluation 116
6.4.1 Implementation 116
6.4.2 Experimental Design 118
6.4.3 Performance Results 119
6.5 Summary 121

V Contributions III and IV: Fault Diagnosis of Business Processes at Runtime 123

7 Diagnosis based on Business Data Constraints 125
7.1 Introduction 125

7.1.1 Motivation 126
7.1.2 Contribution 127
7.1.3 Illustrative Example 129
7.2 Contract-based Diagnosis of BPs 131
7.3 Diagnosis Process by using Business Data Constraints 133
7.3.1 Contract-based Diagnosis Using Reified Constraints and Max-CSP 135
7.3.2 Determination of MUSes for Contract-based Diagnosis . . 136
7.4 Empirical Evaluation 138
7.4.1 Experimental Design 138
7.4.2 Performance Results 139
7.5 Summary 140

8 Diagnosis based on Structural Analysis 143
8.1 Introduction 143
8.1.1 Motivation 144
8.1.2 Contribution 144
8.1.3 Illustrative Example 146
8.2 Main Concepts for the Structural Diagnosis 147
8.3 Logic and Distributed Algorithm 149
8.3.1 Determining the Compiled Orchestration Model 150
8.3.2 Propagation Phase 153
8.3.3 Local Diagnosis Phase 157
8.4 Empirical Evaluation 159
8.4.1 Experimental Design 159
8.4.2 Performance Results 160
8.5 Summary 161

VI Conclusions and Future Work 163

9 Final Remarks 165

10 Future Work 171

Bibliography 173

List of Figures

1.1 Business process lifecycle 4
1.2 Diagram with the contributions for the diagnosis of business processes 8

2.1 Main business process gateways 21

3.1 Polybox system 37

5.1 Example of business process 64
5.2 Example of instance subgraph (shadowed activities) 73
5.3 Diagram of the process 75
5.4 Business process example 78
5.5 Algorithm for checking absence of conflicts 80
5.6 Algorithm for checking may-correctness 82
5.7 Algorithm for checking must-correctness 84
5.8 Screenshot of DiagFlow indicating may-correctness 88
5.9 Screenshot of DiagFlow indicating no must-correctness 88
5.10 Performance results 90

6.1 Motivating example 99
6.2 Graphical representation of the approach 104
6.3 Example of hyperedge transformation 105
6.4 Motivating example with two test points 107
6.5 Business process with four levels 113
6.6 Business process with labels in the activities 113
6.7 Tree of levels 114
6.8 Greedy recursive algorithm for the allocation of test points 115
6.9 Reduced business process 116
6.10 Screenshots of Test-Point Allocator 117

6.11 Execution time for Objective 1 118
6.12 Execution time for Objective 2 119
6.13 Execution time for Objective 3 120

7.1 Example of the business process for the organization of a conference 129
7.2 Contract-based diagnosis process 131
7.3 Algorithm to obtain the *MUSes* (I) 137
7.4 Algorithm to obtain the *MUSes* (II) 138

8.1 Example of Business-to-Business Collaboration 146
8.2 Phases of the diagnosis process 150
8.3 Graphical representation of a local path with three Internal Subsets 153
8.4 Paths and Internal Subsets of the Business-to-Business Collaboration 154
8.5 Algorithm executed when a local diagnoser receives information . 156
8.6 Procedure for labelling interactions as correct or incorrect 157
8.7 Procedure for propagating information 158
8.8 Signature matrix of a path of BP_2 158
8.9 Graphic representation of the results 161

List of Tables

2.1 Mapping of the Business Process Analysis methods addressed . . 29

3.1 Signature matrix for single faults 39

5.1 Data input for the example in Figure 5.1 66
5.2 Data read and written on each activity 66
5.3 Activities with their pre and postconditions 67
5.4 Variables before and after the renaming 79
5.5 Activities with their pre and postconditions in SSA form 85

7.1 Data input domain for the example in Figure 7.1 129
7.2 Business data constraints for the example in Figure 7.1 130
7.3 Comparison between MUSes algorithms 140
7.4 Diagnosis results 141

8.1 Notation for the elements in a collaboration 147
8.2 Temporal results 161

Part I

Preface

Chapter 1

Introduction

Nowadays, organizations are automating more complex and critical processes of their services and operations with business processes that can be enacted using Business Process Management Systems. Within the companies, the performing of the activities related to those processes is carried out by the staff of the organization or by external companies, which can be helped by information systems, or even in a fully automated way by information systems.

For an organization to reach its business goals and become more competitive, it is important to count on high quality business processes. It becomes necessary that the processes are accurate, suitable and effective, which entails they are carried out without failures, or with the percentage of failures as low as possible. Each process that affects a business goal is vital for the business success. Thereby, it is crucial to provide techniques to improve the quality of the execution of the processes, by diagnosing and isolating the possible failures that take place. This is the aim of the current Thesis Dissertation, in order to improve overall system functionalities by applying diagnosis techniques at different stages of the business process lifecycle in efficient ways to detect the responsible for a malfunction.

1.1 Research Context and Motivation

The development of Information Technologies (IT) and the requirements of the organizations have made the integration of systems become necessary (e.g. Web Services, software applications or even business applications). These systems, together with the participation of human resources, develop collaborative tasks to fulfil business goals. To this end, in the last years, there is a trend toward the automation of the operations performed by organizations by using business processes. They are used to collect, organize and synchronize tasks which are logically related and should be carried out to achieve a defined goal, even when the goal requires interaction with other organizations. By means of the use of business processes within the organizations, a better understanding of the operations can be achieved, but it entails pros and cons: on the one hand, they enable the development of larger and more complex projects automating the process, and reusing and paralleling tasks; on the other hand, they imply the use of automatic monitoring and fault diagnosis methodologies.

Business Process Management (BPM) enables to ensure the effective and efficient design of business processes, administration, configuration, enactment and analysis, which are phases of the business process lifecycle, usually organized in a cyclical structure (Weske, 2007), shown in Figure 1.1.

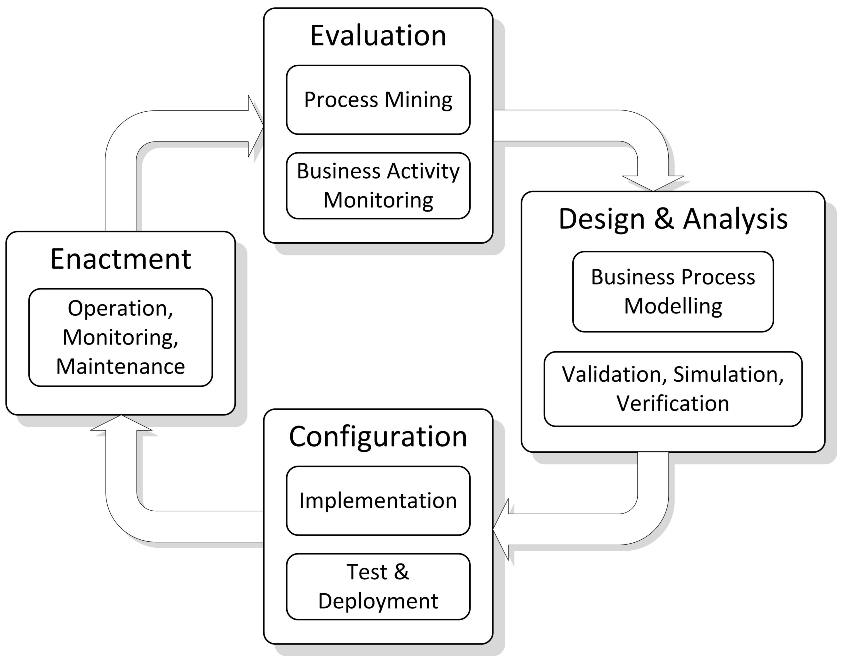

Evaluation
Process Mining
Business Activity Monitoring
Design & Analysis
Business Process Modelling
Validation, Simulation, Verification
Configuration
Implementation
Test & Deployment
Enactment
Operation, Monitoring, Maintenance

Figure 1.1: Business process lifecycle

In the *Design & Analysis* phase, business processes are modelled, including validation and verification by simulating the execution of the just designed processes. Then, they are implemented in the *Configuration* phase by configuring a process-aware information system, entailing the testing of the implementation. In the *Enactment* phase, the business processes are deployed using the configured system, and the execution of process instances takes place. Finally, in the *Evaluation* phase, the processes are monitored and analysed to evaluate and improve business process models and their implementations.

When, during this lifecycle, a business process is enacted using a Business Process Management System (BPMS) and a process is executed, any fault which affects the behaviour of the process -with respect to the managed data- has negative effects on the operational activities over the organization or organizations which are collaborating, revealing low quality of the services and products. Examples of faults could be a process that gets stuck and never finishes, or an expected goal that is not achieved. These faults can be detected by end-users or customers which present their corresponding complaints about the accuracy of the service, or even by monitoring systems, leading to problems such as angry customers, damage claims, and low service levels.

Fault diagnosis aims to determine why a business process does not work as it is expected. The diagnosis goal is to identify the reason of an unexpected behaviour, or in other words, to identify the part or parts which fail in a business process. Hence the fault detection and later diagnosis of business processes becomes crucial, since their proper working is an essential requirement to maintain the business processes in desirable quality and production levels. The unexpected faults can cause undesirable stops in the processes, causing cost increase and production decrease. Therefore, through the inclusion of automatic diagnosis techniques in the business process lifecycle, the quality of the attained business processes is improved by ensuring the right or agreed results or effects of business processes with the needed degree of precision (accuracy), providing an appropriate function for specified user objectives (suitability), which enables users to achieve specified goals with accuracy and completeness (effectiveness) (Weske, 2007; van der Aalst et al., 2003; Heravizadeh et al., 2009).

The different tasks which compose a business process related to important operations or products of a company are often buried among other applications or even the practices of the staff in the organization. This makes difficult to understand how a process works exactly, who does what, and who is responsible for each task. The way in which the faulty elements are isolated in the processes of an organization indicates the level of control that the organization has on the business. It makes possible for the organizations to identify quickly the points where the business processes fail, or also where the human errors are often in activities like introducing data in the information systems or designing business processes in accordance with data inputs and outputs.

During the aforementioned lifecycle of business processes, there are different stages where it should be considered to perform different diagnosis tasks.

In detail, after the analysis of the most appropriate strategies to reach a defined goal, and the designing of the model of a business process to carry it out (*Business Process Modelling* within the *Design & Analysis* phase in Figure 1.1), it is desirable to verify the correctness of the defined model. *Validation, Simulation, Verification* in Figure 1.1 is in charge of this activity. In the case the model is not verified correct, a new redesign of the model is required.

Regarding the kind of errors that a business process model may present, two different kinds of verifications are necessary in accordance with both perspectives of a model:

- **V**erification of the control flow perspective. This perspective concerns the topology and conditions that set the order in which the individual activities within a business process are executed. This kind of verification of a model entails the identification of structural errors, such as deadlocks, livelocks, etc.

- **V**erification of the data flow perspective. This perspective details the flow of data among activities subject to certain constraints that define (i) a business contract to determine behaviour or semantics of the overall process, or (ii) the semantics of individual activities by annotating them with pre and post-conditions. This kind of verification of a model includes the checking of missing or redundant data, read/write data conflicts, or even inconsistencies in the modelling of the semantics.

Once the business process model has been verified correct, the *Design & Analysis* phase ends, and the corresponding business process is configured and enacted in order to put it in operation and enable the executions of business process instances (phases *Configuration* and *Enactment* in Figure 1.1).

In the case that, during the testing of an implementation (*Test & Deployment* in Figure 1.1) or after the execution of a process instance (*Operation* in Figure 1.1), the goal to achieve by the business process is not reached, it is necessary to perform the diagnosis of the process in order to determine the source of the problem. This diagnosis can be focused on two possible causes of the fault: (1) certain activity or activities within the business process are not working as they were modelled; or (2) there exists some kind of inconsistency in the data inputs introduced through some kind of human interface, service, application, etc.

Regarding the first cause, in order to identify and isolate the incorrect activity or activities, it should be considered that the diagnosis process is based on observations, which provide information about the behaviour of the process. Depending on those observations, the business process can be diagnosed with a higher or lower level of accuracy. If the diagnosis is based on a few observations, or even if they are not performed at the most convenient locations in the business process, it would be very difficult to distinguish which parts of the business process are failing. Both the amount of observations and their locations make possible to precisely locate the source of the problem. This determines the diagnosability level of business processes.

The success of the introduction of Business Process Management Systems in an organization lies in the necessity of doing an appropriate integration of the model of the processes (control flow and data flow perspectives) with the automation of these processes and the isolation of the faults that appear during the executions (fault diagnosis).

1.2 Contributions

Fault diagnosis is a problem which has been widely analysed for several areas of knowledge. Nevertheless, the adaptation of the existing diagnosis techniques to business processes, and the performing of new methodologies in accordance with the features of business process models, are researching topics of great application and relevance which have only been deeply explored by other researchers regarding the control flow perspective of business processes.

Those are the aims of the current Thesis Dissertation, whose main goal is to provide automatic techniques to diagnose business processes from both the business process model (describing expected behaviour) and the available observations (facilitating the actual behaviour). In short, our contributions entail: (1) the verification of the correctness of semantic business process models (in terms of the data flow perspective); (2) the improvement of the diagnosability by allocating test points in order to provide sufficient observations showing the actual behaviour of business processes; and the diagnosis of activities which do not work as they were modelled either (3) with semantics or (4) without semantics in their models. The diagram in Figure 1.2 shows the relation between these contributions, represented as a workflow with the corresponding data inputs and outputs, detailed in the following.

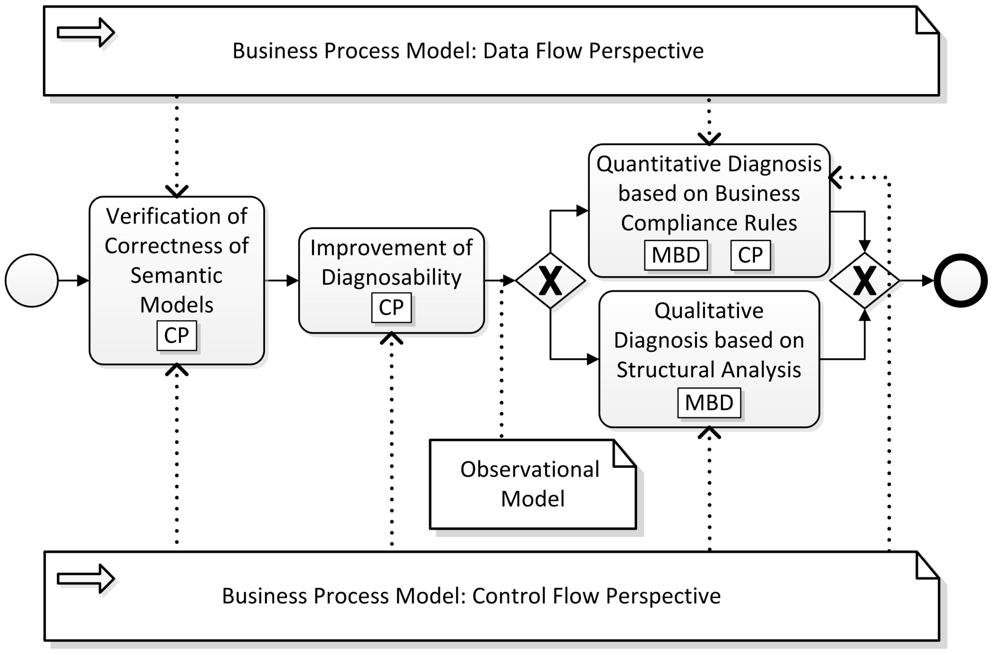

Business Process Model: Data Flow Perspective
Verification of Correctness of Semantic Models
CP
Improvement of Diagnosability
CP
Quantitative Diagnosis based on Business Compliance Rules
MBD
CP
Qualitative Diagnosis based on Structural Analysis
MBD
Observational Model
Business Process Model: Control Flow Perspective

Figure 1.2: Diagram with the contributions for the diagnosis of business processes

In accordance with the stages presented in Figure 1.1, the starting point of our contributions in Figure 1.2 is a designed business process model, counting on both control flow and data flow perspectives, represented as data inputs of our contributions. Likewise, these contributions are represented as tasks within the workflow to provide certain functionalities: (1) *Verification of Correctness of Semantic Models* task, to verify the correctness of business process models with regard to the data flow perspective, in an off-line way (i.e. before the business processes are enacted); (2) *Improvement of Diagnosability* task, using the verified correct models to determine the places where each business process should be monitored in order to attain sufficient observations; (3) *Quantitative Diagnosis based on Business Compliance Rules* task, to perform run-time diagnosis of processes which, counting on a contract to describe the behaviour within the data flow perspective, do not work as expected; and (4) *Qualitative Diagnosis based on*

Structural Analysis task, to also diagnose processes at run-time, without contract to describe the behaviour in this case and thereby using only structural analysis of the topology of the processes. These contributions are detailed in the following.

Verification of Correctness of Semantic Models

As it was stated above, it is desirable to ensure the correctness of the model before using it to configure and enact business processes during the stages of the lifecycle (cf. Figure 1.1). To this end, our proposal counts on the *Verification of Correctness of Semantic Models* contribution, which is in charge of verifying the correctness of business process models with regard to the data flow perspective.

Control flow verification of models has been deeply explored by other researchers, so that errors arisen from incorrect design of the structure of the processes, such as livelocks and deadlocks, count on several previous works in the literature. Therefore, this approach provides a contribution focused on the diagnosis of business process models with available data flow perspective. More precisely, when the data flow perspective of the model includes annotations about the behaviour of each single activity, it is possible to diagnose the semantics of the model in a deeper way. Then, this Thesis Dissertation provides a fully automated approach in Chapter 5 for diagnosing two kinds of correctness of business process models in which the semantics of the activities is specified with pre and postconditions.

To this end, this contribution models the control flow and data flow perspectives in an integrated way by using the Constraint Programming paradigm (referred to as *CP* in Figure 1.2).

Improvement of Diagnosability

Diagnosis techniques are based on observations about the actual behaviour of processes in order to be able to identify and isolate the responsible for any malfunction. Therefore, it is a design-time requirement to consider that, since there is not a single entity which has a global view of the complete data flow model, the observations available during the execution of a business process cannot be enough to precisely locate the source of a problem. In order to avoid a low diagnosability of business processes, the proposal in Figure 1.2 counts on the *Improvement of Diagnosability* task in charge of the diagnosability improvement. That is performed over a business process model already verified correct and before the enactment of the business processes. As a result, *Observational Models* of the processes are attained, so that later diagnosis processes can count on observable data from the data flow in order to get a minimal diagnosis.

As contribution in this field, Chapter 6 provides a technique to monitor business processes in order to improve their diagnosability. This is done by carrying out observations, not only at the outputs of the processes, but at certain intermediate flows by means of the allocation of test points over the input and output variables of the activities. This fact guarantees the observability and monitoring of the data flow at those locations during the execution of a business process instance. Therefore, the optimal allocation of test points is essential, since if the test points are not correctly allocated, the observed data may not be useful to isolate the faults.

In order to get fully diagnosable business processes, a solution to consider would be to allocate test points at every flow between any pair of activities of the business processes. However, this is not an efficient solution due to several reasons like the extra cost derived from the monitoring and evaluation of too many test points, or even to confidentiality, privacy or security policies that forbid the observation of some data flowing through the business process. Therefore, the contribution detailed in Chapter 6 opts for selecting the test points which are necessary and economically feasible -in term of test-points resources- to get an expected diagnosability level. It faces up to three independent objectives to reach, derived from the most common requirements of the organizations, like economic limitations, or even the optimization of the execution time of a later diagnosis process.

Quantitative Diagnosis based on Business Compliance Rules

When business processes are built over models which include a data flow perspective to define business contracts, those contracts determine, by means of rules which should be complied, the behaviour or semantics of the overall business process, providing quantitative information about the relation among data involved in the execution of each activity.

Faults which are caused by the no compliance of the contract have to be detected after the execution of a process instance, since it is not possible to diagnose them before the enactment and execution of the process but at run-time after a discrepancy between the observed and expected behaviour is identified.

Therefore, the *Quantitative Diagnosis based on Business Compliance Rules* task is in charge of this kind of diagnosis, detailed in Chapter 7, which presents an automatic diagnosis method by applying Model-Based Diagnosis principles (*MBD* in Figure 1.2) to verify the correctness of the activities by analysing the corresponding business compliance rules and the available quantitative data. It establishes relationships between the compliance rules and each activity (or set of activities), that represent the contract (i.e. behaviour) that each activity should satisfy throughout the data flow.

The contribution uses two strategies based on Constraint Programming to consider the trade-off between the obtaining of the minimal diagnosis and the performance.

Qualitative Diagnosis based on Structural Analysis

On the other hand, when business processes models, already verified correct, only count on the control flow perspective, or even when the data flow perspective is not modelled as a behavioural contract, it is again not possible to diagnose the discrepancies that may occur after a process instance until they are enacted and executed. That is, a diagnosis process can be performed if, after the execution or testing of the business processes, an abnormal behaviour is reported by the user. Then, some parts of the processes are not working correctly according to their behavioural model.

To this end, the *Qualitative Diagnosis based on Structural Analysis* task, detailed in Chapter 8, is in charge of the diagnosis of business processes based only on the structural analysis of the control flow of the processes and the available qualitative observations indicating if some discrepancy is identified. Due to there is not single entity with a global view of the overall business processes, the method in Chapter 8 uses a local *Diagnoser* linked to each business process involved in the attainment of the objective to reach. The set of local diagnosers works in a coordinate way, being the diagnosis result the activity or activities which are not working as they were modelled. Since the diagnosis process takes place when the business processes have already been enacted, in order to avoid long wait times, the contribution gets a more efficient diagnosis process by performing an off-line analysis of the structure of the processes, attaining a preprocessed model which is the base to perform the Model-Based Diagnosis (*MBD*) getting an improvement of the execution time.

1.3 Structure of this Dissertation

This Thesis Dissertation is organized as follows:

Part I: Preface. The first part consists of an introductory chapter.

1. The first chapter, **Introduction**, gives an overview of the main goals pursued in this dissertation. In includes a brief explanation of the research context and motivation, as well as a short presentation of the main contributions, represented and related in a global diagram, and later detailed independently.

Part II: Background. The purpose of the second part is to describe the main concepts which should be known for a better understanding of the proposals described in subsequent parts. It consists of three chapters.

2. The second chapter, **Business Processes**, collects the main concepts about business process models, and presents background to business process verification, validation and performance techniques, detecting the aspects which have not been previously considered in the literature.

3. The third chapter, called **Diagnosability and Model-Based Fault Diagnosis**, shows the most used methodologies regarding fault identification and isolation, as well as previous works on the study of the diagnosability of systems. Since the possibility to diagnose a system depends on the available observations after its execution, this chapter starts with a review about the existing diagnosability analysis methodologies. Finally, main concepts and techniques regarding diagnosis of systems with distributed information, as in the case of business processes, are introduced.

4. The forth chapter, **Constraint Programming**, presents the main concepts and definitions regarding the Constraint Programming paradigm, which is used as the base for the implementation and automation of most of the contributions provided in this dissertation.

Part III: Contribution I. Verification of Semantic Business Process Models. This part presents the first contribution of this dissertation.

5. The fifth chapter, **Diagnosing Correctness of Semantic Business Process Models**, presents a first fully automated contribution for the diagnosis of the correctness of semantic business process models in which the semantics of activities is specified with pre and postconditions. The diagnosis is performed at design-time, using Constraint Programming and Integer Programming techniques to compute the execution instances allowed by a business process model, and verifying two kinds of correctness of business process models. As a result, a tool has been implemented.

Part IV: Contribution II. Diagnosability in Business Processes. This part includes a chapter to present our contribution in the field of diagnosability in business processes.

6. The sixth chapter, **Improving the Diagnosability in Business Process Models**, presents a contribution which addresses the diagnosability problem in business processes, providing a solution based on the determination of a test-point allocation to obtain sufficient observable data in the data flow to allow the discrimination of faults for a later diagnosis process.

Part V: Contributions III and IV. Fault Diagnosis of Business Processes at Runtime. This part consists of two chapters to presents two contributions on the diagnosis of business processes at run-time, depending on the available semantic information.

7. The seventh chapter, **Diagnosis of Business Processes based on Business Data Constraints**, presents a contribution which proposes an automatic diagnosis method for business processes whose data flow perspective of the business process model is available. The proposed diagnosis method is based on the comparison of the expected and observational model when some discrepancy is identified after the execution of process instances.

8. The eighth chapter, **Diagnosis of Business Processes based on Structural Analysis**, presents the last contribution of this thesis Dissertation, which provides a solution to diagnose faulty activities in business-to-business collaborations in the case that it is not possible to count on semantics to define the behaviour of the processes, or it cannot be modelled as a behavioural contract to be automatically treated.

Part VI: Conclusions and Future Work. Finally, this part includes two chapters to conclude the dissertation.

9. The ninth chapter, **Final Remarks**, summarizes the main conclusions which were obtained during the development of this thesis.

10. And lastly, the tenth chapter, **Future Work**, shows some future work which is intended to be addressed.

Part II

Background

Chapter 2

Business Processes

Since the current Thesis Dissertation aims to provide techniques to improve the quality of business processes, this chapter provides background regarding business process management. Specifically, it gives an overview of the main concepts about business process models, and presents background to business process verification techniques.

2.1 Concepts

Nowadays, in order to attract and retain customers and business partners, organizations need to provide their services by means of business processes (cf. Definition 2.1) with high and consistent quality.

Definition 2.1. *A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal (Weske, 2007).*

The quality of business processes can be defined as the capacity to produce and deliver quality products. This quality is usually measured by means of aspects such as accuracy or security (Heravizadeh et al., 2009). Through the verification of business process models, and the diagnosis of their incorrect behaviour at run-time, some aspects regarding the quality of the processes, often neglected, are considered and improved. Some of those aspects are the accuracy and suitability of business processes, detailed in the following.

- The capability of a business process to provide the expected results with a certain degree of precision is called the **accuracy** of the process. The verification and diagnosis of business processes ensures the obtaining of results in accordance with the expected behaviour, guaranteeing the attainment of the expected results.

- The **suitability** of a business process refers to the capacity to provide the appropriate solution to get the specified business goals. Since those business goals are defined in the model, the diagnosis of business processes improves this aspects by ensuring the correspondence between the obtained results and the expected business goals to reach.

Business processes design, automation and management tasks, included in Business Process Management (cf. Definition 2.2) (van der Aalst et al., 2002; Jablonski and Bussler, 1996), are continuously evolving in order to improve the quality and efficiency of business processes. Business Process Management can be considered as an extension of classical Workflow Management (WFM) systems and approaches.

Definition 2.2. *Business Process Management (BPM) includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes (Weske, 2007).*

These tasks are part of the business process lifecycle (Weske, 2007) (Figure 1.1), some of them coordinated by the use of a Business Process Management System (cf. Definition 2.3).

Definition 2.3. *A Business Process Management System (BPMS) is a generic software system that is driven by explicit process representations to coordinate the enactment of business processes (Weske, 2007).*

By definition, business processes are performed in a single organization, which enables their centralized control and management, known as process orchestration. Nevertheless, several orchestrations from a same or different organizations can interact with each other, performing a business-to-business collaboration, whose interactions is performed by sending and receiving messages. This is due to the execution constraints between activities from different processes, which give rise to business process choreographies (cf. Definition 2.4). In this cases, high quality of business processes is even more difficult to attain.

Definition 2.4. *A business process choreography is an interaction of a set of business processes, by means of sending and receiving messages, with the absence of a central agent to control the activities in the business processes involved.*

The growing development of business processes has stirred several communities' interest, which try to improve and facilitate the performing of the phases in the lifecycle, above all analysing and checking the correctness of the business process model, detailed in the next section.

2.2 Business Process Model

According to Weske (2007), a business process model can be defined as stated in Definition 2.5.

Definition 2.5. *A business process model consists of a set of activity models and execution constraints between them.*

The model of a business process should have a degree of formality in order to clearly specify two perspectives of the processes: the control flow and the data flow perspectives. Several approaches support the modelling of business processes. Some examples are Petri Nets (Petri, 1962) and Statechart and Activity Diagrams (Booch et al., 2005), which allow the modelling of the control flow perspective; Event Driven Process Chains (EPC) (Scheer, 2000) that enables modelling of business processes conceptually; and Business Process Model and Notation (BPMN 2.0) (OMG, 2011), which is a standard that provides graphical notation and is able to represent complex process semantics.
For the performing of the current Thesis Dissertation, BPMN 2.0 has been chosen for business process modelling, since BPMN is an international standard for process modelling accepted by the community, which is independent of any process modelling methodology. It is specifically designed to coordinate the sequence of the processes and the messages flowing between participants in the different activities. BPMN provides a common language to enable the communication between processes in a clear, complete, and efficient way.

2.2.1 Control Flow Perspective

Business process models specify the activities, with their relationships, that are performed within a single organization, or between activities of different processes participating in a business-to-business collaboration.
The interaction between activities in different processes is just through sending and receiving messages. However, within each single business process, the de-

cisions and branching of flows are modelled using gateways, also called control flow patterns. Basic patterns in BPMN 2.0 are:

- **Sequence pattern:** This is the fundamental building block for workflow processes. It represents a series of activities which are executed in turn one after the other (Fig. 2.1 a)).
- **Parallel split (AND-split):** A single thread of execution is split into two or more parallel branches which can execute activities concurrently (Fig. 2.1 b)).
- **Synchronization (AND-join):** Two or more parallel branches that converge into a single branch. The execution of the single branch begins when all the parallel branches have been enabled (Fig. 2.1 c)).
- **Exclusive choice (XOR-split):** A single branch that diverges into two or more branches. When the execution of the activity $A1$ has finished, only one of the outgoing branches is executed depending on the evaluation of a logical expression associated with a condition of the operator (Fig. 2.1 d)).
- **Simple Merge (XOR-join):** Two or more branches converging into a single branch. Each time that the execution of a branch is finished, the thread of control is passed to the single branch (Fig. 2.1 e)).
- **Multi-choice (OR-split):** A single branch that diverges into two or more branches. When the execution of the single branch has finished, the thread of control is passed to one or more branches depending on the evaluation of different logical expressions, one for each branch (Fig. 2.1 f)).
- **Structured Synchronizing Merge (OR-join):** Two or more branches converging into a single branch. It is an assumption of this pattern that the branches come from an OR-split. The single branch will be executed when all the active branches finish their execution (Fig. 2.1 g)).

2.2.2 Data-Flow Perspective

In addition to the control flow perspective to express the structure of business processes, one of the most important aspects in the business process model is the

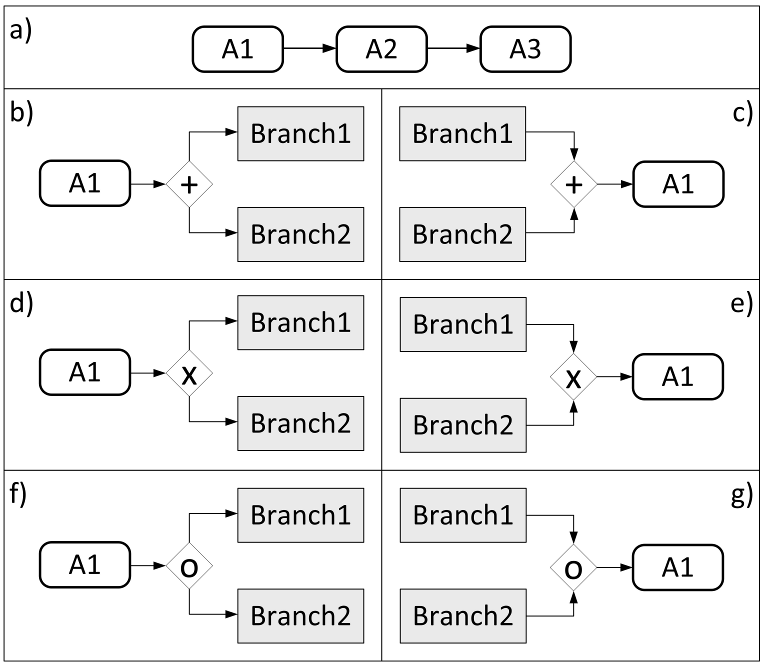


Figure 2.1: Main business process gateways

data-flow perspective. It describes which data are consumed and produced during the execution of a business process.

The data managed can be divided into observable data (data input and final data output of the overall process) and data that can remain unobservable due to privacy or security policies (intermediate data input and output of each activity). The relations between these data are expressed by means of the use of business compliance rules in the data-flow perspective.

Business compliance rules describe both the semantics of the data managed within business processes, and the contract describing the operational behaviour that should be satisfied. The basics of Business Process Management Systems is the explicit representation of business processes with their activities and the execution constraints between them. The description of the model is often insufficient to fully describe the behaviour of the process, and hence business compliance rules are added to improve the capacity of process description. Many studies propose different taxonomies to classify business compliance rules (Cetin et al., 2006; Hay et al., 2000; Ross, 2009; Sponsor, 2008). Business compliance rules can be understood as conforming to a rule such as a specification, a policy or a standardized procedure, that represent a natural step towards the inclusion of semantic requirements between business functionality and data.

The current Thesis Dissertation focuses on the use of Business Compliance Rules (Becker et al., 2011) to describe the data semantics of a business process for the representation of the relations between DataObject values: these we have named Business Data Constraints. These Business Data Constraints are understood as a subset of business compliance rules which represent the semantic relation between the data values that are introduced, read and modified during the business process instances. Since each organization defines its own rules from its business policies about things such as prices, costs, employee numbers, deadlines of tasks, and so on, it is possible to specify a different set of Business Data Constraints for each business process.

Likewise, Business Data Constraints influence the possible executions of activities. An effective means to express them is to annotate activities within a business process with pre and postconditions to specify the behaviour of each activity independently. They enable the specification of the effect on the data state for each activity.

2.3 Background to Business Process Analysis

In the last years, there is a growing interest in the analysis of business processes on the part of certain research communities. They perform Business Process Analysis (BPA, cf. Definition 2.6) to understand the properties from the business process model, using its results to check inconsistencies in the model in order to improve the management and quality of business processes. It is carried out for both perspectives of the models.

Definition 2.6. *Business Process Analysis is the activity of reviewing business processes at different stages of their lifecycle in order to ascertain how far they achieve the business objectives, ranging from model verification at design time to the monitoring of processes at runtime.*

According to both the stage of the business process lifecycle where the analysis is performed, and the aspects of the business process considered in the analysis, it is possible to classify the business process analysis methods into two types of analysis: at design-time and at runtime. The following classification is proposed, which is based on previous classifications which have been discussed in the literature (van der Aalst, 2007; Weske, 2007; Huang et al., 2008).

- During the *Design & Analysis* stage of the business process lifecycle (Figure 1.1), some verification analysis methods should be used to ensure the obtaining of a correct model.

  ▷ Verification analysis methods, which entails methods to detect syntax errors or violations in the control flow and data flow perspectives of designed business processes. This kind of analysis methods are performed at design time, in order to correct these errors, avoiding the incorrect modelling of business processes. Examples of such errors are (1) common control flow anomalies (deadlock, livelock, ...); (2) the data flow anomalies concerning, for example, read/write conflicts; and (3) data flow anomalies regarding the incorrectness of the business data constraints included in the model.

- After the process is designed and implemented, it is necessary to perform other kinds of analysis during the *Enactment* phase, when the process is in operation.

  ▷ Validation analysis methods, entailing semantic or logical conflicts in the models at design time. Examples of such errors are the violation of business compliance rules regarding execution order of the activities, and the violation of the contract describing the behaviour of the activities in a business process.

  ▷ Performance analysis methods, which focus on the quality of service of designed processes. It concerns the detection of errors in runtime, such as time-efficient conflicts, or the correct allocation of resources (both human and machine).

  ▷ Diagnosis methods, in order to determine the activity or activities which are the cause of a malfunction when, after the execution of a process instance, the behaviour of the business process does not correspond to the expected.

In the literature, it is possible to find different proposals of process analysis methods. However, as it is detailed in the next subsections, not all the aforementioned analysis methods count on previous contributions in the bibliography.

2.3.1 Verification Analysis Methods

The aim of verification analysis methods is to check the correctness of business process models, concerning the detection of syntax errors or violations in control flow and data flow perspectives. Most modelling and verification approaches in the literature only consider the control flow perspective of business process models. These contributions typically deal with errors in the modelling of the flow, detecting deadlocks, livelocks (infinite loops) lack of synchronization, and dangling reference (Karamanolis et al., 2000; Sadiq and Orlowska, 2000; Sadiq et al., 2004; van der Aalst et al., 2002; Eshuis and Kumar, 2010), most of them checking an important correctness criterion known as the soundness property, which guarantees proper termination of the business processes.
In detail, Sadiq and Orlowska (2000) proposes a visual verification approach by means of graph reduction rules which allows to discover structural conflicts in business process models.
Moreover, van der Aalst et al. (2002) and van der Aalst (2004) propose to use Petri net-based analysis techniques to diagnose workflows, providing three main types of diagnostics by means of the use of their proposed workflow-verification tool, Woflan: mismatches, locking scenarios, and coverability of threads of control.
Furthermore, the contribution in Huang et al. (2008) proposes a step-by-step conflict detecting mechanism to support process designers in analysing structural conflicts regarding livelock, starvation and lack of synchronization.
The contribution by Eshuis and Kumar (2010) introduces an approach for analysing and diagnosing workflows based on Integer Programming (IP). The diagnosis method has been implemented in a tool called DiagFlow, which reads and diagnoses XPDL models.
On the other hand, besides the structural verification regarding conflicts in the control flow perspective, when the data flow perspective is not correctly designed within the business process model, it can cause errors or conflicts, referred to as data-anomalies in Sun et al. (2004), classified into three types in Sun et al. (2006):

1. **Missing data.** It takes place when a data item is accessed before it is initialized, for four different causes: (i) a data is never assigned an initial value; (ii) an activity which uses a data item is executed before the activity which initializes it; (iii) the initialization and access are performed in parallel branches, thereby the data item may not have been initialized when it is read; and (iv) the data item is used but not initialized under certain workflow routing conditions.

2. **Redundant data.** A data item that does not contribute to the production of the final output data of the process is produced, causing inefficiency. The main cause is that a data item is produced by an activity, but never used by any activity executed after it in the process instance.

3. **Conflicting data.** There exist different versions of the same data item. It is impossible to decide which version of the data item should be considered.

Only in the last years, researchers started considering verification of workflow models with data flows. However, these approaches do not consider verification of data flow errors. Sun et al. (2004, 2006) define a data flow perspective on workflows and identify several types of errors, based on earlier work by Sadiq et al. (2004). The workflow models are abstract: they specify variables read and written per activity, but do not specify the effect of each activity by means of pre and postconditions.
Weber et al. (2010) consider verification of semantic business processes, in which activities are annotated with pre and postconditions. They detect conflicts between pre and postconditions of parallel activities and next study the reachability and executability of the activities, but only if the activities are conflict free. In their contribution, pre and postconditions are considered as CNF formulas with only boolean variables. Therefore, the approach by Weber et al. (2010) cannot verify the correctness of workflows whose activities count on pre and postconditions involving other kinds of data. Moreover, they focus on analysing the complexity of several verification tasks for semantic process models and do not focus on diagnosis of errors.
Sidorova et al. (2011) verify correctness for a subclass of Petri nets, workflow nets, extended with data operations. The workflow models they consider are abstract and need to be refined to be executable. The verification procedure checks whether the abstract workflow models can be refined into correct, concrete workflow models that have no deadlocks. The check sometimes results in a ”yes, if ...” answer, indicating that only under certain conditions a correct refinement exists.

2.3.2 Validation Analysis Methods

Validation analysis methods aim to test if business processes behaves as expected by detecting semantic or logical conflicts or violations of business compliance rules.

Previous works in compliance checking of business process models can be divided into two main goals: compliance by design in order to get correct business process designs, and compliance checking based on the verification of whether existing models were designed in accordance with the compliance rules. Extensive literature research regarding business process compliance has been presented (Sadiq et al., 2007; Namiri and Stojanovic, 2007; Ghose and Koliadis, 2007).

The contribution by Ly et al. (2009) addresses runtime compliance checking, proposing a framework which provides formal trace-based compliance criteria for static compliance validation and for dealing with process changes.

Awad et al. (2009) presents an approach to resolve execution order compliance rule violation, providing a semi-automatic solution instead of leaving all decisions in experts' hands, considering two types of execution order compliance rules (referred to as *leads to* and *precedes*).

2.3.3 Performance Analysis Methods

Performance evaluation of business processes by means of performance analysis methods is a real necessity of organizations that aim to achieve superior efficiency and competitiveness. Carrying out performance analysis on existing and planned process models offers a great way for organisations to detect flaws, bottlenecks, and other issues within their processes and allows them to make more effective decisions.

This kind of analysis methods focus on estimating key quantitative performance indicators by simulating the behaviour of business processes, such as resources allocation (human and machine) and time conflicts that violate organizational constraints (Huang et al., 2008).

One of the first works, (Zhao and Stohr, 1999), develops a framework for temporal workflow management, including turnaround-time predication, time allocation, and task prioritization.

In a related way, the contribution by (Hyun Son and Ho Kim, 2001) proposes a schema for maximizing the number of workflow instances satisfying a predetermined deadline, based on a method to determine the critical activities.

Regarding performance analysis for resource allocation, some previous works integrate scheduling tools for the enactment phase, in order to take dispatching decisions as to which activity should be executed using a resource when it becomes free.

Moreover, the work (Ha et al., 2006) proposes a set of process execution rules based on individual worklists, and develops algorithms for the task assignment in order to maximize the overall process efficiency, while taking resource capacities into account.
Recently, the work (Tsai et al., 2010) proposes distributed server architecture for the management of the BP workflow, and presents techniques for the dynamic allocation of the resources.
Furthermore, (Thompson and Goodale, 2006) addresses the scheduling of a group of employees which present different productivity considering the stochastic nature of customer arrivals and replans during run-time when estimates are incorrect

2.3.4 Discussion

Upon the study of previous contributions in the literature (a significant part of them has been aforementioned), it is concluded that, to the best of our knowledge, some necessary aspects in Business Process Analysis have not been previously addressed by other authors so far. In detail, Table 2.1 collects the mapping relations showing the Business Process Analysis aspects, detailed in the following:

- *Verification of control flow*: concerning the detection of errors in the modelling of the flow, such as deadlocks, livelocks (infinite loops) lack of synchronization, and dangling reference, checking the soundness property (guaranteeing proper termination).

- *Verification of data flow*: concerning the detection of data-anomalies, classified into *missing data*, *redundant data* and *conflicting data*.

- *Verification of data flow values*: concerning the detection of data flow anomalies regarding inconsistencies between the data flow perspective and the domain (i.e. possible values) of the data. It is possible to detect when business data constraints and/or annotations in the activities are available.

- *Validation of process compliance*: concerning the validation of business process compliance by design in order to get correct business process designs, and compliance checking based on the verification of whether the control flow and data flow perspectives of a model were designed in accordance with the defined business compliance rules.

- *Validation of data flow values*: concerning the detection at runtime of the violation of the contract -by means of business data constraints- that describes the behaviour of the activities in a business process.

- *Performance (time & resources)*: concerning methods focused on the quality of services of designed processes, related to errors in runtime such as time-efficient conflicts and allocation of resources.

- *Diagnosis of activities*: concerning methods to determine the activity or activities which are responsible for a malfunction, at runtime, either with data flow perspective available to define business contracts (quantitative information) or only counting on the control flow perspective (qualitative information).

Table 2.1 enables to easily notice the aspects which have not been previously considered in the literature. The verification, validation and diagnosis of those aspects compose the aim of the current Thesis Dissertation, in order to improve overall Business Process Management Systems functionalities by applying new analysis methods at different stages of the business process lifecycle in efficient ways.
In order to perform this analysis methods, a review of the existing diagnosis techniques in the literature has been performed, selecting as a base for our work the Model-based Fault Diagnosis methodologies due to the verification, validation and diagnosis of business processes can be performed by the comparison of the expected behaviour of the processes (the business process model) with the observed behaviour (the observational model after a process instance is executed). To this end, the techniques used in Model-Based Fault Diagnosis methodologies can be adapted and used to detect and automatically verify, validate and diagnose business processes. Therefore, the state of the art regarding Model-based Fault Diagnosis is discussed in Chapter 3.
Likewise, in order to model, implement and automate the proposed diagnosis methodologies, Constraint Programming is chosen due to its efficiency and expressiveness when it comes to modelling problems which include both topological aspects (i.e. the control flow perspective of the business process models) and numeric data (i.e. data and values participating in the definition of the model as business data constraints). To this end, Chapter 4 presents the main concepts and definitions regarding Constraint Programming.

Table 2.1: Mapping of the Business Process Analysis methods addressed

	Verification			Validation		Performance	Diagnosis
	Control flow	Data flow	Data flow values	Process Compliance	Data flow values	Time & Resources	Activities
Karamanolis et al. (2000)	✓						
Sadiq and Orlowska (2000)	✓						
van der Aalst et al. (2002)	✓						
van der Aalst (2004)	✓						
Huang et al. (2008)	✓					✓	
Eshuis and Kumar (2010)	✓						
Sun et al. (2004)		✓					
Sun et al. (2006)		✓					
Sadiq et al. (2004)		✓					
Weber et al. (2010)		✓					
Sidorova et al. (2011)		✓					
Sadiq et al. (2007)				✓			
Namiri and Stojanovic (2007)				✓			
Ghose and Koliadis (2007)				✓			
Ly et al. (2009)				✓			
Awad et al. (2009)				✓			
Zhao and Stohr (1999)						✓	
Hyun Son and Ho Kim (2001)						✓	
Ha et al. (2006)						✓	
Tsai et al. (2010)						✓	
Thompson and Goodale (2006)						✓	

Chapter 3

Diagnosability and Model-Based Fault Diagnosis

The current Thesis Dissertation aims to provide methods to improve the quality of business processes by determining the responsible for a malfunction, in an automatic and efficient way. Model-based diagnosis can be used to know if the behaviour of a business process is correct or not, and which are the activities that are not working correctly. It can be performed by the comparison of the expected behaviour of the processes (the business process model) with the observed behaviour (the observational model after a process is executed). To this end, the techniques used in Model-Based Diagnosis methodologies are adapted so that they can be applied to business processes, enabling the automatic diagnosis of faults when an error is detected after the execution of a correctly modelled business process.

The aim of fault diagnosis is to improve the reliability, security, efficiency, maintainability of any system. A fault diagnosis system is used to detect, isolate and determine the location of a fault in a monitored system. Thereby, the monitoring should be a reflect of the real behaviour of the system and the produced deviations from the expected behaviour. The diagnosis enables the identification of the parts which fail, determining why a system correctly designed does not work as expected. The explanation of that abnormal behaviour, from a determined observation, is the main goal of the diagnosis.

Since the capacity to diagnose a system depends on the available observations, this chapter starts with a review about the diagnosability analysis methodologies to consider in order to decide the flows where a business process should be monitored. The monitored values of the data flow compose the observational model and make the business processes diagnosable. Then, this chapter gives an overview of

Model-Based Diagnosis principles, presenting some concepts and techniques that are used or adapted in the contributions proposed in this document. In short, since the model of the business processes to diagnose can count on either semantic information or only structural information, two classic techniques based on this types of models are detailed. And finally, since the information of a business process can be distributed in a semantic of physical way (as in business-to-business collaborations), main concepts and techniques regarding diagnosis of systems with distributed information are introduced.

3.1 Diagnosability

The monitoring of a system, and later diagnosis, are based on observations, which provide information about the behaviour of the system. For the later diagnosis process to be successful, diagnosability analysis becomes a design-time requirement to diagnose in run-time. The diagnosability level of the system depends upon the observations.

If the diagnosis is based on few observations, or if observations are not allocated at the most convenient places, it is very difficult to distinguish which parts of the system are failing. Both the number of observations and the location where they are performed enable the source of the problem to be precisely located. In general, diagnosis systems not only incorporate monitoring, but also the identification and isolation of faults. The explanation of abnormal behaviour, from a determined observation derived from the monitoring, is the main task of diagnosis.

The study of the diagnosability of a system corresponds with the study of whether the system can be diagnosed based on the available observations, provided by sensors allocated at strategic places of the system. The diagnosability determines the number of faults that is possible to diagnose and distinguish with the available sensors. This is a property that is crucial at some stages, for instance during the design of the system, in order to decide the number and locations of the sensors to place.

The approach in Console et al. (2000) presents an study of the diagnosis and diagnosability of a system by using process algebra, more precisely Performance Evaluation Process Algebra (PEPA), within the model-based diagnosis field. The mentioned contribution adopts the next definition for full diagnosability:

Definition 3.1. *A system is diagnosable with a given set of sensors if and only if:*

- *for any relevant combination of sensors readings there is only one minimal candidate diagnosis; and*
- *all faults of the system correspond to a candidate diagnosis for some sensor reading.*

Travé-Massuyès et al. (2001) apply diagnosability and model-based allocation of sensors. They stated that a system may be partially diagnosable, having different levels of diagnosability, defining the discriminability of faults as follows:

Definition 3.2. *A fault F1 is discriminable from another fault F2 if and only if exists at least a sensor reading where F1 appears in some minimal diagnosis candidate but not F2, and vice versa.*

According to this definition, the diagnosability level can de determined as stated in Definition 3.3.

Definition 3.3. *The diagnosability level of a system is the quotient of the number of faults which can be discriminated from each other, and the number of all possible faults. If the number of components of a system is called nComp, and since only one fault per component is possible, the maximum number of possible faults is initially* $2^{nComp} - 1$.

The diagnosability problem has been analysed in previous work, but, to the best of our knowledge, it has never been adapted to business processes. Moreover, not only is an analysis of the diagnosability required, but an improvement of the diagnosability is also necessary. Some works in the literature are detailed in the following bullet points:

- Some previous work regarding diagnosability analysis are performed in Bocconi et al. (2007) and Console et al. (2000). In detail, the proposal by Bocconi et al. (2007) presents an approach to compute diagnosability for a decentralized framework to diagnose Web Services. It performs an incremental analysis, refining the results at each step of the process, and focused on discrete event systems.
- With regard to work related in the field of the allocation of sensors in systems composed of many components, a certain number of these systems fail to perform the placement in accordance with the diagnosability (Madron and Veverka, 1992; D. Maquin and Ragot, 1997; C. Commault and Agha, 2006).

Although test-point allocation is studied in the proposal by Narendra et al. (2008), it is not focused on diagnosability, but on the problem of continuous compliance monitoring at run time in order to prevent non-compliance against policies (security, confidentiality, and data integrity). The selection of locations is useless from the diagnosis point of view since no diagnosability criterion is taken into account.

- Other proposals allocate the sensors in order to improve diagnosability. A selection of these studies are given in more detail below.

  In Travé-Massuyès et al. (2006), an analysis of many systems is performed in order to study their physical models. This study takes into account the diagnosability criteria by means of a technique that allocates sensors consecutively.

  The proposal by Spanache et al. (2004) allocates sensors according to an economic criterion, and presents a method to obtain an optimal-cost sensor system for a certain degree of diagnosability.

  The contribution in Zhang et al. (2009) allocates test points in circuits and physical systems using a genetic algorithm. Nevertheless, since the signal propagation in this kind of system flows in a different way to that of the data flow of a business process, this method is inapplicable in the context discussed in this Thesis Dissertation.

  In the contribution by Ceballos et al. (2005), a technique is proposed in order to improve the computational complexity for the isolation of faults within a system. The method is based on the addition of the minimal set of sensors. A CSP (Rossi et al., 2006; Dechter, 2003) is obtained in order to select the necessary sensors to guarantee the problem specification, and an algorithm for the determination of the bottleneck locations of the system is developed, in order to improve the computational complexity of the CSP.

- Finally, there are some works which uses other kind of techniques to study or improve the diagnosability of systems.

  The contribution presented by Frisk and Krysander (2007) shows an efficient technique that is based on the Dulmage-Mendelsohn decomposition introduced in Dulmage and Mendelsohn (1959). Although the constraint models of most practical systems are under-determined before taking the sensors into account, and over-determined afterwards, this method only applies to just-determined sets of constraints.

A method based on the study of structural matrices is presented in Yassine et al. (2008), where each component of the system is represented by a constraint. Regarding detectability and diagnosability, all subsets of constraints are considered (diagnosable, discriminable, and detectable). The contribution by Yassine et al. (2010) presents a continuation of Yassine et al. (2008), where the Dulmage-Mendelsohn decomposition (Dulmage and Mendelsohn, 1959) together with a combinatorial algorithm are used in the search for the optimal solution.

3.2 Model-Based Diagnosis

In the last decades, model-based diagnosis has become the most extensive research area in the diagnosis field. The reasoning is carried out from a model which represents the system to diagnose in an explicit way. A fault exists when the observed behaviour does not correspond with the behaviour derived from the model. This model comes from the knowledge of the system. The component responsible for the fault is identified with a later analysis of the discrepancies.
Model-based diagnosis is based on the comparison between the available observations about the operation of a system and the predictions made from the model of the system. The observations indicate how the system is behaving, whereas the model expresses how should it behave during a correct execution.
When a symptom is detected, that is, a discrepancy between the observed and expected is detected, it is deduced that at least one of the components involved in it is not working correctly. The description of the systems, done by the models, uses the relations between inputs and outputs. Most of the approximations for components characterize the diagnosis of a system as a collection of minimal sets of components that fail to explain the observed behaviour (symptoms). That is why it is important to count on a detailed model to determine the diagnosis of a system. With this kind of models, it is possible to diagnose quickly the main parts of the systems.
There are two distinct and parallel research communities that work on model-based diagnosis.

- *Fault Detection and Isolation (FDI)* community, which bases the foundations of its solution approaches on engineering disciplines such as control theory and statistical decision making.

- *Diagnosis (DX)* community, which bases the foundations of its solution approaches on the fields of computer science and artificial intelligence.

Each community has developed its own terminology, tools, techniques, and approaches to solve diagnosis problems. In the past, these two communities have only had a little communication between them, but since a few years ago, a growing number of researchers have tried to understand and incorporate approaches from the parallel research fields to build better and more effective diagnostic systems (Biswas et al., 2004; Cordier et al., 2000).

The pioneer work Davis (1984) within the DX community presents an approximation that allows to perform the diagnosis of systems using their structure and behaviour. DART Genesereth (1984) and GDE Kleer and Williams (1987) were the first implementations for diagnosis, which use different inference ways to detect the possible failures. The works of Reiter (1987) and de Kleer et al. (1992) introduce the basic definitions and foundations of diagnosis. To explain the discrepancies between the observed and the correct behaviour A general theory was proposed, using a logical-based diagnosis process.

There are diagnostic models that use models which require the development of fault models together with the model of normal operation. To build fault models in a system, it is useful when the failures are well-known and easy to model, but it also limits the diagnosis process to known failures. A revision of the approximations about the automation of the diagnosis tasks can be found in Dressler and Struss (2003), and for a discussion about the applications of the Model-Based Diagnosis it can be consulted in Console and Dressler (1999). The generation of the diagnosis based on consistency to cover dynamic systems was proposed by Heller and Struss (2001).

Related to FDI community, Staroswiecki and Declerk (1989) and Cassar and Staroswiecki (1997) present the formalization of the process to obtain the ARRs (Analytical Redundancy Relation) of the system. The obtaining of the ARRs is based on searching the overdetermined systems in which it is possible the detection and location of the faults. The FDI methodology allows an off-line analysis of a part of the work, in contrast with the DX methodology where the work is almost completely on-line. Fattah and Holzbaur (1994) propose an approximation to the FDI methodology, but using logic constraints models to solve the diagnosis problem.

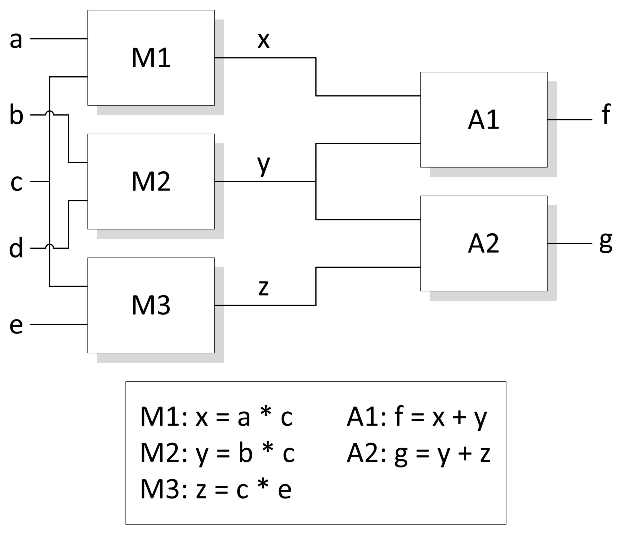


Figure 3.1: Polybox system

Example of analysis

The example that has been chosen to support the explanations is the well-known system from Davis (1984) composed of three multipliers and two adders referred as the *polybox example* (Figure 3.1).
This example is used in the rest of this chapter to clarify the explanations about the DX and FDI methodologies.

3.2.1 FDI: Analytical Redundancy Approach

For the FDI approach, the behavioural model (BM) is derived from its structure, which shows the connections between the components and the behaviour of each component. For this reason, the FDI methodologies are useful when it comes to diagnose business processes where only the control flow perspective of the model is available.

Definition 3.4. *A System Model (SM) is defined as the behavioural model (BM), i.e., the set of relations of the model together with the observation model (OM).*

For the polybox example in Figure 3.1, the system model is:

- $BM : \{RM1 : x = a * c;\ RM2 : y = b * d;\ RM3 : z = c * e;\ RA1 : f = x + y;\ RA2 : g = y + z\}$

- $OM : \{RSa : a = a_{obs};\ RSb : b = b_{obs};\ RSc : c = c_{obs};\ RSd : d = d_{obs};\ RSe : e = e_{obs};\ RSf : f = f_{obs};\ RSg : g = g_{obs};\}$

Definition 3.5. *A diagnosis problem is defined by a system model (SM), a set of observations (OBS) and a set of faults (F).*

For the polybox example in Figure 3.1:

- $OBS = \{a_{obs} = 2;\ b_{obs} = 2;\ c_{obs} = 3;\ d_{obs} = 3;\ e_{obs} = 2;\ f_{obs} = 10;\ g_{obs} = 12\}$
- Set of simgle faults (SF): $\{F_{A1}, F_{A2}, F_{M1}, F_{M2}, F_{M3}\}$, being $F = 2^{SF}$.

Definition 3.6. *The structure of a system is defined by means of a binary application $S : SM \times V \to \{0,1\}$, where $V = X \cup O$ is the set of variables and $s(rel, v) = 1$ iff v appears in the relation rel.*

Definition 3.7. *An Analytical Redundancy Relation (ARR) is a relation established by the SM which contains only observed variabls and which can therefore be evaluated by OBS. It is noted $r = 0$, where r is the residual of the ARR. For a given OBS, the instantiation of the residual is noted by $val(r, OBS)$, referred to as $val(r)$, which is equal to 0 if the observations for the ARR are satisfied.*

The ARRs can be obtained from the SM by eliminating the unknown variables. For the polybox example in Figure 3.1, the next two ARRs can be obtained:

- $ARR_1 : r1 = 0$, where $r1 \equiv f_{obs} - a_{obs} * c_{obs} - b_{obs} * d_{obs}$
- $ARR_2 : r2 = 0$, where $r2 \equiv g_{obs} - b_{obs} * d_{obs} - c_{obs} * e_{obs}$

Assuming sensors are fault-free, these two ARRs can be rewritten as:

- $ARR_1 : f - (a * c + b * d) = 0$
- $ARR_2 : g - (b * d + c * e) = 0$

Additional ARRs can be obtained through combination of the elementary ARRs. In this case, subtracting ARR_2 from ARR_1:

- $ARR_3 : f - g - a * c + c * e$

ARR	F_{A1}	F_{A2}	F_{M1}	F_{M2}	F_{M3}
1	1	0	1	1	0
2	0	1	0	1	1
3	1	1	1	0	1

Table 3.1: Signature matrix for single faults

It is important to notice that the components involved in this third ARR are not a combination of the components used in the previous ARRs.

Definition 3.8. *Given a set $R = \{ARR_1, ARR_2, \ldots, ARR_n\}$ composed of n ARRs and a set $F = \{F_1, \ldots, F_m\}$ with m faults, the fault signature F_j is given by the binary vector $FS_j = [s_{1j}, .., s_{nj}]^T$ where s_{ij} is given by $(ARR_i, F_j) \rightarrow s_{ij} = 1$ if at least a component from F_j is involved in ARR_i, and $(ARR_i, F_j) \rightarrow s_{ij} = 0$ otherwise.*

This specifies if the occurrence of the fault affects to the corresponding ARR.

Definition 3.9. *Given a set R composed of n ARRs, the signature of a set of m faults is called signature matrix.*

The signature matrix for single faults (SF) for the polybox example is shown in Table 3.1.
For multiple faults it is only necessary to add the fault combinations as columns in the signature matrix.
The diagnosis of the FDI approach is based on the interpretation of the columns in the signature matrix, and lies in comparing the observation signature with the fault signature. This decision is treated as a decision-making problem.

Definition 3.10. *The observation signature OS is a binary vector $OS = [OS_1, .., OS_n]^T$ where $OS_i = 0$ iff $val(r_j, OBS) = 0$.*

The first step (detection phase) lies in building the observation signature. For the polybox example in Figure 3.1, the observation signature is $OS = [1, 0, 1]^T$.
Then, in the second step (isolation phase) the observation signature and the fault signature are compared. The result to this decision is to define a fault criterion.
For the polybox example:
$OS = [1, 0, 1]^T \leftrightarrow F_{A1}$ *or* F_{M1} *or* $F_{\{A1,M1\}}$
In case of $f = 10$ and $g = 10$:
$OS = [1, 1, 0]^T \leftrightarrow F_{M2}$

3.2.2 DX: Logical Diagnosis Approach

Reiter (1987) propose a logical theory for the diagnosis, known as consistency-based diagnosis. This theory was extended and formalized later by de Kleer et al. (1992), and can be used or adapted for the verification, validation and diagnosis of business processes whose behaviour is modelled by means of a semantic contract (business rules).
For the DX community, the system model is defined as follows:

Definition 3.11. *A system model (SM) is composed of the pair (SD, COMPS), where SD (system description) is a set of logic equations; and COMPS is a finite set of components. The system description uses the predicate AB to represent the abnormal behaviour of the system, and $\neg AB(c)$, where c is a component, to represent the correct behaviour of c.*

For the polybox example in Figure 3.1:

- $COMPS = \{A1, A2, M1, M2, M3\}$
- $SD = \{ADD(x) \wedge \neg AB(x) \rightarrow Output(x) = Input1(x) + Input2(x),$
  $MULT(x) \wedge \neg AB(x) \rightarrow Output(x) = Input1(x) * Input2(x),$
  $ADD(A1),\ ADD(A2),\ MULT(M1),\ MULT(M2),\ MULT(M3),$
  $Output(M1) = Input1(A1),\ Output(M2) = Input2(A1),$
  $Output(M2) = Input1(A2),\ Output(M3) = Input2(A2),$
  $Input2(M1) = Input1(M3)\}$

The diagnosis problem lies in the discrepancy between the set of available observations and the correct behaviour of the system described in the model.

Definition 3.12. *A set of observations OBS is a set of first order predicates.*

For the polybox example:
$$OBS = \{Input1(M1) = 2,\ Input2(M1) = 3,\ Input1(M2) = 2,\ Input2(M2) = 3,\ Input2(M3) = 2,\ Output(A1) = 10,\ Output(A2) = 12\}$$

Definition 3.13. *A diagnosis problem is represented by $\{SD, COMPS, OBS\}$, where $\{SD, COMPS\}$ is the system model and OBS is a set of observations.*

Definition 3.14. *A diagnosis for $\{SD, COMPS, OBS\}$ is a set of components $\mathcal{D} \subseteq COMPS$, so that $SD \cup OBS \cup \{AB(c) | c \in \mathcal{D}\} \cup \{\neg AB(c) | c \in COMPS - \mathcal{D}\}$ is satisfied. A minimal diagnosis is a diagnosis $\mathcal{D}$ so that $\forall \mathcal{D}' \subset \mathcal{D}$, $\mathcal{D}'$ is not a diagnosis.*

In order to attain the minimal diagnoses, Reiter (1987) propose the concept of conflict to generate them, being that concept the base for most DX approaches.

Definition 3.15. *A R-conflict for $\{SD, COMPS, OBS\}$ is a set of components $C = \{c_1, c_2, \ldots, c_n\} \subseteq COMPS$, so that $SD \cup OBS \cup \{\neg AB(c) \mid c \in C\}$ is inconsistent. A minimal R-conflict is a R-conflict which does not contains any other R-conflict.*

Definition 3.16. *Hitting set is a set of components which intersect the minimal R-conflicts. The minimal hitting set is a set that includes a component from each minimal R-conflict.*

Using the concepts of minimal R-conflict and minimal hitting set, the minimal diagnosis can be formalized as follows:

Proposition 1. *$\mathcal{D}$ is a minimal diagnosis for $\{SD, COMPS, OBS\}$ iff $\mathcal{D}$ is a minimal hitting set in the collection of minimal R-conflicts of $\{SD, COMPS, OBS\}$.*

For the polybox example in Figure 3.1, with $f = 10$ and $g = 12$, there are four minimal diagnoses given by the minimal hitting sets $\{\{A1, A2, M1, M3\}, \{A1, M1, M2\}\}$, which are: $\mathcal{D}_1 = \{A1\}$, $\mathcal{D}_2 = \{M1\}$, $\mathcal{D}_3 = \{A2, M2\}$, $\mathcal{D}_4 = \{M2, M3\}$.

3.3 Distributed Diagnosis

Since the information defining business process models can be distributed in a semantic or physical way, it is necessary to introduce the main concepts in distributed diagnosis. Fault diagnosis can be classified in accordance with either the methodology used to perform the diagnosis process, or the degree of distribution of the knowledge of the system to diagnose (centralized, decentralized or distributed diagnosis). The classification is as follows:

- *Centralized diagnosis.* The knowledge of the overall system is available, since it is running on an only node.
- *Decentralized diagnosis.* The diagnosis uses a central coordination process (supervisor) and a local diagnoser for each subsystem.

- *Distributed diagnosis.* The model is distributed, using only a local diagnoser for each subsystem instead of a global coordination process. Each diagnoser communicates directly with other diagnosers, avoiding the bottleneck derived from the access to the supervisor.

Currently the hardware and software systems are generally composed of a large number of distributed subsystems (or components) that interact with the physical world via a set of users, sensors or actuators. Examples of such systems are ad-hoc and mesh wireless networks, cars, industrial systems, Web Services, and, such as the case of study of this dissertation, business processes.
This is also the case of business processes, where different organizations perform collaborations and interactions in a coordinated way. This gives rise to distributed scenarios which need to be control and manage by means of process choreographies due to the overall system and knowledge remains distributed and unobservable in a global way.
As it was aforementioned, fault diagnosis enables the determination of why as system correctly designed does not work as it is expected. Therefore the main objective of fault diagnosis is to identify the parts which fail in a system, for this reason it proposes to find out the discrepancies between the observed and correct behaviour of a system, with the added difficulty of doing the same with distributed systems.
The first diagnosis tool could be considered as only one diagnostic agent (diagnoser) with a model of the overall system to be diagnosed. However, the total integration of knowledge into one model of the complex, dynamic or distributed systems is infeasible. In some systems, the knowledge integration can proceed from different local diagnostic processes situated in different subsystems (it is called spatially distributed) or from different fields of expertise (it is called semantically distributed) (Frohlich et al., 1997).
As it was seen at the beginning of this chapter, the distributed diagnosis can be classified in decentralized or distributed, depending on the degree of distribution of the diagnosers associated to the subsystems and the supervision process of them. In the following the most important works in this field are presented, classified as decentralized or distributed approaches.

3.3.1 Decentralized Approaches

- Diagnostic reasoning should identify diagnoses, as assignments of behaviour modes to components, for a given set of observation. A diagnostic engine should, in general, explore the space of candidate diagnoses and perform discrimination among alternative candidates, possibly suggesting additional pieces of information to be acquired to this purpose. For reasoning on global failures of the overall service, it has been proposed (Ardissono et al., 2005) to:

  - Associate with each basic service a local diagnoser, owning a description of how the service is supposed to work; the role of local diagnosers is to provide the global diagnoser with the information needed for identifying causes of a global failure.
  - Provide a global diagnoser which is not tied to any specific service, but is able to invoke local diagnosers and relate the information they provide, in order to reach a diagnosis for the overall complex service. In case the supply chain has several levels, several global diagnosers may form a hierarchy, where a higher level global diagnoser sees the lower level ones as local diagnosers.

  Each local diagnoser interacts with its own Web Service and with the global diagnoser. The global diagnoser interacts only with local diagnosers. It is used for fault tolerant Web Services (Ardissono et al., 2006).

- Another decentralized approach to diagnosis has been proposed by Pencolé and Cordier (2005). The application (telecommunication networks) is significantly different from previous work, posing a very different problem. In previous case, an alarm may be raised in a point that is far away from the failure source. In their case, a failure causes a chain of alarms, the first of which points to the failure source. However, due to the distributed nature of the network, the order in which alarms are received is not the same in which they are raised, thus the problem of finding the failure source.

- In Console et al. (2007) a framework for decentralized model-based diagnosis of complex systems is proposed. The case considered is where subsystems are developed independently along with their associated (or embedded) software modules - in particular their diagnostic software. This is useful in those situations where subsystems are developed (possibly by different suppliers) without a-priori knowledge of the system in which they will be exploited, or without making assumptions on the role they will play in such system. A decentralized architecture is described, where subsystems are analysed by Local Diagnosers, coordinated by a Supervisor. Within the framework, both the Local Diagnosers and the Supervisor can be designed independently of each other, without any advance information on how the subsystems will be connected (provided that they share a common modelling ontology) and allowing also for runtime changes in the overall system structure. Local diagnosers are thus loosely coupled and communicate with the Supervisor via a standard interface, supporting independent implementations.

3.3.2 Distributed Approaches

- In Frohlich et al. (1997) an agent-based framework for the diagnosis of spatially distributed systems is introduced. The motivation for such a framework is the unnecessary complexity and communication overhead of centralized solutions. Consider a distributed system with n nodes, e.g. a computer network consisting of n machines. When using a centralized diagnosis system the size of the system description (i.e. number of ground formulas) is linear in n. Diagnosis time will usually be worse than linear in n Mozetic and Holzbauer (1993). Also all observations have to be transmitted to the central diagnosis machine, causing a large communication overhead.

  This agent-based approach decomposes a system into a set of subsystems. Each subsystem is diagnosed by an agent which has detailed knowledge over its subsystem and an abstract view of the neighbouring subsystems. Most failures can be diagnosed locally within one subsystem. This decreases diagnosis time dramatically in large systems. In the case of the computer network most machines in a subnet can usually fail without affecting machines in other subnets. Only those computers in other subnets can be affected which have sent messages to the faulty machine. Moreover, the local computation of diagnoses avoids the communication overhead which would be needed to forward all observations to the central diagnosis engine.

  Failures which affect more than one subsystem are diagnosed by the agents cooperating with each other. The cooperation process is triggered locally by an agent, when it realizes that it can not explain the observations by a failure in its own subsystem. The cooperation process is guided by a small amount of topological information.

- In Provan (2002) a new technique for diagnosing distributed systems using a model-based approach is proposed. It is assumed that a system consisting of a set of inter-connected components exists. Each one of the components computes a local (component) diagnosis. It is adopted the structure-based diagnosis framework of Darwiche (1998) for synthesizing component diagnoses into globally-sound diagnoses, where we obtain the structure from the component connectivity. Unlike previous approaches that compute diagnoses using the system observations and a system description Darwiche (1998) Kleer and Williams (1987), it transforms the component diagnosis synthesis into the space of minimal diagnoses. Assuming that each component can compute a local minimal diagnosis based only on sensors internal to that component and knowledge only of the component system description, it describes an algorithm that guarantees a globally sound, complete and minimal diagnosis for the complete system. This algorithm uses as input the directed graph (digraph) describing the connectivity of distributed components,with arc directionality derived from the causal relation between the components. Given that real-world graphs of this type are either tree-structured or can be converted to tree-structured graphs, it is proposed a graph-based message-passing algorithm which passes diagnoses as messages and synthesizes local diagnoses into a globally minimal diagnosis in a two-phase process. By compiling diagnoses for collections of components (as determined by the graph's topology), it can significantly improve the performance of distributed embedded systems. It is shown how this approach can be used for the distributed diagnosis of systems with arbitrary topologies by transforming such topologies into trees.

  One important point to highlight is that this approach synthesizes diagnoses computed locally, and places no restriction on the technique used to compute each local diagnosis (e.g., neural network, Bayesian network, etc.), provided that each local diagnosis is a least-cost or most-likely diagnosis. The synthesis approach takes this set of self-diagnosing sub-systems, together with the connectivity of these sub-systems, to compute globally-consistent diagnoses.

- In Roos et al. (2003) it is analysed the problem of multi-agent diagnosis when knowledge is semantically distributed over the agents. Especially the case that the agents' knowledge concerning the faulty behaviour of some components, is incorrect has been considered. A solution based on an abstraction hierarchy on the fault modes is proposed in the paper, and a protocol for determining the global diagnoses with a minimal number of broken components is given.

A knowledge distribution over multiple agents induces a division of a system S into several subsystems. In the case of a semantic knowledge distribution, each agent makes diagnosis of a different aspect of the system S.

- The contribution in Biteus et al. (2008) is a method that calculates the diagnoses with minimal cardinality in a distributed system. A distributed system consists of a set of agents, where an agent is a more or less independent software entity. The diagnoses can, in distributed systems, be divided into two different levels, global diagnoses that are diagnoses for the complete distributed system and local diagnoses that are diagnoses for a single agent. The method designed in Biteus et al. (2008) first calculates the set of minimal local diagnoses in each agent. These sets of minimal local diagnoses are then used to calculate the set of global diagnoses with minimal cardinality.

  The work in Biteus et al. (2008) was inspired by diagnosis in distributed embedded systems used in automotive vehicles and especially that in a heavy-duty vehicle from Scania. These systems typically consist of precomputed diagnostic tests that are evaluated in the different agents, which in the automotive industry correspond to electronic control units (ECUs). The results from the diagnostic tests can be used to calculate the sets of local diagnoses in the agents. These embedded distributed systems typically consist of ECUs with both limited processing power and limited RAM memory. Therefore, the method designed in Biteus et al. (2008) calculates the global diagnoses with minimal cardinality in a cooperation between the agents, such that the computational expensive tasks are distributed between the different agents.

- The approach by Roychoudhury et al. (2009) developed a systematic model-based approach to distributing the diagnosis task by designing multiple diagnosers that operate independently and generate globally correct diagnoses. A model-based fault diagnosis scheme for continuous systems is developed, where the local diagnosers are generated off-line. At runtime, the local diagnosers operate independently to generate local diagnosis results that are globally correct. This approach does not require a coordinator, and there is minimal or no exchange of information among the diagnosers.

Chapter 4

Constraint Programming

In the two previous chapters, main concepts regarding Business Process Management and Model-based Fault Diagnosis have been introduced. Since the aim of the current Thesis Dissertation is the application of Model-based Fault Diagnosis techniques for the automatic diagnosis of business processes, this chapter presents the main concepts related to the Constraint Programming paradigm, which is used as support for the implementation and automation of the proposed diagnosis methodologies due to its efficiency and expressiveness when it comes to modelling problems which include both topological aspects and numeric data. This is the case of business processes, where it is necessary to consider both control flow and data flow perspectives, being necessary to count on a technique to model business data constraints (including the numeric values of the data).

Constraint Programming is based on the resolution of Constraint Satisfaction Problems (CSPs), which are problems where an assignment of values to variables should be found in order to satisfy a number of constraints. A large number of problems in Artificial Intelligence and other areas of Computer Science can be seen as special cases of Constraint Satisfaction Problems. Some examples are belief maintenance, scheduling, temporal reasoning, graph problems, configuration problems, etc.

A number of different approaches to solve these problems have been developed. Some of them use constraint propagation to simplify the original problem. Others use backtracking to directly search for possible solutions. Some are a combination of these two techniques. In general, a CSP is composed of a set of variables, a domain for each variable, and a set of constraints. Each constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take. The goal is to find one assignment to the variables such that the assignment satisfies all the constraints. In some problems, the goal is to find all such assignments Kumar (1992).

4.1 Basic Concepts in CSP Modelling

Constraint Satisfaction Problems are problems that require the assignment of values to variables according to some constraints. Formally, a finite CSP is defined as follows.

Definition 4.1. *A Constraint Satisfaction Problem is defined by a triple (X, D, C), where:*

- $X = \{x_1, \ldots, x_n\}$ *is a finite set of n variables.*
- $D = \{d_{x_1}, \ldots, d_{x_n}\}$ *is a collection of finite domains related to each variable* x_i. d_{x_i} *is the finite set of values that can be assigned to the variable* x_i.
- *C is a set of constraints among variables. A constraint* c_i *is a tuple* (W_i, R_i)*, where* R_i *is a relation defined over the subset of variables* $W_i \subseteq X$*, such that* $R_i \subseteq D_{i1} \times \ldots \times D_{ik}$*. The arity of a constraint is the number of variables which compose the constraint.*

The search of solutions for a CSP is based on the instantiation concept. An assignment of a variable, or instantiation, is a pair variable-value (x, a) which represents the assignment of the value a to the variable x. An instantiation of a set of variables is a tuple of ordered pairs, where each sorted pair (x, a) assigns the value a to the variable x. A tuple $((x_1, a_1), \ldots, (x_i, a_i))$ is consistent if it satisfies all the constraints formed by variables of the tuple.
A solution to a CSP is an assignment of values to all the variables in order that all the constraints must be satisfied. That is to say, a solution is a consistent tuple which contains all the variables of the problem. A partial solution is a consistent tuple which contains some of the variables of the problem. A problem is consistent if it exists, at least, a solution, i.e., a consistent tuple.
The techniques used in constraint satisfaction depend on the kind of constraints being considered. Often used are constraints on a finite domain, to the point that constraint satisfaction problems are typically identified with problems based on constraints on a finite domain. Such problems are usually solved via search, in particular a form of backtracking or local search. Constraint propagation are other

methods used on such problems; most of them are incomplete in general, that is, they may solve the problem or prove it unsatisfiable, but not always. Constraint propagation methods are also used in conjunction with search to make a given problem simpler to solve. Other considered kinds of constraints are on real or rational numbers; solving problems on these constraints is done via variable elimination or the simplex algorithm.

4.1.1 Consistency

One of the main difficulties in CSP resolution is the appearance of local inconsistencies. Local inconsistencies are values of the variables that cannot take part of the solution because they do not satisfy any consistency property. Therefore if any consistency property is forced, we can remove all the values which are inconsistent in regard to the property. But it can be possible that some values which are consistent in regard to a property are inconsistent in regard to another property at the same time. Global consistency implies that all values which can't take part in a solution can be removed. The constraints of a CSP generate local inconsistencies because they are combined. If the search algorithm does not store these inconsistencies, it will waste time and effort trying to carry out instantiations which have already been tested.

4.1.2 Search Algorithms

The search techniques to find solutions to a CSP are based normally on Backtracking algorithms. The try to find a solution through the space of possible assignments of values to the variables, if it exists, or to prove that the problem has not a solution. Because of this they are known as complete algorithm. The incomplete algorithms do not guarantee to find a solution, but they are very used in optimization problems since their mayor efficiency and the high cost that a complete search requires. A lot of complete search algorithms have been developed.

4.1.3 Constraint Optimization Problem

There are often a lot of solutions to a CSP, but a user is interested only in some of them, on only in a specific one. To solve this limitation some extensions of the model have been proposed, where it is allowed to have weak constraints (which indicate preferences, not obligation) with different semantics, such as priorities, preferences, costs, or probabilities. In the Constraint Optimization Problems (COP) the aim is to find the best solution, where the preference criteria between the solutions is specified by the weak constraints and an objective function that has to be optimized.

4.1.4 Integer Programming

In many optimization problems, the domains of the variables are limited to integer values. This way, a problem in integer linear programming is an optimization problem where some of the variables are limited to integer values, so that the objective function and the constraints are linear functions of the variables.
This problems are classified in accordance with the type or number of integer variables. When de domain of all variables of the problem are limited to integer values, it is called pure-integer programming. If, in addition, those integer values are limited to be 0 or 1, the problem is binary-integer. When integer and float variables coexist in a problem, it is called mixed-integer.

4.2 Overconstrained Constraint Satisfaction Problems

When solving a Constraint Satisfaction Problem, it is necessary to assign values to variables satisfying a set of constraints. In real applications it often happens that problems are overconstrained and do not have any solution. In this situation, it is desirable to find the assignment that best respects the constraints under some preference criterion. Under this view, overconstrained CSPs are optimization problems for which branch and bound is a suitable solving strategy. The efficiency of branch and bound-based algorithms greatly depends on the lower bound used to detect dead ends and to avoid the exploration of large regions in the search space. This lower bound should be both as large and as cheap to compute as possible.
Approaches (Affane and Bennaceur, 1998; Freuder and Wallace, 1992; Wallace, 1995) for lower bound computation aggregate two main elements: (i) the global contribution of assigned variables, and (ii) the addition of individual contributions of unassigned variables. Another approach (G. Verfaillie and Schiex, 1996) keeps (i) but substitutes (ii) by a global contribution of unassigned variables. This

is done by the Russian Doll Search (RDS) method, which requires n successive searches on nested sub-problems to finally solve a problem of n variables.

A discrete binary constraint satisfaction problem is defined by a finite set of variables $X = \{1, \ldots, n\}$, a set of finite domains $\{D_i\}_{i=1}^{n}$ and a set of binary constraints $\{R_{ij}\}$. Each variable i takes values in its corresponding domain D_i. A constraint R_{ij} is a subset of $D_i \times D_j$ which only contains the allowed value pairs for variables i, j. An assignment of values to variables is complete if it includes every variable in X, otherwise it is partial solution for a CSP is a complete assignment satisfying every constraint. The problem is called overconstrained if such an assignment does not exist. It may be of interest to find a complete assignment that best respects all constraints (Bistarelli et al., 1995; Schiex et al., 1995). The Maximum Constraint Satisfaction Problem (Max-CSP) can be consider (Kask, 2000; Larrosa and Meseguer, 1999), for which the solution of an overconstrained CSP is a complete assignment satisfying the maximal number of constraints. That is, in a Max-CSP a number of constraints are allowed to be violated, and the quality of a solution is measured by the number of satisfied constraints. This way, in order to identify the constraints which need to be relaxed (or removed) to get a solution in an overconstrained CSP, the concept of Max-CSP can be combined with reified constraints. A reified constraint consists of a constraint associated to a boolean variable which denotes its truth value. Therefore, by maximizing the number of reified constraints whose truth values are equal to *true*, the constraints to relax will be the rest.

Overconstrained numeric CSPs

Many practical problems require solving constraint satisfaction problems (CSPs) with numerical constraints. A numerical CSP (NCSP), (V, C, D), is stated as a set of variables V taking their values in domains D over the reals and subject to a finitely many set of constraints C. In practice, the constraints can be equalities or inequalities of arbitrary type and arity, usually expressed using arithmetic expressions. The case of NCSPs with non-isolated solutions is often encountered in real-world engineering applications where under-constrained problems, problems with inequalities or with universal quantifiers are ubiquitous. In practice, a set of non-isolated solutions often expresses a spectrum of equally relevant choices, as the possible moving areas of a mobile robot, the collision regions between objects in mechanical assembly, or different alternatives of shapes for the components

of a kinematic chain. These alternatives need to be identified as precisely and completely as possible.
A lot of Artificial Intelligence problems can be cast in terms of Numeric Constraint Satisfaction Problems (NCSPs), and a large number of systems have been developed to compute efficiently solutions of these problems. NCSPs are more and more often used to solve engineering problems arisen in different areas such as qualitative reasoning, diagnosis, planning, scheduling, configuration, distributed artificial intelligence, etc... This work focuses on problems related to engineering field, what play a prominent role in industrial applications. Generally, these problems are formed by a set of constraints among variables whose domains are real interval values. Usually, the numeric constraints are linear or polynomial relations (equations or inequations).
However, not every set of numeric constraints is satisfiable. Different researchers have proposed methods for the identification of Minimally Unsatisfiable Subsets of Constraints (MUSes) or Conflict Sets (CS) as they are also named in over-constrained CSPs. Determining MUSes can be very valuable in many industrial applications, because it describes what is wrong in a NCSP instance. They represent the smallest explanations -in terms of the number of involved constraints- of infeasibility. Indeed, when we check the consistency of a NCSP, we prefer knowing which constraints are contradicting one another rather than only knowing that the whole NCSP is inconsistent.

Definition 4.2. *Numeric Variable: variable of the NCSP whose domain is a real interval value. The set of numeric variables of the problem is denoted by X^{Ψ} and $X^{\Psi}(c_i)$ stands for the set of variables of a constraint c_i.*

Definition 4.3. *Numeric Constraint: linear or polynomial relation (equations or inequations) involving a finite subset of numeric variables.*

Definition 4.4. *Goal: predicate that denotes the users' preferences to search why the NCSP is overconstrained.*

Definition 4.5. *Numeric CSP: four-tuple psi $= (X, D, C, G)$ where $X^{\Psi} = \{x_1, \dots, x_n\}$ is a set of variables, whose continuous domains are respectively $D^{\Psi} = \{d_1, \dots, d_n\}$ $(n \geq 1)$, $C^{\Psi} = \{c_1, \dots, c_m\}$ $(m \geq 1)$ is a set of numeric constraints and G^{Ψ} is the goal.*

Definition 4.6. *Overconstrained NCSP: NCSP with no solution caused by some of the domains or constraints contradicting others.*

In the bibliography, different types of CSPs have been treated in order to obtain the MUSes (Liffiton and Sakallah, 2008; Shah, 2011). They are related to Satisfiability Problems (Junker, 2001; Bruni, 2003; Oh et al., 2004; Éric Grégoire et al., 2007), Disjunctive Temporal Problem (DTP) (Moffitt and Pollack, 2005; Liffiton and Sakallah, 2005; Mark Liffiton Michael and Pollack, 2007) and model-based diagnosis and debugging problems (Mauss and Tatar, 2002; de la Banda et al., 2003; Gómez-López et al., 2004; Ceballos et al., 2006). Due to the high computational complexity of these problems, the goal of most of these approaches was to reduce the amount of satisfaction checking and subsets examined. However, some approaches were designed to derive only some MUSes and no all MUSes of these overconstrained CSPs.

To derive MUSes in overconstrained NCSP, we are aware of very few technical works. Irreducible Infeasible Subsets (IIS) was studied for only linear and integer domains, but not all MUSes are obtained. These problems may contain multiple MUSes, and all of them must be resolved by constraint relaxation before the NCSP can be solved. Also, other authors of the model-based diagnosis community have treated the high complexity of these problems using constraint databases (Gómez-López et al., 2004) and new concepts such as constraint clusters and nodes (Ceballos et al., 2006).

In Gasca et al. (2007), a set of new derivation techniques are presented to obtain efficiently MUSes of a overconstrained NCSP. These techniques improve the complete technique in several ways depending on the structure of the constraint network. It makes use of the powerful concept of the structural lattice of the constraints and neighbourhood-based structural analysis to boost the efficiency of the exhaustive algorithms. As systematic methods for solving hard combinatorial problems are too expensive, structural analysis offers an alternative approach for quickly generating all MUSes. Accordingly, experimental studies of these new techniques outperform the best exhaustive ones. They avoid to solve a high number of NCSPs with exponential complexity, however they add some new procedures with polynomial complexity.

4.3 Business Data Constraints and Constraint Programming

As it was aforementioned in Chapter 2, the relations between the data managed in a business process, and even the values held by those data, are expressed by means of business data constraints. In the current Thesis Dissertation, the business data constraints in a business process model are modelled by using Constraint Programming, expressed as numerical constraints whose values and variables are defined in Natural, Float or Integer domain, according to the next grammar in BNF:

Definition 4.7. *Let $v \in V$ be variable, let int_val be an integer value, let nat_val be a natural value, and let $float\_val$ be a float value. The set of constraints on V, denoted $C(V)$, is generated by the following grammar in BNF:*

$$\begin{array}{rcl}
Constraint & ::= & Atomic\_Constraint\ BOOL\_OP\ Constraint \\
 & \mid & Atomic\_Constraint \mid \neg Constraint \\
BOOL\_OP & ::= & \wedge \mid \vee \\
Atomic\_Constraint & ::= & Function\ \ PREDICATE\ \ Function \\
Function & ::= & v\ \ FUNCTION\ \ Function \\
 & \mid & v \mid int\_val \mid nat\_val \mid float\_val \\
PREDICATE & ::= & < \mid \leq \mid = \mid > \mid \geq \\
FUNCTION & ::= & + \mid - \mid * \mid \div
\end{array}$$

During the diagnosis process, all the business data constraints have to be stored and obtained in an efficient way to determine which business data constraints are involved in the execution of the abnormal instance. As the business data constraints are represented by constraints, Constraint Databases (CDBs) can be used to support and handle these rules (Gómez-López and Gasca, 2010).
When dealing with a great quantity of data, the use of a database is a mandatory decision. The storage of business data constraints also implies storing all the details related to their variables, the domain of variables and data persistence relationships. These types of information and business data constraints expressed by constraints are supported by Constraint Database Management Systems (CDBMS).
CDBs were initially developed by Kanellakis et al. (1992). The basic idea behind the CDB model is to generalize the notion of a tuple in a relational database to a conjunction of constraints, since a tuple in relational algebra can be represented as an equality constraint between an attribute of the database and a constant. In a real business process, great quantity of compliance rules must be defined, hence a repository is required in order to evaluate these rules as soon as possible.

The CDB used in this Thesis Dissertation is based on Labelled Object-Relational Constraint Database Architecture (LORCDB Architecture) (Gómez-López et al., 2009).

Part III

Contribution I: Verification of Semantic Business Process Models

Chapter 5

Diagnosing Correctness of Semantic Business Process Models

5.1 Introduction

In order to model operational business processes in an accurate way, business process models need to reference both the control flow and data flow perspectives. Checking the correctness of such models and giving precise feedback in case of faults is challenging due to the interplay between these difference perspectives.
This chapter presents a fully automated contribution for diagnosing correctness of semantic business process models in which the semantics of activities is specified with pre and postconditions that describe the expected behaviour of the activities and corresponding to the phase *Verification of Correctness of Semantic Model* in the diagram in Figure 1.2. To this end, the control flow and data flow perspectives of a semantic business processes are modelled in an integrated way using Artificial Intelligence techniques (Integer Programming and Constraint Programming).
The approach has been implemented in the DiagFlow tool, which reads and diagnoses annotated XPDL models, using a constraint solver as back end. By using this novel approach, complex semantic business process models can be verified and diagnosed in an efficient way.

5.1.1 Motivation

For organizations it is essential to ensure the correct operation of business process models at design time, before the business processes get enacted. An incorrect operational business process cannot satisfy customers and fixing the faults can be very costly, certainly compared to the cost of fixing the business process model before it is deployed. Correctness of a business process model can be verified by exhaustively checking all possible execution instances. Detected faults should be diagnosed, for instance by providing a fault path that shows the cause of the fault, such that faults can be repaired in a quick and effective way.

Business process models can reference different perspectives. As it was analysed in the state-of-the-art in Chapter 2, most business process modelling and verification approaches only consider the control flow perspective, which is about the order in which the individual activities of a business process are executed. Another relevant perspective to be considered is the data flow perspective, which details the flow of data among activities subject to certain constraints. The data flow perspective is important since data constraints influence the possible executions of activities and in turn, the execution of activities results in certain data constraints being enforced.

The verification of business process models allows to determine the correctness of business processes in terms of: a business process cannot be correct (1) if one or more of its activities can never get executed for any possible valuation of the data input; or (2) if there is not any possible process instance for a certain valuation of the input data. In these cases, despite the business process model can be correct regarding the control flow perspective, it is not correct from the perspective of the data flow.

In order to diagnose faults in data semantics, an effective means to express data constraints is to annotate activities in a business process model with pre and postconditions that specify the effect on the data state for each activity. For instance, in the Sarbanes-Oxley Act of 2002, the internal audit department takes the lead and works alongside business process owners for each process that has a direct effect on the data for the financial reporting. Annotating activities inside these processes with pre and postconditions facilitates compliance checking to ensure that business processes are properly designed. It is an effective means to capture such dependencies between control flow and data flow. Moreover, pre and postconditions can capture requirements on the data flow that any execution of the business process has to comply with.

5.1.2 Contribution

The goal of the contribution in this chapter is to develop an approach for diagnosing the correctness of the relation between values in semantic business process

models, containing activities whose effects are formally specified using pre and postconditions. An activity can start if the execution of the business process model has reached the activity and its precondition is satisfied. Upon completion, the activity delivers data that satisfies its postcondition. An execution of the business process can reach an activity whose precondition is not satisfied. In that case the execution gets stuck at the activity and fails.
In order to detect these errors, we distinguish between two different notions of correctness to diagnose such data-flow errors:

- May-correctness. A business process model is may-correct if every activity can be reached and executed at least once, so there is an execution in which the activity is done. The fulfilment of this correctness ensures that there is at least one valuation of the data inputs which makes an activity executable.

- Must-correctness. A business process model is must-correct if every possible execution that reaches an activity satisfies the precondition of the activity. The fulfilment of this correctness ensures that there is no valuation of the data inputs which makes an activity non-executable.

The diagnosis is performed at design-time, using artificial intelligence techniques to compute the execution instances allowed by a business process model. For diagnosis, the business process model is translated into two models: (1) an Integer Programming model (IP model), to determine the different instances of execution of the business process, and (2) the pre and postconditions of the activities are modelled as constraints in a Constraint Satisfaction Problem (CSP) (Rossi et al., 2006), following a BNF grammar in order to avoid any ambiguity.
This approach makes several contributions:

- The *Workflow data graphs* definition is proposed as a formalism for modelling the semantics of the values for the data in the business processes with pre and postconditions for the activities. These conditions are modelled as constraints according to a well-defined grammar in BNF.

- Two correctness notions for workflow data graphs, *may* and *must-correctness*, are proposed and innovative diagnosis algorithms are developed for verifying may and must-correctness. The algorithms are complete: neither false positives nor false negatives are generated. Moreover, the algorithms offer precise diagnosis of the detected errors, indicating the execution causing the error where the business process gets stuck.

- The approach has been implemented in the DiagFlow tool, as an extension of the one presented in Eshuis and Kumar (2010). The tool reads XPDL models (WFMC, 2005) in which the semantics of activities and the corresponding business process are specified using extended attributes.

The remainder of the chapter is organized as follows. Next subsection 5.1.3 presents a motivating example to illustrate the concepts of may and must-correctness all along the contribution. Section 5.2 introduces workflow data graphs as a formal model for semantic business processes and defines may and must-correctness on workflow data graphs. Section 5.3 defines the IP and CSP formulation of a workflow data graph. The process of diagnosis is explained, and two algorithms for the corresponding correctness are presented. The diagnosis of the illustrative example is also performed. Section 5.4.1 gives implementation details and presents the developed tool. Section 5.4 shows experimental results. And finally, a summary of the contribution is provided in Section 5.5.

5.1.3 Illustrative Example

This section introduces an example of a semantic business process model, shown in Figure 5.1. This example describes the handling of a conference for an organizing committee, and it is used to illustrate the concepts of may and must-correctness in semantic business process models. The graphical representation BPMN 2.0 (OMG, 2011) is used to visualize business process models.

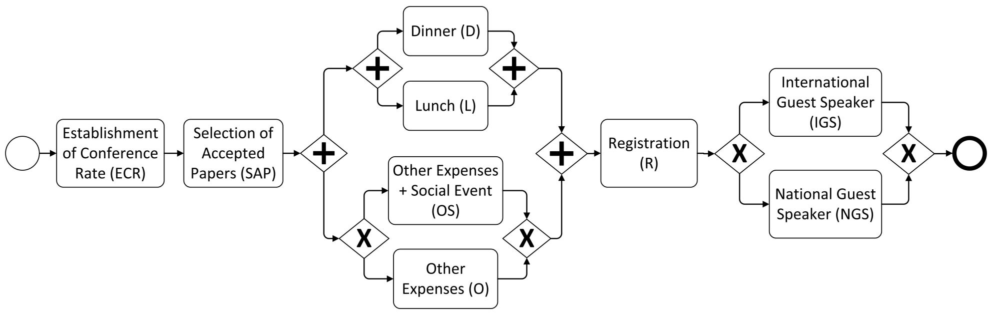

Establishment of Conference Rate (ECR)
Selection of Accepted Papers (SAP)
Dinner (D)
Lunch (L)
Other Expenses + Social Event (OS)
Other Expenses (O)
Registration (R)
International Guest Speaker (IGS)
National Guest Speaker (NGS)
+
X

Figure 5.1: Example of business process

Figure 5.1 shows a business process that consists of nine activities (rectangles with rounded corners) and eight gateways or control nodes (diamonds), and a start and end event (circles). A gateway with one incoming edge and multiple outgoing edges is called a split; a gateway with multiple incoming edges and one outgoing edge is a join. Gateways with the +-symbol are parallel gateways: all incoming edges are required to pass the gateway, and the gateway activates all outgoing edges. The other gateways are XOR: one incoming edge can pass the gateway and one of the outgoing edges is activated as a result. In the figure, the activity labels include the activity names and their abbreviations in brackets. The business process performs the following steps:

1. The process starts with the *Establishment of Conference Rate* activity, in order to begin the registration period.

2. In the activity *Selection of Accepted Papers*, the process of acceptance of papers for the conference takes place. The number of final papers is determined.

3. The flow is split into two branches. In the upper one, the cost of the gala dinner (*Dinner*) and the lunches (*Lunch*) to serve during the conference are calculated concurrently. On the lower branch, the flow is routed according to the money spent in the social events during the conference (*Other Expenses* or *Other Expenses + Social Event* activities).

4. Next, the *Registration* of the attendees of the conference takes place.

5. And finally, the process is routed depending on the available money to spend in the invitation of national or international guest speakers (*National Guest Speaker* or *International Guest Speaker* activities).

The activities in the example consume and produce data during the execution of the business process by reading and writing variables. Those variables are listed in Table 5.1 with their corresponding domains and meanings. Table 5.2 shows how these variables are used by the activities of the example, indicating if they are read (*rd*) or written (*wt*).
Each activity in a business process uses two types of condition over the data flow which must be satisfied. A precondition must be satisfied prior to execution of the activity. If not, the business process gets stuck at the activity and fails. A postcondition is satisfied immediately after the activity has finished. The pre and postconditions of the activities in the example in Figure 5.1 are shown in Table 5.3. The notation is explained in the next section.

Table 5.1: Data input for the example in Figure 5.1

Variable	Domain	Meaning
regFee	{200..390}	Conference registration fee
sponsorship	{0..15000}	External contributions to support the event
numPapers	{50..80}	Number of accepted papers
dinner	{60..100}	Gala dinner cost
lunch	{10..30}	Cost of each lunch served during the conference
others	{30..185}	Money for other expenses, like social events
confAtt	{75..170}	Number of conference attendees
guestSpeaker	{0..10000}	Money to spend in inviting a guest speaker

Table 5.2: Data read and written on each activity

	Activities								
Variables	ECR	SAP	D	L	OS	O	R	IGS	NGS
regFee	wt	-	rd	rd	rd	rd	rd	rd	rd
sponsorship	wt	-	rd	rd	rd	rd	-	rd	rd
numPapers	-	wt	rd	rd	rd	rd	rd	-	-
dinner	-	-	wt	-	-	-	rd	rd	rd
lunch	-	-	-	wt	-	-	rd	rd	rd
others	-	-	-	-	wt	wt	rd	rd	rd
confAtt	-	-	-	-	-	-	wt	rd	rd
guestSpeaker	-	-	-	-	-	-	-	wt	wt

It is easy to check that the business process is correct from the control flow perspective: each activity can be performed and there are no deadlocks, so the process can always complete. To assess the correctness of the example from the data flow perspective, there are two important types of questions. Both questions test whether the precondition of an activity *a* can be satisfied by considering an arbitrary partial execution of the business process in which *a* is to be executed next. The partial execution of the business process results in a data state (assignment of values to variables) that has to satisfy the precondition of *a*.

Table 5.3: Activities with their pre and postconditions

Activity	Precondition and Postcondition
ECR	**pre:** $true$ **post:** $true$
SAP	**pre:** $true$ **post:** $true$
D	**pre:** $sponsorship > 0 \vee numPapers > 60$ **post:** $regFee * 0.1 \leq dinner \wedge dinner \leq regFee * 0.35$
L	**pre:** $sponsorship > 0 \vee numPapers > 60$ **post:** $regFee * 0.1 \leq 3 * lunch \wedge 3 * lunch \leq regFee * 0.35$
OS	**pre:** $sponsorship > 0 \vee numPapers > 60$ **post:** $others \leq 0.2 * regFee + 0.05 * sponsorship \wedge$ $others \geq 0.05 * regFee + 0.05 * sponsorship$
O	**pre:** $sponsorship > 0 \vee numPapers > 60$ **post:** $others \leq 0.25 * regFee \wedge others \geq 0.05 * regFee$
R	**pre:** $3 * lunch + dinner + others < regFee$ **post:** $numPapers * 1.8 \geq confAtt$ $\wedge numPapers * 0.5 \leq confAtt$
NGS	**pre:** $confAtt * (3 * lunch + dinner + others) <$ $confAtt * regFee + sponsorship$ **post:** $guestSpeaker \geq 0.2 * sponsorship \wedge$ $guestSpeaker \leq sponsorship + 0.1 * regFee * confAtt$
IGS	**pre:** $confAtt * (3 * lunch + dinner + others) <$ $confAtt * regFee + sponsorship$ **post:** $guestSpeaker \geq 0.4 * sponsorship \wedge$ $guestSpeaker \leq sponsorship$

One question is whether for each activity a there exists at least one partial execution of the business process in which a can be done next and the resulting data state satisfies the precondition of a. In that case, activity a can become enabled and executed. Otherwise, the execution of the business process may get stuck at a, if the current data state does not satisfy the precondition of a. If every activity can be executed, the business process is *may-correct*. For the business process in

Figure 5.1, an example would be: is it possible to have at least a valuation of the variables that allows to invite an *International Guest Speaker*?

The other relevant question is whether every possible partial execution of the business process results in a data state that satisfies the precondition of the activity to be executed next. Phrased differently, can every possible partial execution always be continued such that eventually the end state is reached? If the answer is positive, the business process is *must-correct*. For the example in Figure 5.1, is it always possible to count on the money to invite an *International Guest Speaker*?

Note that must-correctness is stronger than may-correctness. It is straightforward to check that if a business process is must-correct, it is also may-correct. May-correctness can be used as sanity check for each activity to see whether its precondition is not too strict. Must-correctness can be used to check the correctness of the entire business process with all the activities and to avoid possible incorrect input values that makes the process fail.

In the case of the example in Figure 5.1, the business process is may-correct since every activity is executable. On the other hand, it is not must-correct: for example, if the partial execution contains activities *Establishment of Conference Rate*, *Selection of Accepted Papers*, *Dinner*, *Lunch*, and *Other Expenses + Social Event* then the resulting data state can assign the following values to the variables: *regFee*=200, *dinner*=100, *lunch*=30, *others*=30. But now the activity to be executed next, *Registration*, has a precondition that is false, since $3 * 30 + 100 + 30 \not< 200$. Therefore, with that assignment of the variables, the business process gets stuck at activity *Registration*.

Note that all activities have syntactically correct pre and postconditions, and that the business process model has a correct control flow definition: no deadlock occurs if the data flow (pre and postconditions) is abstracted form. The error is caused by the interplay between the control flow, which specify that *Establishment of Conference Rate*, *Selection of Accepted Papers*, *Dinner*, *Lunch*, and *Other Expenses + Social Event* are performed before *Registration*, and the data flow as specified by the pre and postcondition of each activity, which determines the possible date states just before *Registration*.

The error can be repaired in several ways, for instance by relaxing the precondition of *Registration*, by strengthening the postconditions of *Dinner*, *Lunch* and *Other Expenses + Social Event*, by rearranging the control flow, or even by reducing the domains of the variables. So finding an error at a precondition does not necessarily imply the precondition itself is flawed.

5.2 Workflow Data Graphs

In order to analyse may and must-correctness of semantic business processes, this contribution is based on graph-theory and Artificial Intelligence techniques. To this end, the information regarding both control flow and data flow perspectives of the model is collected in a data structure called workflow data graph.
In this section, workflow data graphs are formally defined, including the structural constraints that they should satisfy. Next, the notions of *data instance subgraph* and correctness for workflow data graphs are introduced. Finally, the concepts of *partial instance subgraph* and *border activity* are presented. The definitions extend earlier proposed definitions for the control flow perspective of business process models (Eshuis and Kumar, 2010; Sadiq and Orlowska, 2000) by adding data.

5.2.1 Definitions

A workflow data graph is a set of activities that is ordered to a set of procedural rules. The effect of each activity can be specified with pre and postcondition. The order of execution of the activities is specified by means of directed edges, with a unique start and a unique end node.

Definition 5.1. *A workflow data graph is a tuple $P = (Act, V, D, C, E, pre, post)$ where:*

- *Act is a set of activities;*
- *V is a set of typed variables;*
- *D is a set of finite domains (types), that contains for each variable $v \in V$ a finite domain D_v;*
- *C is a set of control nodes (gateways), partitioned into disjoints sets of XOR splits S_{XOR}, AND splits S_{AND}, XOR joins J_{XOR}, AND joins J_{AND}, and $\{start, end\}$ where $start$ is the unique start node and end the unique end node. Each split in S_{XOR} counts on condition expressions for each gate of the gateway in order to specify the flow depending on the data;*
- *$E \subseteq Act \times Act$ is a set of edges which determine precedence relation;*
- *$pre : Act \rightarrow Cs(V)$ assigns to each activity its precondition (a constraint Cs over variables in V);*

- *$post : Act \to Cs(V)$ assigns to each activity its postcondition (a constraint Cs over variables in V);*
- *$wt : Act \to V$ assigns to each activity its written variables;*

Let $a \in Act$ be an activity. Variables in the precondition of a, so in $vars(pre(a))$, are read by a. Variables in the postcondition of a, so in $vars(post(a))$, are read and/or written by a.
For an activity a, the pre and postcondition of a are expressed as numerical constraints, according to the grammar in Definition 4.7:
Next, each workflow data graph should satisfy the following structural constraints on its control flow:

1. the start node has no incoming edge and one outgoing edge;
2. the end node has one incoming edge and no outgoing edge;
3. each activity has one incoming and one outgoing edge;
4. each split node has one incoming and at least two outgoing edges;
5. each join node has at least two incoming edges and one outgoing edge;
6. each node is on a path from the start to the end node (connectedness);
7. the precedence relation is acyclic.

Formalizations of these constraints are presented elsewhere (Eshuis and Kumar, 2008). The first five constraints are self-explanatory. Constraints 1 and 2 were also included by Sadiq and Orlowska (2000). A business process having more than one start point can be modelled by using immediately after the start node a split node that connects to the different start points. Similarly, a business process with more than one end point can be modelled by using a join before the end node. The sixth constraint rules out unconnected business processes because such workflow graphs contain unreachable parts and therefore are flawed by default.
The last constraint is also placed by other works on workflow verification (Sadiq and Orlowska, 2000; van der Aalst et al., 2002; Touré et al., 2008). However, workflow graphs are still sufficiently expressive to model loops that involve blocked iteration (Sadiq and Orlowska, 2000; Weber et al., 2010). Basically, any block in a correct workflow graph can be repeated multiple times without affecting control flow correctness, since a block has a single point of entry and a single point of exit.

Since we do not consider guard conditions (i.e. logical conditions associated to the gateways), to verify business process models with loops a strong fairness constraint is required to ensure that loops are exited eventually (Eshuis and Wieringa, 2004).
In the sequel, we also use auxiliary functions $inedge, outedge : N \rightarrow \mathcal{P}(E)$, which both map each node to a set of edges. For a node n, $inedge(n)$ is the set of edges entering n, while $outedge(n)$ is the set of edges leaving n. Formally, $inedge(n) = \{(x,y) \in E \mid y = n\}$ and $outedge(n) = \{(x,y) \in E \mid x = n\}$. We use subscripts to identify the different elements of $inedge(n)$ and $outedge(n)$. For example, if $inedge(n) = \{e_1, e_2\}$, then $inedge_1(n) = e_1$ and $inedge_2(n) = e_2$.

5.2.2 Analysis of the Diagnosis Problem

In order to check the may and must-correctness of a business process, it is necessary to check the executability of each activity. That executability depends on the precondition of the activity being checked, and on the pre and postconditions of the activities executed before it in a particular instance of the business process. To define it formally, Sadiq and Orlowska (2000) introduce the notion of an *instance subgraph*, which corresponds to a particular execution instance of a workflow graph.
An instance subgraph represents a subset of activities that may be executed for a particular instance of a business process. The part of the workflow graph that covers the visited nodes is an instance subgraph, because it represents a specific execution instance based on the workflow graph. A formal definition of instance subgraphs is presented elsewhere (Eshuis and Kumar, 2008).
For this contribution, we introduce two types of instance subgraphs: partial and complete. A *partial instance subgraph* of a workflow graph is generated by traversing a workflow graph from the start node, using the following rules:

- if an XOR split node is visited, one of its outgoing edge is visited based on a guard condition;
- if an AND split node is visited, then all outgoing edges are visited;
- if an XOR join node is visited, then its outgoing edge is visited only if one of its incoming edges has been visited too, and all other incoming edges have not been visited;

- if an AND join node is visited, then its outgoing edge is visited only if all its incoming edges have been visited too;
- if an activity is visited, then its outgoing edge is visited too.

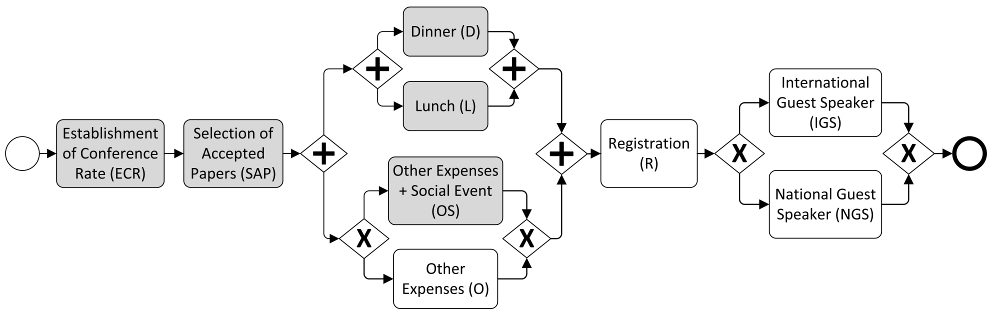
A *complete instance subgraph* is generated by traversing a workflow graph from the start node using the rules for partial instance subgraphs plus the additional rule:

- if the incoming edge of an activity is visited, then the activity is visited too.

Control flow verification (Eshuis and Kumar, 2010; Sadiq and Orlowska, 2000) only considers complete instance subgraphs. However, to verify data flows, we also need to consider partial instance subgraphs, which contain the incoming edge of the last activity in the instance, but not the activity itself. To check whether the precondition of the activity is satisfied by the partial instance subgraph, we need to identify the possible assignments of variables written by the activities in the instance subgraph.
A *data instance subgraph* is an instance subgraph together with an assignment of values to the variables in V. The assignment must be feasible according to the postcondition of the activities visited last. To formalize this properly, we introduce the following notion: an activity a in an instance subgraph (partial or complete) is a *border activity* if there is no activity a' in the instance subgraph such that there is a directed path from a to a'. For the business process in Figure 5.1, the subset of activities shadowed in Figure 5.2 ($\{Establishment\ of\ Conference\ Rate$, $Selection\ of\ Accepted\ Papers, Dinner, Lunch, Other\ Expenses + Social\ Event\}$) induces a partial data instance subgraph, whose border activities are $Dinner$, $Lunch$, and $Other\ Expenses + Social\ Event$. Note that due to the traversal rules the data instance subgraph also contains the successor (control) nodes of the border activities. In a data instance subgraph, the assignment of values to variables must be consistent with the post condition of each border activity.

Definition 5.2. *A data instance subgraph of a workflow graph* $(Act, V, D, C, E, pre, post)$ *is a tuple* $(Act', V', D', C', E', pre', post', \nu)$ *where:*

- $(Act', V', D', C', E', pre', post')$ *is an instance subgraph with* $Act' \subseteq Act$, $V' = V$, $D' = D$, $C' \subseteq C$, $E' \subseteq E$, $pre' = pre \cap (Act' \to Cs(V))$, *and* $post' = post \cap (Act' \to Cs(V))$, *and*



Establishment of Conference Rate (ECR)
Selection of Accepted Papers (SAP)
Dinner (D)
Lunch (L)
Other Expenses + Social Event (OS)
Other Expenses (O)
Registration (R)
International Guest Speaker (IGS)
National Guest Speaker (NGS)

Figure 5.2: Example of instance subgraph (shadowed activities)

- ν *is a valuation assigning a value* $\nu(v)$ *to each variable* $v \in V$*. That value is in its domain* D_v *such that for each border activity a, the valuation of variables in* $var(post(a))$ *satisfies the postcondition* $post(a)$.

There are two types of possible error for data instance subgraphs. First, a data instance subgraph can get stuck at an XOR or AND join. Then, it contains the join but not the outgoing edge of the join (Eshuis and Kumar, 2010). This is a control flow error that can be detected using existing techniques (Eshuis and Kumar, 2010; Sadiq and Orlowska, 2000). We therefore ignore such errors for the remainder of this chapter.
The second error is a data-flow error. Different types of data-flow errors can occur, depending on the level of detail on which the data flow is specified. At the minimal level, a business process model specifies for each activity which variables it reads and writes, but the business process model contains no pre and postconditions. Sun et al. (2006) have analysed which data-flow errors can occur in such business process models. For the purpose of this contribution, the following two errors are important:

- Data is *missing* if a variable is read by an activity, so referenced in its precondition, but not written in any preceding activity.
- Data is *conflicting* if the same variable is written in two parallel activities. In that case, one activity overwrites the value of the variable written earlier by the other activity.

These data-flow errors at the basic level can cause unexpected process interruptions and should therefore be avoided. In Section 5.3.2 we define an algorithm for detecting data conflicts.

At a more advanced level, a business process model not only contains variables read and written by activities, but also contains pre and postconditions for activities. For such business process models, data-flow errors can occur that are not considered by Sun et al. (2006). A precondition of an activity is *violated* if the precondition is not satisfied when the activity can be executed from a control flow point of view. In that case, the business process gets stuck and fails.
To formalize precondition violation errors, we introduce the notion of trigger. A partial instance subgraph *triggers* activity a if it does not contain a but does contain the incoming edge of a, which is unique. So the instance subgraph "stops" just before a. In the example in Figure 5.2, the shadowed partial instance subgraph triggers the activity $Registration$.
Based on the notion of trigger, we define two new notions of data-flow correctness. First, we introduce two auxiliary notions.

Definition 5.3. *An activity* **a** *is* may-executable *if there exists a data instance subgraph that triggers* **a** *and whose valuation of variables satisfies the precondition of* **a**.

Definition 5.4. *An activity* **a** *is* must-executable *if every data instance subgraph that triggers* **a** *has a valuation that satisfies the precondition of* **a**.

Next, we define the two new notions of data-flow correctness.

Definition 5.5. *A workflow data graph is* may-correct *if every activity is may-executable.*

Definition 5.6. *A workflow data graph is* must-correct *if every activity is must-executable.*

In a must-correct workflow data graph, no preconditions can be violated. But a may-correct workflow data graph might contain an activity whose precondition can be violated. Still may-correctness is useful, as explained in Section 5.1.3: may-correctness can be used for testing whether all activities can be executed under certain conditions while must-correctness can be used to check absence of precondition violations.
In the next section, we will formalize data instance subgraphs, including the valuations they allow, as Constraint Satisfaction Problems. We will define algorithms that use the IP and CSP formalizations to analyse the may and must-correctness of workflow data graphs.

5.3 Diagnosis of Workflow Data Graphs: May and Must-correctness

In this section, we explain how correctness of workflow data graphs can be diagnosed in a formal way. To this end, diagram in Figure 5.3 shows the method proposed in this contribution, corresponding to the task *Verification of Correctness of Semantic Models* of the diagram in Figure 1.2.

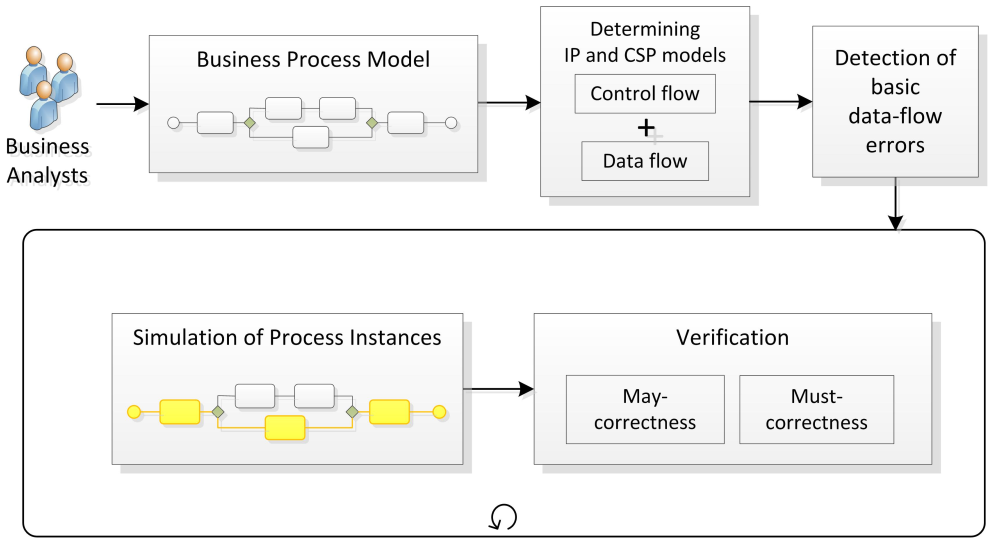


Figure 5.3: Diagram of the process

The process starts from the business process model designed by analysts, and performs the formalization of both the control flow and data flow perspectives by using respectively Integer Programming (IP) formulation introduced in Eshuis and Kumar (2010), and Constraint Satisfaction Problems (CSPs) to formalize pre and postconditions.
Then, the diagnosis of the model begins, starting with the detection of certain basic data-flow errors, and going on with the simulation of process instances to verify the two types of correctness over each one of them.
In the remainder of this section, we first explain the IP and CSP models, describing also the preprocessing which is necessary to avoid conflicts among postconditions, and detect errors in the data flow. Then, the algorithms for verifying the may and must-correctness are detailed, ending the section with their application over the example in Figure 5.1.

5.3.1 Combined IP and CSP Model

We first explain the IP formulation that covers the control flow perspective of workflow data graphs. Next we extend the IP model with constraints that model the data flow perspective of workflow data graphs, getting a CSP model.
The selection of the Integer Programming formulation to model the control flow perspective lies in it can be solved in a more efficient way since variables to simulate instances in the control flow only need 0-1 values in their domains.

IP formulation For every activity $a \in Act$, we need to compute an instance subgraph that triggers a. For this, we use the IP formulation developed in Eshuis and Kumar (2010), in which an IP variable is introduced for each node and each edge of the workflow graph. That IP formulation models AND and XOR gateways (splits and joins) and obviates the modelling of OR gateways for being a particular case of the AND gateways, where no all branches have to be executed at every process instance.
A solution to the IP formulation encodes an instance subgraph, where an IP variable has value 1 if and only if the corresponding node or edge is part of the instance subgraph. Complicating factor is that the existing IP formulation considers complete instance subgraphs, that can only get stuck at (faulty) AND or XOR joins. But an instance subgraph that triggers a is not complete.
To generate a partial instance subgraph that triggers a, we take the existing IP formulation (Eshuis and Kumar, 2010) but replace one constraint. The existing IP formulation uses for each $a \in Act$ the constraint $inedge_1(a) - a = 0$, which states that if the incoming edge of a is activated, so $inedge_1(a) = 1$, then a is activated as well, so $a = 1$. To model that the incoming edge of a is activated but not a, we replace for every $a \in Act$ the constraint $inedge_1(a) - a = 0$ in the original IP formulation with $inedge_1(a) >= a$. This constraint allows that the subgraph "stops" at a ($inedge_1(a) = 1$ and $a = 0$) but disallows that a is spontaneously activated, so $inedge_1(a) = 0$ and $a = 1$ is not allowed.
To generate partial instance subgraphs, as it was mentioned before, we use a slightly modified version of the basic IP formulation. All constraints below, except IP4, are taken from the basic IP formulation. For AND joins we use the relaxed IP formulation (IP4) to allow for partial instance subgraphs that stop at an AND join. In that case, one of the parallel branches synchronised by the AND join has completed, but the other one has to complete. This behaviour is disallowed by the basic IP formulation.

Definition 5.7. *For a workflow data graph $P = (Act, V, D, C, E, pre, post)$, the* ***Relaxed IP formulation*** *maximizes the value at the end node subject to the following constraints at each node in the workflow graph. For each node and edge x of P, so $x \in \{Act \cup C \cup E\}$, an IP variable* `x` *is created. The constraints are:*

IP0 `start = 1`

IP1 For $n \in (S_{AND} \cup S_{XOR} \cup \{end\})$: $\texttt{inedge}_1\texttt{(n) - n = 0}$

IP1a For $n \in Act$: $\texttt{inedge}_1\texttt{(n) >= n}$

IP2 For $n \in (Act \cup J_{AND} \cup J_{XOR} \cup \{start\})$: $\texttt{outedge}_1\texttt{(n) - n = 0}$

IP3 For $n \in S_{AND}$: $\texttt{outedge}_1\texttt{(n) + outedge}_2\texttt{(n) - 2n = 0}$

IP4 For $n \in J_{AND}$: $\texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n)- n} \leq \texttt{1}$

$\texttt{n} \leq \texttt{inedge}_1\texttt{(n)}$

$\texttt{n} \leq \texttt{inedge}_2\texttt{(n)}$

IP5 For $n \in S_{XOR}$: $\texttt{outedge}_1\texttt{(n) + outedge}_2\texttt{(n) - n = 0}$

IP6 For $n \in J_{XOR}$: $\texttt{inedge}_1\texttt{(n) + inedge}_2\texttt{(n) - n = 0}$

CSP formulation Constraint programming is an Artificial Intelligence technique which provides us a way to model the semantic information of a workflow data graph.
The IP model encodes the control flow of a data instance subgraph. We now explain how the data flow, so the pre and postconditions of the activities contained in the data instance subgraph, are translated into constraints in a CSP. For each activity $a \in Act$, a constraint of the form $a = 1 \implies (pre(a) \wedge post(a))$ is defined. That is, if a is part of the partial instance subgraph, then its pre and postcondition should be satisfied. We need the conjunction stipulating that $a = 1$ to ensure that the pre and postcondition are only enforced if a is activated, so a is in the instance subgraph.
Note that the pre and postcondition constraints hold for each activity in the data instance subgraph, not just for border activities (cf. Definition 5.2). This way, the constraints can be easily encoded in a CSP model. However, this encoding complicates finding a solution to the CSP model, since the postcondition of a border activity might conflict with the postcondition of an earlier executed activity, if

both activities modify the same variable. For instance, a workflow data graph can contain two activities A and B with postconditions $i < 10$ and $i \geq 10$ respectively. In the execution of a real process instance, both postconditions could be satisfied if the variable i is modified more than once, getting different values which makes true those conditions when they are reached. However, a data instance subgraph containing A and B can assign only one value to i, since it is considered as only one variable (i.e. one value), so either the postcondition of A or of B is violated. It would be necessary to count on auxiliary variables to store the different values that each activity can be assigned during the instances. To resolve conflicts, we put the CSP model in SSA form, explained next.

SSA form In order to resolve conflicts among postconditions, we will convert the variables and constraints of the CSP model into Static Single Assignment (SSA) form. The SSA form is used in compiler design as an intermediate representation for a program (Alpern et al., 1988; Cytron et al., 1991). If the workflow data graph is in SSA form, each variable is assigned a value by only one activity. To turn a workflow data graph into an SSA form, each variable v is separated into several variables v_i, each of which is assigned a value by only one activity.

A review of the literature reveals a highly cited algorithm to get the variables in a program in SSA form (Cytron et al., 1991). The algorithm computes the control flow properties of programs, like conditions (XOR) or loops. As an example of variables renaming, the business process in Figure 5.4 uses 4 activities which read and/or write the variables w, x, y, and z. Table 5.4 shows the pre and postconditions before and after renaming. Note that two new constraints have been added at activity D. They are known as Φ-functions in Cytron et al. (1991), and they indicate which assignment to the variable y reaches the join point. That is, the value for variable y depends on the activity which was executed (B or C).

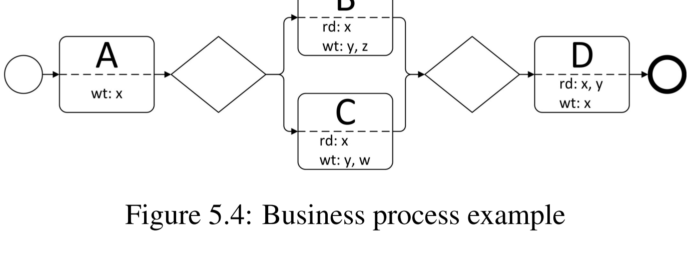


Figure 5.4: Business process example

Table 5.4: Variables before and after the renaming

Activity	Before SSA	After SSA
A	**pre:** $true$	**pre:** $true$
	post: $x > 20$	**post:** $x1 > 20$
B	**pre:** $true$	**pre:** $true$
	post: $y = x + 10$	**post:** $y1 = x1 + 10$
	$z = x * 2$	$z1 = x1 * 2$
C	**pre:** $x < 100$	**pre:** $x1 < 100$
	post: $y = x + 50$	**post:** $y2 = x1 + 50$
	$w = y * 2$	$w1 = y2 * 2$
D		$B = 1 \Rightarrow y3 = y1$
		$C = 1 \Rightarrow y3 = y2$
	pre: $y > x$	**pre:** $y3 > x1$
	post: $x = x * y$	**post:** $x2 = x1 * y3$

However, the SSA form and the renaming algorithm is defined for sequential programs while workflow data graphs can contain parallelism. Due to parallelism, data-flow errors can arise. For instance, two parallel activities can assign the same variable a value (conflicting data, cf. Section 5.2). In that case, the assignment to the variable at a subsequent AND join may not be possible due to conflicting constraints in the CSP model. For instance, if in Figure 5.4 the XOR nodes are replaced with AND nodes, both B and C write variable y. The constraints encoding the Φ-function for the subsequent AND join are now unsatisfiable for $y3$.

Such data-flow errors are at a more basic level than violations of must and may correctness, as explained in Section 5.2. Therefore, these data-flow errors need to be detected and resolved before may and must-correctness can be diagnosed. The next section defines an algorithm for detecting basic data-flow errors.

5.3.2 Detecting Basic Data-Flow Errors

As explained in Section 5.2, two basic data-flow errors are missing data and conflicting data. We next discuss how each type of data-flow error can be detected.

To identify missing data, we use the following constraint. For each activity $a \in Act$ that reads a variable v, there has to be an activity $a_1 \in Act$ such that there is a

```
procedure DATA-FLOW-NO-CONFLICT-CHECK($Act, V, D, C, E$)
    $error = false$
    $unmarked = V$
    $IP$ = make IP formulation for $(Act \cup C, E)$
    while $unmarked \neq \varnothing \wedge error = false$ do
        $current$ = a variable from $unmarked$
        for $a_1 \in Act$ such that $v \in wt(a_1)$ do
            for $a_2 \in Act$ such that $v \in wt(a_2)$ and $a_1 \neq a_2$ do
                $IP_1 = IP$ && $(inedge_1(a_1)$=1) && $(a_1$=0) && $(inedge(a_2)$=1) && $(a_2 = 0)$
                $sol$ = solve $IP_1$
                if $sol$ is not null then
                    Print "Race between activities $a_1$ and $a_2$ for variable $current$"
                    $error = true$
                end if
            end for
        end for
        $unmarked = unmarked \setminus \{ current \}$
    end while
    if $error = false$ then
        Print "The workflow data graph is data-flow-correct"
    end if
end procedure
```

Figure 5.5: Algorithm for checking absence of conflicts

directed path from a_1 to a and a_1 writes v. The presence of a directed path from a_1 to a indicates that a_1 precedes a.

To identify conflicting data, we use the algorithm presented in Figure 5.5. Variables in V are processed one by one in a while-loop. The current variable being processed is *current* (line 6). The algorithm iterates over all activities that write *current* in a nested for-loop. For each pair of distinct activities a_1 and a_2, the algorithm tests whether there exists a partial instance subgraph that triggers both a_1 and a_2. The pre and postconditions of the activities are not relevant for this check, so the CSP formulation is not used but only the IP formulation.

The next theorem asserts the correctness of the algorithm.

Theorem 1. *Let $(Act, V, D, C, E, pre, post)$ be a workflow data graph, Algorithm Data-Flow-No-Conflict-Check finds no error if and only if there is no conflicting data.*

Proof 1. *In the proof, we use the following lemma: two activities are triggered by the same instance subgraph if and only if they are in parallel. This lemma follows immediately from the definition of instance subgraph and the definition of trigger.*
$\Longrightarrow$*: Since algorithm Data-Flow-No-Conflict-Check finds no error, for each variable there is no partial instance subgraph triggering two activities that write the same variable. Therefore, there are no two parallel activities writing the same variable. Therefore there is no conflicting data.*
$\Leftarrow$*: Suppose the algorithm finds an instance subgraph that triggers two activities a_1 and a_2 that both write variable $v \in V$. Then by definition there is a data conflict.*

5.3.3 Algorithm for May-correctness

To check may-correctness of a workflow data graph and provide proper feedback in case of an incorrectness, we developed an algorithm that verifies whether each activity *a* is may-executable (Figure 5.6). For each activity *a*, the algorithm tries to find a data instance subgraph that triggers *a* and whose valuation satisfies the precondition of *a*. The data instance subgraph that is searched for is a solution to the combined IP and CSP model defined in Section 5.3.1 plus additional constraints that encode that *a* is triggered and that the precondition of *a* is satisfied.
Looking at the algorithm in more detail, it begins with the IP formulation of the workflow data graph (line 4 in Figure 5.6), which states the control flow constraints for data instance subgraphs. This IP formulation is combined with the CSP formulation of the workflow (line 5), which states the pre and postcondition constraints for the activities in data instance subgraphs. This combined IP and CSP model is used in the sequel of the algorithm for every data instance subgraph. Next, the algorithm performs a loop to check if all the activities are may-executable (line 6). The activity being processed in the loop is stored in variable *current* (line 7). To test whether activity *current* may-executable, the combined IP and CSP model is extended with constraints that are true if the data instance subgraph triggers *current* and satisfies the precondition of *current* (line 8). If no solution exists, there is no such data instance subgraph for *current*, so *current* is not may-executable (line 10). If all activities are may-executable, the workflow data graph is may-correct (line 17).
The next theorem asserts that the algorithm is correct.

```
1: procedure MAY-CORRECTNESS-CHECK($Act, V, D, C, E, pre, post$)
2:     $error = false$
3:     $unmarked = Act$
4:     $IP$ = make IP formulation for $(Act, E)$
5:     $CSP$ = $IP$ + CSP formulation for $(Act, pre, post)$
6:     while $unmarked \neq \varnothing$ do
7:         $current$ = an activity from $unmarked$
8:         $CSP'$ = $CSP$ && $inedge_1(current) = 1$ && $current = 0$ && $pre(current)$
9:         $sol$ = solve $CSP'$
10:        if $sol$ is null then // $CSP'$ is unsatisfiable
11:            Print "Activity $current$ is not may-executable"
12:            $error = true$
13:        end if
14:        $unmarked = unmarked \setminus \{ current \}$
15:    end while
16:    if $error = false$ then
17:        Print "The workflow graph is may-correct"
18:    end if
19: end procedure
```

Figure 5.6: Algorithm for checking may-correctness

Theorem 2. *Let $(Act, V, D, C, E, pre, post)$ be a workflow data graph. Algorithm May-Correctness-Check finds no error if and only if $(Act, V, D, C, E, pre, post)$ is may-correct.*

Proof 2. $\Longrightarrow$: *If algorithm May-Correctness-Check finds no error, for each activity* a *a data instance subgraph exists, represented by the solution to the CSP model (line 9), that triggers* a *(line 8) and whose valuation, represented by the assignment of variables to the CSP variables, satisfies the precondition $pre(\mathrm{a})$ of* a *(line 8). Therefore, each activity is may-executable, and therefore the workflow data graph is may-correct.*
$\Leftarrow$: *If $(Act, V, D, C, E, pre, post)$ is not may-correct, then there is an activity* a *that is not may-executable. By Definition 5.2.2, every data instance subgraph that triggers* a *can only have a valuation that violates the precondition of* a. *Therefore, the CSP model (line 8) has no solution (line 10).* □

The performance of the algorithm is discussed in Section 5.4.

5.3.4 Algorithm for Must-correctness

To verify must-correctness of a workflow data graph, we developed an algorithm (Figure 5.7) that diagnoses whether each activity in the workflow data graph is must-executable. If an activity a is not must-executable, the algorithm provides a counter example in the form of a data instance subgraph that triggers a and whose valuation violates the precondition of a. If every activity is must-executable, the workflow data graph is must-correct by definition.
First, the combined IP and CSP model is created (line 4 and line 5) as defined in Section 5.3.1. As in algorithm May-Correctness-Check, each data instance subgraph is a solution to this CSP model extended with additional constraints. Next, the algorithm performs a loop that processes each activity of the input workflow data graph (line 6). Variable *current* stores the activity processed in the loop. The algorithm extends for *current* the combined IP and CSP model with constraints that state that the data instance subgraph triggers *current* and that the precondition of *current* is violated. If a solution to this extended CSP model exists (line 10) then there is a data instance subgraph that triggers *current* and whose precondition violates *current*. Therefore, *current* is not must-executable (line 11). Otherwise, *current* is must-executable and the next activity is processed. If every activity is must-executable, the workflow data graph is must-correct (line 17).
We next prove that the algorithm is correct.

Theorem 3. *Let $(Act, V, D, C, E, pre, post)$ be a workflow data graph, Algorithm Must-Correctness-Check finds no error if and only if $(Act, V, D, C, E, pre, post)$ is must-correct.*

Proof 3. $\Longrightarrow$: *If algorithm Must-Correctness-Check finds no error, for each activity* a *no data instance subgraph exists that triggers* a *(line 8) and whose valuation, represented by the assignment of variables to the CSP variables, satisfies the negation of the precondition* $pre(\mathrm{a})$ *of* a *(line 8). Equivalently, each data instance subgraph that triggers* a *has a valuation that satisfies the precondition* $pre(\mathrm{a})$. *Therefore, each activity is must-executable, and therefore the workflow data graph is must-correct.*
$\Leftarrow$: *If $(Act, V, D, C, E, pre, post)$ is not must-correct, then there is an activity* a *that is not must-executable. By Definition 5.2.2, there exists a data instance subgraph that triggers* a *and that has a valuation that violates the precondition* $pre(\mathrm{a})$ *of* a*. Therefore, the CSP model (line 8) has a solution and the activity is not must-executable (line 11), so the algorithm finds an error.* □

```
 1: procedure MUST-CORRECTNESS-CHECK($Act, V, D, C, E, pre, post$)
 2:     $error = false$
 3:     $unmarked = Act$
 4:     $IP$ = make IP formulation for $(Act, E)$
 5:     $CSP$ = $IP$ + CSP formulation for $(Act, pre, post)$
 6:     while $unmarked \neq \varnothing$ do
 7:         $current$ = an activity from $unmarked$
 8:         $CSP'$=$CSP$ && $inedge_1(current) = 1$ && $current = 0$ && $\neg pre(current)$
 9:         $sol$ = solve $CSP'$
10:         if $sol$ is not null then // $CSP'$ is satisfiable
11:             Print "Activity $current$ is not must-executable"
12:             $error = true$
13:         end if
14:         $unmarked = unmarked \setminus \{ current \}$
15:     end while
16:     if $error = false$ then
17:         Print "The workflow graph is must-correct"
18:     end if
19: end procedure
```

Figure 5.7: Algorithm for checking must-correctness

5.3.5 Diagnosing the Motivating Example

This section presents the results of applying the algorithms to diagnose may and must-correctness of the business process model in Figure 5.1. The business process model has no missing and no conflicting data. As explained in Section 5.3.1, the CSP model with the pre and postconditions for each activity needs to be in SSA form. Table 5.5 shows the SSA form of the pre and postconditions of the activities with the new names of the variables. Notice that for the activity R two new constraints are introduced because the variable $others$ has two new names ($others1$ and $others2$) assigned in two different branches of a XOR split. These two new constraints unify the name of the variable to $others3$ after the join.

Next, we verify the business process model for may-correctness by apply the algorithm May-Correctness-Check on the business process model in Figure 5.1 with the renamed variables and pre and postconditions as shown in Table 5.5. The algorithm determines that the business process in Figure 5.1 is may-correct, since for activity a with precondition $pre(a)$ it is always possible to find at least one data

Table 5.5: Activities with their pre and postconditions in SSA form

Activity	Precondition and Postcondition
ECR	**pre:** $true$ **post:** $true$
SAP	**pre:** $true$ **post:** $true$
D	**pre:** $sponsorship1 > 0 \vee numPapers1 > 60$ **post:** $regFee1 * 0.1 \leq dinner1 \wedge dinner1 \leq regFee1 * 0.35$
L	**pre:** $sponsorship1 > 0 \vee numPapers1 > 60$ **post:** $regFee1 * 0.1 \leq 3 * lunch1 \wedge 3 * lunch1 \leq regFee1 * 0.35$
OS	**pre:** $sponsorship1 > 0 \vee numPapers1 > 60$ **post:** $others1 \leq 0.2 * regFee1 + 0.05 * sponsorship1 \wedge$ $others1 \geq 0.05 * regFee1 + 0.05 * sponsorship1$
O	**pre:** $sponsorship1 > 0 \vee numPapers1 > 60$ **post:** $others2 \leq 0.25 * regFee1 \wedge others2 \geq 0.05 * regFee1$
R	$OS = 1 \Rightarrow others3 = others1$ $O = 1 \Rightarrow others3 = others2$ **pre:** $3 * lunch1 + dinner1 + others3 < regFee1$ **post:** $numPapers1 * 1.8 \geq confAtt1$ $\wedge numPapers1 * 0.5 \leq confAtt1$
NGS	**pre:** $confAtt1 * (3 * lunch1 + dinner1 + others3) <$ $confAtt1 * regFee1 + sponsorship1$ **post:** $guestSpeaker1 \geq 0.2 * sponsorship1 \wedge$ $guestSpeaker1 \leq sponsorship1 + 0.1 * regFee1 * confAtt1$
IGS	**pre:** $confAtt1 * (3 * lunch1 + dinner1 + others3) <$ $confAtt1 * regFee1 + sponsorship1$ **post:** $guestSpeaker2 \geq 0.4 * sponsorship1 \wedge$ $guestSpeaker2 \leq sponsorship1$

instance subgraph that triggers a and whose valuation of the variables is within the determined finite domains listed in Table 5.1 such that $pre(a)$ is satisfied.

We diagnose for must-correctness by applying the algorithm in Figure 5.7 to the business process model in Figure 5.1 in SSA form. The algorithm finds for instance an error when activity R is processed, since a data instance subgraph exists that assigns the values 200, 100, 30 and 30 to the variables $regFee$, $dinner$, $lunch$ and $others$ respectively, which makes the precondition of activity R unsatisfiable. Therefore the workflow is not must-correct.

5.4 Implementation and Empirical Evaluation

The worst-case complexity of solving CSPs is high. Therefore, this section includes implementation details and the empirical evaluation of the performance of the developed algorithms. This way, we can asses whether in practice the time it takes to diagnose semantic business process models with the algorithms is acceptable.

5.4.1 Implementation

We have implemented a tool that performs the verification algorithms of the previous section by extending the tool DiagFlow (Eshuis and Kumar, 2010). The new tool takes XPDL 1.0/2.0 models (WFMC, 2005) as input and translates them into CSPs according to the formalization presented in this chapter. For solving these CSPs, the tool uses the COMET^{TM} solver by Dynadec (Dynamic Decision Technologies, 2011). COMET^{TM} combines the methodologies used for constraint programming, linear and integer programming, constraint-based local search, and dynamic stochastic combinatorial optimization and offers a comprehensive software platform for solving complex combinatorial optimization problems.
In order to carry out our new approach, we need to define some details about the format of the input files. The original XPDL schema does not model any semantic information of the business process, so no pre and postconditions are considered. Therefore we propose the following extension of the XPDL schema, which conforms to the XPDL standard (WFMC, 2005):

1. input data of the activity (read operations)

2. output data of the activity (write operations)

3. precondition: constraint (or constraints) over the input data

4. postcondition: constraint (or constraints) over the input and output data

In order to include these data, the XPDL input file must contain several ExtendedAttributes to extend the XPDL node *Activity* (WFMC, 2005):

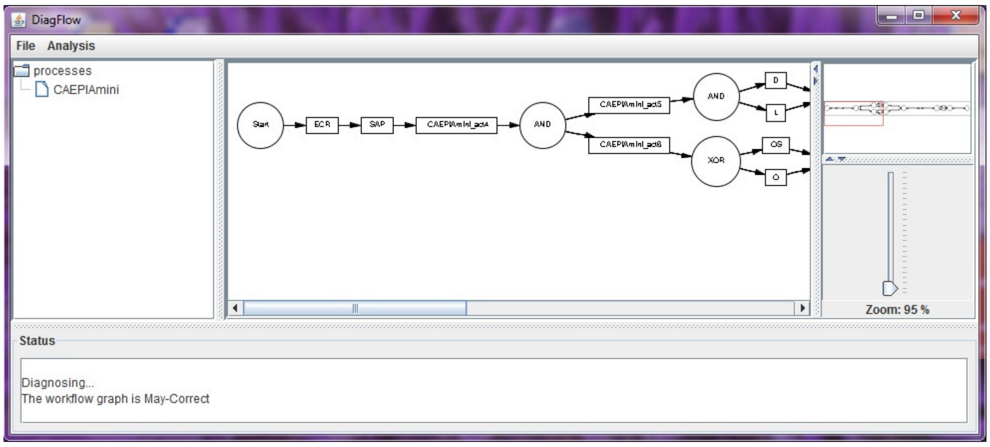
```
<xpdl:Activity ...>
  ...
  <xpdl: ExtendedAttribute Name="InputVariables">
    <variable>...</variable>
    <variable>...</variable>
    ...
  </xpdl: ExtendedAttribute>
  <xpdl: ExtendedAttribute Name="OutputVariables">
    <variable>...</variable>
    <variable>...</variable>
    ...
  </xpdl: ExtendedAttribute>
  <xpdl: ExtendedAttribute Name="Precondition" Value="..."/>
  <xpdl: ExtendedAttribute Name="Postcondition" Value="..."/>
  ...
</xpdl:Activity>
```

The variables that are referenced in the pre and postconditions of the activities use finite domains to define the ranges of values they can take. To define those domains in the XPDL file, it is necessary to add ExtendedAttributes within the node *WorkflowProcess*:

```
<xpdl:WorkflowProcess ...>
  ...
  <xpdl:ExtendedAttributes>
    <xpdl:ExtendedAttribute Name="Domain">
      <variable>...</variable>
      <initialValue>...</initialValue>
      <finalValue>...</finalValue>
    </xpdl: ExtendedAttribute>
    ...
  </xpdl:ExtendedAttributes>
  ...
```

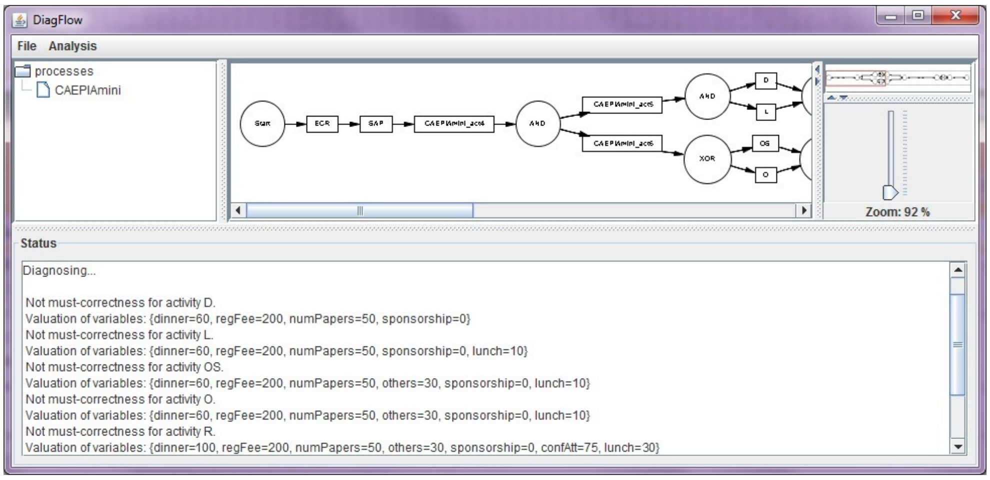
```
</xpdl:WorkflowProcess>
```

Figure 5.8 shows a screenshot of the DiagFlow tool after the verification of the may-correctness of the example discussed in this chapter (Figure 5.1). According to the pre and postconditions of its activities and the domains of the variables, the DiagFlow tool determines it is may-correct. On the other hand, the business process is not must-correct as can be seen in the screenshot in Figure 5.9.



```
Diagnosing...
The workflow graph is May-Correct
```

Figure 5.8: Screenshot of DiagFlow indicating may-correctness



```
Diagnosing...

Not must-correctness for activity D.
Valuation of variables: {dinner=60, regFee=200, numPapers=50, sponsorship=0}
Not must-correctness for activity L.
Valuation of variables: {dinner=60, regFee=200, numPapers=50, sponsorship=0, lunch=10}
Not must-correctness for activity OS.
Valuation of variables: {dinner=60, regFee=200, numPapers=50, others=30, sponsorship=0, lunch=10}
Not must-correctness for activity O.
Valuation of variables: {dinner=60, regFee=200, numPapers=50, others=30, sponsorship=0, lunch=10}
Not must-correctness for activity R.
Valuation of variables: {dinner=100, regFee=200, numPapers=50, others=30, sponsorship=0, confAtt=75, lunch=30}
```

Figure 5.9: Screenshot of DiagFlow indicating no must-correctness

5.4.2 Experimental Design

The DiagFlow tool receives XPDL files as inputs, with the extensions explained in the previous section, and provides options to verify the different kinds of correctness.
The primary purpose of the experimental evaluation is to determine the execution time from start to completion of a correctness checking process over workflow data graphs with different control flows and data flows.
With the aim of performing the execution time measurements, the algorithms are executed over different extended XPDL files, getting test cases with different number of activities, control nodes and data.
The test cases are measured using a Windows 7 machine, with an Intel Core 2 Duo processor, 1.86GHz and 2.0Gb RAM.

5.4.3 Performance Results

In this subsection, the execution time of the correctness checking algorithms is measured.
Solving CSPs takes exponential time due to the dependency of their complexity on the number of values each variable can take (Traxler, 2008; Dechter, 1992; Kumar, 1992). However, in practice, since CSP solvers run very fast, this does not limit the applicability of our approach, as we will show next.
Since both algorithms for checking may and must-correctness have a similar structure, a while-loop which processes every activity by solving a combined IP and CSP model for the activity, they also have the same time complexity. Since the algorithms check all the activities in the business process to find the ones which are not may or must-executable, the performance results depend on the size of the workflow data graph being analysed.
Figure 5.10 shows the performance results of the May-correctness-check algorithm in terms of the business process size (number of activities), including the range of the Control-flow Complexity metric (CFC, Cardoso (2005)) in brackets. That metric is used to quantify the presence of control nodes in a workflow W, defined as follows:

$$CFC(W) = \sum_{n \in S_{XOR}} CFC_{S_{XOR}}(n) + \sum_{n \in S_{AND}} CFC_{S_{AND}}(n)$$

where $CFC_{S_{XOR}}(n) = |outedge(n)|$ and $CFC_{S_{AND}}(n) = 1$. The results obtained for the must-correctness-check algorithm are not shown since they are very similar.

For both algorithms, the execution time scales linearly with respect to the number of activities checked by the algorithm. Therefore, the time it takes to diagnose the faults in even large business process models with the algorithms is acceptable.

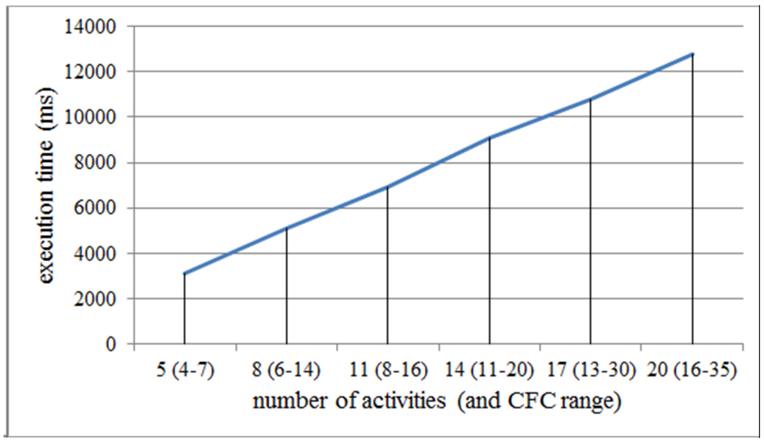


Figure 5.10: Performance results

5.5 Summary

This chapter provides an approach for diagnosing the correctness of semantic business process models, whose activities count on pre and postconditions to represent their semantics. This way, we check if the execution of a business process may get stuck at any activity due to the no satisfaction of the precondition under certain conditions. In that case, the model of the business process is considered incorrect. To this end, we distinguish between two different notions of correctness to diagnose such errors: (1) may-correctness, that is used to verify if every activity in a business process can be executed at least once, so there is an execution in which the activity is done; and (2) must-correctness, to check if every possible execution of a business process that reaches an activity satisfies the precondition of the activity.

Our contribution presents an automatic model-based diagnosis method, which is performed at design-time by using Artificial Intelligence techniques to compute the execution instances allowed by a business process model. This way, it is possible to determine errors in the model before the business process is enacted.

To this end, the semantic business process model has been translated into two models: (1) an Integer Programming model (IP model), to determine the different instances of execution of the business process, chosen due to its efficient solving; and (2) the pre and postconditions of the activities are modelled as constraints in a Constraint Satisfaction Problem (CSP), following a proposed BNF grammar to avoid any ambiguity.

The proposed diagnosis process consists of several phases. First, preprocessing is applied to detect basic data anomalies. Then, the workflow data graph is translated into an IP formulation that models the executable instances, and into a CSP formulation that models the data states acceptably according to the pre and postconditions of the activities. The combined IP and CSP models can be efficiently solved using Constraint Programming techniques. In case of error, feedback is provided in the form of an error path showing where the business process gets stuck under certain conditions over the data flow. Such feedback provides valid information for the business process designer to fix future errors before the workflow is deployed. The approach is complete, so it always generates accurate feedback in case of an error.

The approach has been implemented by extending the DiagFlow tool Eshuis and Kumar (2010). The tool diagnoses semantic business process models in an extended XPDL format. Therefore, an XPDL extension has been provided to store the semantic information of each business process model, adding the data flow with the pre and postconditions in the activities. Performance evaluation of the tool shows that the algorithms scale well for large business process models with data flows, despite the high worst-case complexity of solving constraints satisfaction programs.

Part IV

Contribution II: Diagnosability in Business Processes

Chapter 6

Improving the Diagnosability in Business Process Models

6.1 Introduction

A business process configured and enacted from a correct model may present abnormal behaviour during its execution. This is due to activities composing the process may work incorrectly, or certain data managed throughout the data flow are wrong. This can be derived from the expected behaviour has not to be equal to how each activity actually works. In order to detect and diagnose these incorrect activities or data, it is necessary to have the possibility to observe the actual behaviour.

Diagnosability analysis aims to determine whether observations available during the execution of a system are sufficient to precisely locate the source of a problem. Previous work already dealt with the diagnosability problem in other contexts, such as circuits and systems. This chapter presents a contribution which addresses the diagnosability problem in business processes to provide the functionality of the task *Improvement of Diagnosability* in the diagram of Figure 1.2. Therefore, the aim of this contribution is to determine a test-point allocation to monitor sufficient observable data in the data flow to allow the discrimination of faults for a later diagnosis process at run-time.

The objective to achieve is formally stated in terms of an optimization problem, and reaches a given objective function depending on the business policy, under a set of constraints due to the availability of test-point resources within the domain. Three test-point allocation strategies are proposed to ensure the best business process diagnosability, and are implemented in the Test-Point Allocator tool. Experimental results indicate that optimal test-point allocation yields optimal cost and a shorter time to diagnosis.

6.1.1 Motivation

A correctly designed business process may present abnormal behaviour due to incorrect managed data, or activities which do not operate as they were modelled. The fault detection and later diagnosis of this abnormal behaviour of business processes are crucial from the strategic point of view of the organizations, since their proper working is an essential requirement, being crucial to efficiently diagnose faults which take place at run-time in order to get competitive companies. Unexpected faults can provide undesirable halts in the processes, thereby causing cost increase and production decrease. Therefore, to maintain business processes at desirable reliability and production levels, it is necessary to develop techniques to detect and diagnose their faults.

When a business process is executed, certain errors can be detected. The diagnosis process is used for the isolation of those activities or sub-processes which are responsible for any incorrect behaviour within the whole process.

The monitoring of a business process, and later diagnosis, are based on observations, which provide information about the behaviour of the process. For the later diagnosis process to be successful, diagnosability analysis becomes a design-time requirement. The diagnosability level of the business process depends upon the observations. If the diagnosis is based on few observations, or if observations are not allocated at the most convenient places, it is very difficult to distinguish which parts of the business process are failing. Both the number of observations and the location where they are performed enable the cause of the error to be precisely located. In general, diagnosis systems not only incorporate identification and isolation of faults, but also the monitoring. The explanation of abnormal behaviour, from a determined observation derived from the monitoring, is the main task of diagnosis.

Since there is no single entity which provides an overall view of the complete data-flow model of a business process, it is desirable to improve the monitoring of business processes to be fully aware of possible deviations from expected behaviour. To this end, observations can be carried out at certain intermediate flows of business processes, and not only at the output, by means of the allocation of test points. A test point can be allocated in the *flow* (sequence flow, conditional

flow or default flow according to BPMN 2.0 (OMG, 2011)) in order to guarantee the observability of the data flow at that point during the execution of a business process instance. Previous work in the literature deals with the diagnosability analysis problem (Console et al., 2000; Travé-Massuyès et al., 2006; Dressler and Struss, 2003; Bocconi et al., 2007) for other scenarios, but these proposals cannot be adapted directly to business processes since, although some of them allocate sensors to perform observations, they try to get fully diagnosable systems, not considering requirements like cost or time limitations.
A business process with test points is said to be diagnosable (adapted from Console et al. (2000)) if and only if:

- For any relevant combination of test-point observations, there is only one minimal candidate diagnosis.
- Each fault of the business process corresponds to at least one candidate diagnosis for a certain test-point observation.

The allocation of the test points is therefore a crucial task, since if the test points are not correctly allocated, then the observational model may be rendered useless for the isolation of faults, and hence the diagnosis process would fail to determine the activities which are not behaving as they were modelled.

6.1.2 Contribution

The aim of this contribution is to apply techniques for the allocation of test points in business processes in order to improve the computational efficiency of isolating faults in the diagnosis process, thereby rendering the business process diagnosable.
It is possible to consider that, in order to obtain complete diagnosability, the best solution is to allocate test points at all possible locations, but this solution presents certain problems:

- Minimization of the number of transmissions (in this case, observations) is very important in data-centric business processes.
- Some parts of a business process may not be observable due to confidentiality, privacy and security policies.
- The monitoring and evaluation of the observations require extra human resources, with the consequent cost increase.

Therefore, for the optimal allocation of test points, there is a trade-off between diagnosability and the features listed above. The current contribution opts for the selection of those test points which are necessary and economically feasible to achieve the diagnosability level desired, and surrender one benefit to gain others. After the allocation, the business process is divided into sets of activities without intermediate observations, called clusters. In this chapter, three independent objectives are sought depending on the requirements, which can entail economic limitations, the achievement of a certain diagnosability level, or even the optimization of the execution time of a later diagnosis process.

Each business process is described as a set of activities related by means of control structures and data exchange. This, together with the requirement for automatic modelling and reasoning in the problem of allocating test points, the Constraint Programming paradigm is proposed since the specification of the variables from the data flow associated to the activities (according to the Business Process Modelling Notation (BPMN) model) can be modelled as a Constraint Satisfaction Problem (CSP).

In this contribution, the flows between activities in the business processes are assumed to be not faulty. Nevertheless, if the flows have to be considered as candidates for the responsibility of any abnormal behaviour, this possibility can easily be modelled by adding a new fictitious component per flow which would be considered during the test-point allocation and later diagnosis processes.

This contribution has been implemented in the Test-Point Allocator tool (Borrego et al., 2011). The tool analyses BPMN models and provides options for the allocation of test points according to the three objectives previously mentioned.

The remainder of the chapter is organized as follows. Next subsection presents an example to illustrate the concept of diagnosability in business processes. Section 6.2 defines concepts related to diagnosability and introduces the three objectives in greater depth. Section 6.3 details the methodology used in the allocation of test points in business processes. Section 6.4 gives implementation details and shows experimental results. And finally, a summary of the contribution is outlined in Section 6.5.

6.1.3 Illustrative Example

This section introduces an example of a business process called *Arrival of a New Employee*, extracted from the Bonita Open Solution documentation (BOS, 2011), shown in Figure 6.1 using BPMN 2.0 for the graphical representation.

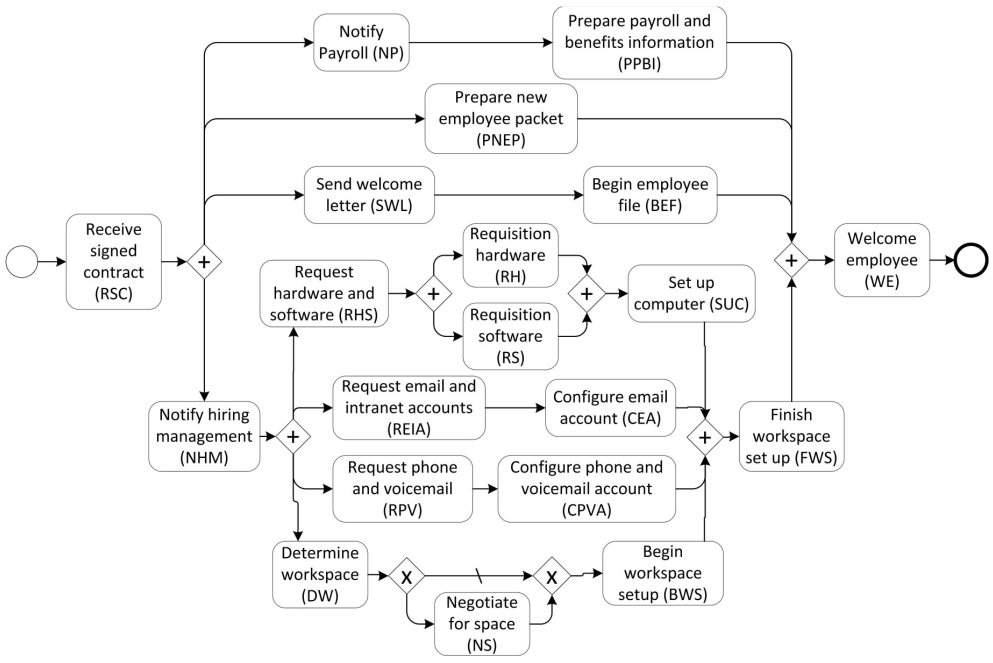

Notify Payroll (NP)
Prepare payroll and benefits information (PPBI)
Prepare new employee packet (PNEP)
Send welcome letter (SWL)
Begin employee file (BEF)
Receive signed contract (RSC)
Requisition hardware (RH)
Request hardware and software (RHS)
Requisition software (RS)
Set up computer (SUC)
Welcome employee (WE)
Request email and intranet accounts (REIA)
Configure email account (CEA)
Notify hiring management (NHM)
Finish workspace set up (FWS)
Request phone and voicemail (RPV)
Configure phone and voicemail account (CPVA)
Determine workspace (DW)
Begin workspace setup (BWS)
Negotiate for space (NS)

Figure 6.1: Motivating example

Figure 6.1 shows a business process that is executed when a new employee joins the company, and entails the tasks of data notification and of preparation of the new workspace. This process consists of twenty activities (rectangles with rounded corners) and eight gateways or control nodes (diamonds), and a start and end event (circles). A gateway with one incoming edge and multiple outgoing edges is called a split; a gateway with multiple incoming edges and one outgoing edge is called a join.

The activities in the example consume and produce data during the execution of the business process. This flow of data between activities remains unobservable to an external diagnoser until the execution finishes and the end event is reached.

After the execution of an instance of the business process, the observed result may differ from the expected result. This implies that a certain activity or activities within the process present abnormal behaviour. Taking into account that, without the allocation of test points, it is possible to observe data only at the end of the process, and hence the question to answer is: how can the activity or activities which are responsible for the incorrect behaviour of the process be identified? By counting on only the observations at the end of the process, it is impossible to

isolate those activities which cause the fault since they cannot be discriminated from the rest.
For example, if, after the execution of the process in Figure 6.1, the computer that was prepared for the new employee does not include the expected software, was there a mistake in the data notified in the execution of the *Notify hiring management* activity? Or maybe the incorrect behaviour took place at *Requisition software*?
By means of the allocation of test points, we ensure a higher diagnosability of the processes, thereby easing discrimination between activities.

6.2 Diagnosability of Business Processes

In order to ensure a degree of diagnosability of business processes, we propose an approach based on Artificial Intelligence techniques. Before applying these techniques, some definitions are introduced.

6.2.1 Definitions

A diagnosis of a business process, such as the one shown in Figure 6.1, can be defined as:

Definition 6.1. *A diagnosis of a business process is a particular hypothesis which shows that the current behaviour of the process differs from its expected behaviour. Any activity has two possible fault modes, (i) it is working correctly (ok mode), or (ii) it is faulty (abnormal mode). Thus, the diagnosis space for the business process initially consists of* $2^{|Act|} - 1$ *diagnoses (de Kleer et al., 1992), where* $|Act|$ *is the number of activities in the business process. The goal of diagnosis is to identify and refine the set of minimal diagnoses.*

The problem is that, without any monitoring of the data flow during the execution of a business process instance, it is not possible to discriminate between the fault modes of the activities, leading us to a business process where it is not possible to distinguish the activity or activities responsible for the abnormal behaviour.

Definition 6.2. *(Bocconi et al., 2007) Two fault modes are discriminable if their patterns of observable values are disjoint.*

Therefore, without the allocation of any test point, the data flowing between activities remain unobservable until the end of the execution of a business process instance, since it is only one cluster, defined formally as follows:

Definition 6.3. *A set of activities C is a cluster of activities: (i) if no unobservable information exists between any activity within the cluster with any activity outside the cluster; and (ii) if the set of activities is minimal, then for all $Q \subset C$, Q is not a cluster of activities.*

The allocation of test points allow us to divide the business processes into several clusters.

Definition 6.4. *A Test point is a location within the flow of a business process where the observability of the data flow is guaranteed.*

The allocation of test points allows the diagnosability of business processes to be improved, by monitoring the data flow to locate the source of abnormal behaviour, in accordance with the following definitions.

Definition 6.5. *Diagnosability level is the quotient of the number of (classes of) faults which can be discriminated from each other, and the number of all possible faults. Since the number of activities in a business process is $|Act|$, and since only one fault per activity is possible, the maximum number of the possible faults is initially* $2^{|Act|} - 1$.

Definition 6.6. *(adapted from Console et al. (2000)) A business process is diagnosable with a given set of test points TP if and only if: (i) for any relevant combination of test-point readings there is only one minimal diagnosis candidate; and (ii) every fault of the business process belongs to a candidate diagnosis for certain test-point readings.*

6.2.2 Objectives for the Improvement of the Diagnosability of a Business Process

The test points enable the activities of the business process to be separated into different clusters, thereby improving the computational efficiency in the diagnosis process and making it easier to isolate incorrect activities. This is possible since the diagnosis of the whole business process can be performed based on the diagnosis of each cluster separately. Adapting the idea in Ceballos et al. (2005) to business processes, being A the set of activities of a business process, with n activities, this set can be divided into two clusters C_1 and C_2, with n - m and m activities respectively ($n > m$), such that $C_1 \cup C_2 = A$. When it comes to detecting the conflicts, the computational complexity in either of the separate clusters is lower than in the whole business process A, since the number of possible diagnoses of C_1 and C_2 is

$(2^{n-m}) + (2^m) - 2 << 2^{n-m} \cdot 2^m - 2$

which is less than the computational complexity for the whole set of activities A, $2^n - 1$.

According to these calculations, it is possible to consider that the best solution to the problem is the allocation of test points at every possible location in the business process, thereby isolating each activity from the rest and obtaining a cluster per activity. However, this solution is infeasible due to economic and/or privacy policies:

- The test-point readings take place during the execution of an instance of a business process. In the case of abnormal behaviour of the process, these readings have to be monitored and evaluated by extra human resources, since this IT service is not automated. This issue entails hiring, training and retaining skilled personnel, with the consequent cost increase.

- Likewise, the test-point readings must be handled by a fault-diagnosis system, which is external to the business process. Minimizing the number of transmissions is vital in data-centric business processes. Even in the case when the test points are attached to business process management systems with ample power supply, the reduction in bandwidth consumption may still be a major factor due to the wireless, multi-hop nature of communication and short-range radios.

- Furthermore, not all the possible locations in a business process can include a test point: (i) due to legal regulations (e.g. SOX, HIPAA) and data protection acts, some data flowing through a business process may not be observable due to confidentiality, privacy and security policies; (ii) certain parts of a business process cannot include an observational model which enables the determination of whether the observation performed by a test point at that location differs from the expected observation. Hence, any information collected by the test point is useless from the diagnosis point of view.

Based on these points, the contribution presented in this chapter applies techniques to minimize the number of test points allocated in a business process while ensuring a determined diagnosability level. This brings us to the three different objectives for consideration:

- *Objective 1:* If the requirements entail cost limitations, and assuming that the cost of allocating a test point is independent of the location within the business process, this involves a fixed maximum number of test points to allocate. In this case, the objective is to reach the highest possible diagnosability level while obeying the economic limitation restriction. Therefore, our approach allocates that fixed number of test points and obtains the maximum possible number of clusters.

- *Objective 2:* If the requirements aim to optimize the execution time of the later diagnosis process, since the complexity of the diagnosis methods (Reiter, 1987) is exponential in terms of the number of activities, then the objective becomes the minimization of the number of test points to allocate in order to obtain a fixed number of balanced clusters.

- *Objective 3:* If the requirements entail a certain diagnosability level, then a maximum number of activities per cluster is fixed, and the objective becomes the minimization of the number of test points to allocate in order to obtain that level.

6.3 Test-Point Allocation Methodology

A graphical representation of the architecture proposed for the solution of the diagnosability problem is shown in Figure 6.2.
This contribution presents a set of steps to locate test points which improve the diagnosability and computational efficiency of the diagnosis process:

1. Previously, business processes have already been modelled using BPMN. The modelling of business processes using this kind of graphical representation entails the automatic generation of .bpmn files which contain this same visual information in XML format. This information is interpreted as input to a Test-Point Allocator to attain an optimal allocation of test points.

2. The Test-Point Allocator analyses these XML files, which contain the information of the overall business process. Each business process in an XML file is denoted by the label `<pools>`, and, within each business process, every activity and gateway has the format:

```
<vertices xmi:type="bpmn:Activity" xmi:id="..." name="..."
outgoingEdges="..." incomingEdges="..." activityType="..." ... />
```

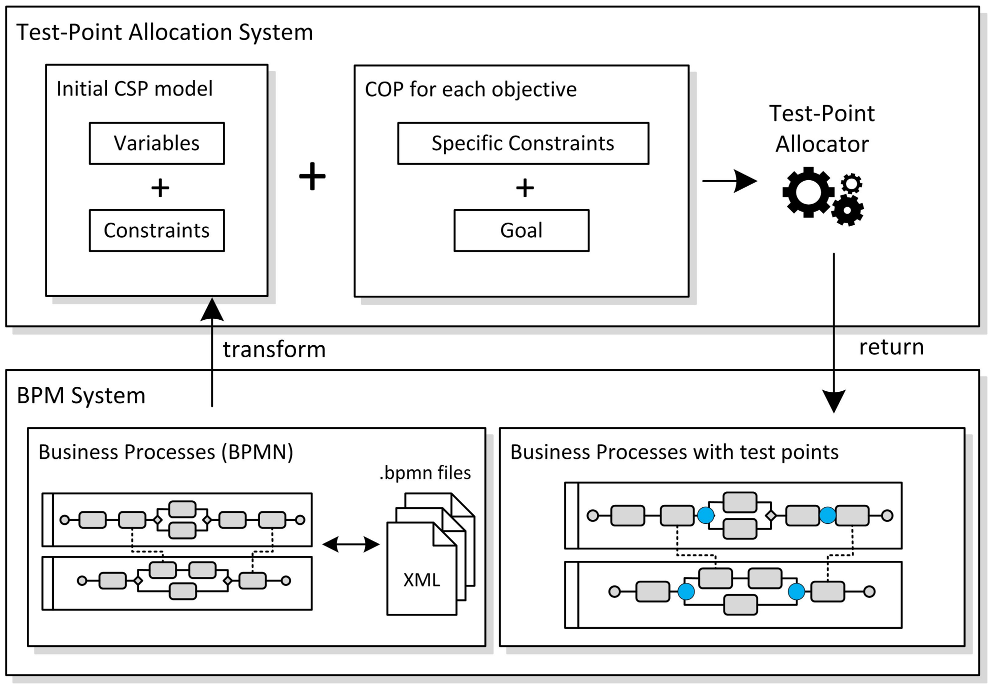


Figure 6.2: Graphical representation of the approach

3. These elements in the XML file are modelled as variables and constraints in a Constraint Satisfaction Problem according to the topology of the business processes. This step is laid out in detail in the following subsection.

4. Depending on which objective is to be achieved, the initial CSP is modified to include some specific constraints and goals. Then, Constraint Programming techniques are used for the allocation of test points.

5. The result is provided by showing a graphical representation of the business processes, including a label on the flows where the test points should be allocated.

Due to the morphological similarities between business processes and hypergraphs, where the activities and flows could be equivalent to nodes and edges, this contribution may resemble well-known hypergraph-partitioning problems (Karypis et al., 1999; Çatalyürek and Aykanat, 1999). The fact of not considering the gateways as nodes transforms those the edges whose source or target node is a split into hyperedges that connect the previous and subsequent nodes of the split. As an example, in Figure 6.3, the activities and split in a) are represented as a hypergraph in b), where there is only one single possible location to observe the data flow, which is the output of *Node A* (i.e. hyperedge e_i).

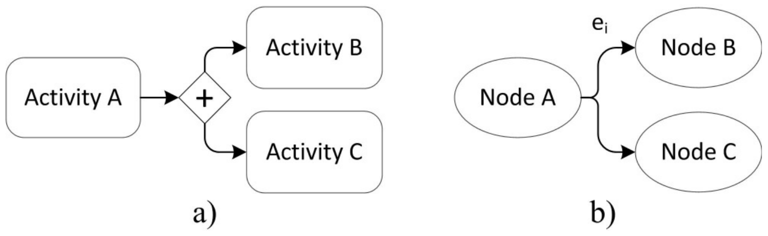


Figure 6.3: Example of hyperedge transformation

Those mentioned works (Karypis et al., 1999; Çatalyürek and Aykanat, 1999) in hypergraph partitioning are not applicable to business processes since they fail to comply with the allocation criterion derived from the features of business processes. That is, since the allocation of a test point in the flow of a business process implies that the data flow at that location become observable, when a test point is allocated in a hyperedge then all the activities connected by means of that hyperedge no longer share unobservable data. Hence, they could be located in different clusters.
However, if, for example, the multilevel hypergraph-partitioning algorithm *hMETIS* (presented by Karypis et al. (1999)) is applied to divide a business process into clusters, it can be observed that the cut of a hyperedge does not separate all the nodes connected through it, but only one node is considered in a cluster different to the remaining nodes. For the example in Figure 6.3 b), a hyperedge cut of e_i by *hMETIS* divides the hypergraph into two clusters (for example, {*Node A, Node B*} and {*Node C*}), whereas our method allocates only one test point in order to obtain a cluster for each node. Therefore, to attain the same diagnosability level, the existing hypergraph-partitioning methods need more cuts than does our approach.

6.3.1 Improving Diagnosability Using Constraint Programming

In order to automate the reasoning for the allocation of test points in accordance with certain objectives, the Constraint Programming paradigm is chosen since it enables the modelling of both the topology and the properties of the problem to be solved as CSPs, even allowing the determination of the objective function to reach.

From each designed business process, an initial CSP is defined, that is enriched with specific constraints and an objective function depending on the objective to achieve. Hence, a Constraint Optimization Problem (COP) is attained.
In order to model the topology of the business process (i.e., the activities and flows between them) as an initial CSP, the main variables and constraints are obtained automatically from the business process in the .bpmn file. With *nAct* as the number of activities in the business process, and $nFlow$ the number of flows between them, the variables of the initial CSP can be defined as the following:

- ▷ *clusterOfAct*$_i$: set of *nAct* variables which represents the cluster where each activity i is contained.
- ▷ *testPoint*$_j$: set of $nFlow$ variables to hold the test points allocated in the flows of the business process. The possible values are Boolean: $true$ implies that there must be a test point in a determined flow, and $false$ means that there is no test point in a determined flow.
- ▷ *nTestPoints*: variable which holds the number of allocated test points.
- ▷ *nClusters*: variable which holds the number of obtained clusters.

Once the variables are defined, our method automatically transforms each flow in the business process into a constraint within the CSP. This constraint establishes that if there is no test point in the link between any two activities, then these two activities must be in the same cluster. Likewise, this constraint is also generated for each pair of activities connected through a gateway, since gateways do not affect the data which flows from one activity to another, and hence two activities connected through a gateway are in the same cluster unless a test point is allocated in that flow.
Therefore, the initial CSP which models the business process is composed of these variables and constraints:

Variables
$clusterOfAct_i, D : \{0, \ldots, nAct - 1\}$
$testPoint_j, D : \{true, false\}$
$nTestPoints, D : \{0, \ldots, nFlow\}$
$nClusters, D : \{1, \ldots, nAct\}$
Constraint (one per flow j between each pair of activities A and B)
if($testPoint_j$ = false) $\Rightarrow clusterOfAct_A = clusterOfAct_B$

Each created constraint indicates that if there is no test point allocated in the flow between A and B, then A and B are necessarily in the same cluster. It is impossible to state the opposite, since the existence of a test point in the flow between A and B cannot imply that A and B are in different clusters since it is possible that these two activities are connected through another path in the business process.

Likewise, to model a flow k where a test point cannot be allocated for reasons explained in Section 6.2.2, it only has to be indicated that the variable $testPoint_k$ is always equal to *false*.

For instance, Figure 6.4 shows the business process of Figure 6.1 with two test points allocated ($nTestPoints = 2$). As a result, two clusters have been attained ($nClusters = 2$, with cluster 0 and cluster 1), represented in the figure by different colours of the activities. This way, and considering the order of the activities as $\{RSC, NP, PPBI, PNEP, SWL, BEF, NHM, RHS, RH, RS, SUC, REIA, CEA, RPV, CPVA, DW, NS, BWS, FWS, WE\}$, the set of variables $clusterOfAct_i$ would be as follows:

$$clusterOfAct = \{0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0\}$$

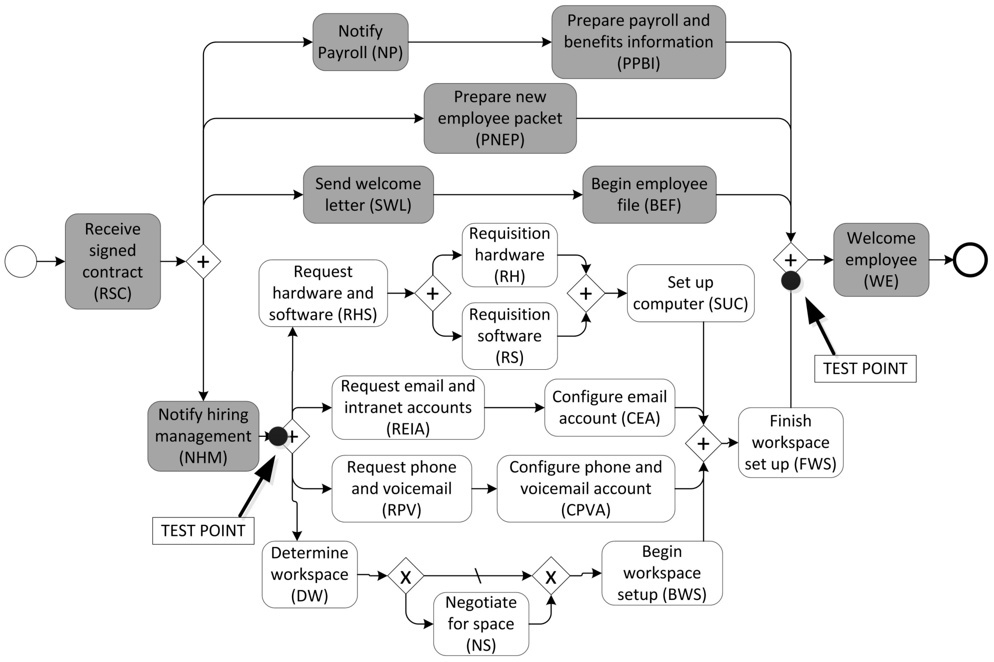

Notify Payroll (NP)
Prepare payroll and benefits information (PPBI)
Prepare new employee packet (PNEP)
Send welcome letter (SWL)
Begin employee file (BEF)
Receive signed contract (RSC)
Requisition hardware (RH)
Request hardware and software (RHS)
Set up computer (SUC)
Requisition software (RS)
Welcome employee (WE)
TEST POINT
Request email and intranet accounts (REIA)
Configure email account (CEA)
Notify hiring management (NHM)
Finish workspace set up (FWS)
Request phone and voicemail (RPV)
Configure phone and voicemail account (CPVA)
TEST POINT
Determine workspace (DW)
Begin workspace setup (BWS)
Negotiate for space (NS)

Figure 6.4: Motivating example with two test points

Furthermore, and due to the high computational complexity of this problem, additional constraints are added to the initial CSP in order to prevent repeated calculation of equivalent solutions (Mears et al., 2009). For instance, if three activities *A*, *B*, *C* are allocated within clusters 1, 1, and 2 respectively, then that solution is equivalent to their allocation within clusters 2, 2 and 3, since in both solutions *A* and *B* are allocated within the same cluster and *C* in a different cluster. The inclusion of these constraints in the model of the CSP reduces the computational complexity through its huge reduction of the search space.

Constraints

$clusterOfAct_1 = 0$

$\forall i \in \{0,\ldots,nAct-1\} : clusterOfAct_i \leq max(clusterOfAct_j) + 1$

$\quad j \in \{1,\ldots,i-1\}$

These additional constraints are common to all three objectives achieved by the solution proposed in this contribution and form the initial CSP. However, each objective needs its own additional and specific constraints and goals so that it can be completely modelled, thereby creating a COP. The three objectives and their corresponding configurations are given in detail in the following subsections.

6.3.2 Objective 1: Maximization of the Number of Clusters with the Allocation of a Fixed Number of Test Points.

In order to maximize the diagnosability, the objective here becomes the maximization of the number of clusters obtained with the fixed number of test points. Beginning with the initial CSP, it is necessary to add new constraints and the goal to be achieved for this specific objective. This new information is also generated and included in the initial CSP in an automatic way.
More precisely, the new information to be added is:

- The number of test points must be limited to a value t indicated by the user or the problem specification. That is, the number of $true$ values in the set $testPoint_j$ have to be equal to the number of test points to be allocated. If this value t is greater than or equal to the maximum number of existing flows for the allocation of test points, it is not necessary to execute any algorithm, since a test point will be allocated at each possible location of the business process.

- The goal is included in the CSP: taking into account that the assigned clusters in $clusterOfAct$ are consecutive integers beginning at 0, the objective function becomes the maximization of the maximum integer in $clusterOfAct$, thereby obtaining the maximum possible number of clusters.

Constraint
$nTestPoints = t, t \in \{0, \ldots, nFlow\}$
Goal
$maximize(max(clusterOfAct))$

Although the allocation of test points is an off-line process, the complexity and execution time can be improved, especially when the business process has a large number of activities. In order to sort out the problem of the computational complexity, the greedy algorithm presented by Ceballos et al. (2005) is used as a previous step. In short, this algorithm assigns a weight to each flow in the business process, and applies Floyd's algorithm to find the minimal path between each pair of activities. These minimal paths decide, through a voting mechanism, which paths are the bottlenecks of the business process, that is, which are the most important flows for the allocation of test points.
The result obtained from the execution of the greedy algorithm is a set with the various flows for the business process and with the determined bottlenecks. This set is used in the selection of the collection of variables in $testPoint_j$ for the best candidates for the allocation of test points.
On the other hand, the use of the greedy algorithm by Ceballos et al. (2005) to allocate the test points does not guarantee the optimal solution. Therefore, we propose using the solution provided by this greedy algorithm (that is, the number of clusters obtained) as a bound to limit the minimum number of clusters that the complete method based on COP must try to achieve. In this way, the new constraint that is added to the initial CSP becomes:

Constraint
$min(clusterOfAct) \geq numClustersAllocatedByGreedyMethod + 1$

If the complete method fails to find a solution after adding this new constraint, then the greedy method has already provided the optimal solution. Otherwise, the optimal solution is provided by the complete method, thereby attaining a much shorter execution time thanks to the use of the greedy algorithm as a bound.

6.3.3 Objective 2: Allocation of the Minimum Number of Test Points in Order to Obtain a Fixed Number of Balanced Clusters.

When the requirements of the problem aim to optimize the execution time of a later diagnosis process, it has to be considered that the complexity of the diagnosis methods is exponential in terms, of the size of the system to diagnose (Reiter, 1987), which in our case is the number of activities in the business process.
In order to model the COP to achieve this objective, it is necessary to include certain new constraints in the initial CSP, as laid out in the following:

- The number of clusters to attain becomes a fixed value called *numClusters*. This value is provided by the user or the problem specification.
- In order to attain balanced clusters, the number of activities per cluster is calculated by dividing the whole number of activities in the business process by the fixed number of clusters to obtain. We also let the user determine an extra parameter *v* to allow a range of values for the number of activities per cluster.
- The goal to be achieved is the minimization of the number of test points to be allocated. That is, an objective function is needed in order to establish the minimum number of values equal to *true* in the set *testPoints*.

Variables
$numClusters, D : \{1, \ldots, nAct\}$
$v, D : \{0, \ldots, nAct/numClusters\}$
Constraints
$nClusters = numClusters$
$minNumAct = nAct/numClusters - v$
$maxNumAct = nAct/numClusters + 1 + v$
$\forall i \in \{0, \ldots, nClusters - 1\}$
$\quad minNumAct \leq occurrences(i, clusterOfAct) \leq maxNumAct$
Goal
$minimize(nTestPoints)$

Thanks to these newly added constraints, the search space for the new COP is bounded, since, for example, the domain of the variables in the set *clusterOfAct* is drastically reduced.

Therefore, this COP presents no computational problems, since it attains a constant execution time, and hence no previous method is required to improve the temporal results.

6.3.4 Objective 3: Minimization of the Number of Test Points to be Allocated in Order to Obtain Clusters with a Maximum Number of Activities.

When it is necessary to achieve a determined diagnosability level, a maximum number of activities per cluster is fixed, and the objective becomes the minimization of the number of test points to be allocated in order to minimize costs.
In order to model this objective, it is necessary to add more information to the initial CSP. That information is:

- Constraints to limit the number of activities belonging to each cluster to the value *maxNumAct*. This value is provided by the user or the problem specification.
- The goal to be achieved is given by an objective function which establishes the minimization of the number of test points to be allocated.

Constraints
$\forall i \in \{0, \dots, nClusters - 1\}$
$\quad occurrences(i, clusterOfAct) \leq maxNumAct$
Goal
$minimize(nTestPoints)$

Once these new constraints have been included in the model of the problem, the CSP solver can be executed in order to find the optimal solution. However, the computational complexity of the problem is exponential, and hence it is necessary to add some kind of bound to reduce the search space of the variables in the COP.
In order to obtain a bound, we propose a new greedy method. This method is able to allocate test points in the business process in linear time. The solution provided by this new greedy algorithm may not be the optimal solution, but it provides a very useful bound for the number of test points to be allocated. This bound is used to drastically reduce the domain of the variables $clusterOfAct_i$.
As an example, an application of the algorithm without any bound to the business process in Figure 6.1 to obtain clusters with a maximum size of 5 activities, allocates 6 test points. When it comes to executing the greedy algorithm, it allocates

10 test points, which is obviously not an optimal solution. Nevertheless, when these 10 test points are used as a bound for the complete algorithm, the optimal solution (6 test points) can now be found in only a tenth of the time that would have been spent without the bound.

Greedy Method for the Improvement of the Computational Complexity of Objective 3.

The greedy algorithm is based on the topology of the business processes, and takes advantage of knowledge on the different control flow patterns that are used in the modelling of a business process. Since, in business processes, topological structures frequently exist where a set of branches that form a split are synchronized by means of a join, it is possible to analyse the processes in great depth. The splits and joins that appear in a business process allows us to divide it into several levels. That is, when a single thread of execution splits into two or more branches, and those branches later converge in a join, the activities in those branches are in an inferior level than the activities in the main thread.
This concept is illustrated in the example in Figure 6.5, where the business process is composed of ten activities and three AND-splits with their three corresponding AND-joins. These splits and joins divide the business process into four levels:

- Level 1: the main level, composed of all the activities $A0$, $A1$, $A2$, $A3$, $A4$, $A5$, $A6$, $A7$, $A8$ and $A9$.
- Level 2: composed of the activities $A1$, $A2$, $A3$, $A4$, $A5$, $A6$, $A7$ and $A8$, which are the activities within the outer split and join.
- Level 3: where the activities $A2$ and $A3$ are included. These are the activities within the second split and join (one of the inner splits and joins).
- Level 4: with the activities $A6$ and $A7$ (within the other split and join).

Based on this idea of levels within the business process, the greedy algorithm consists of two steps:

i. **Labelling the activities.** This first step assigns a label to each activity in order to store the level where the activity is within the business process (this information is obtained in this step of the algorithm). This task is performed beginning in the start event of the business process and following the direction of the flow up to the end event. During this walk through the business

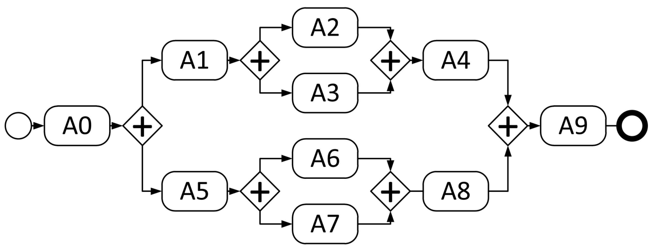


Figure 6.5: Business process with four levels

process, the activities are labelled depending on the level where they are located. In order to do this, the splits are matched to their corresponding joins, and the labels are assigned from upper to lower levels.

The label of the main level (this level includes the whole business process) is the string ”1”. The labels assigned to the activities in the rest of the levels are formed by the label of the upper level, linked at the end to a number that will represent the new level. Continuing with the example in Figure 6.5, this business process together with the different labels assigned to its activities are shown in Figure 6.6.

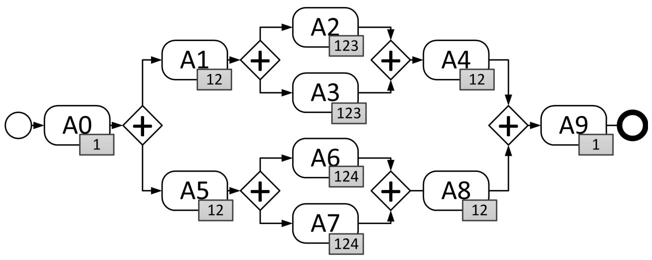


Figure 6.6: Business process with labels in the activities

The four levels in this business process are labelled as ”1”, ”12”, ”123”, and ”124”. At the same time as these labels are assigned, a tree with the hierarchy of levels is built. Each node of this tree stores the label of a level and the activities of the business process which are previous and subsequent to that level.

For the example in Figure 6.6, the corresponding tree of levels is presented in Figure 6.7. The information shown in a node of the tree is a level in the business process and the previous and subsequent activities for that level.

Obviously, level ”1” has no previous nor subsequent activity, since that level represents the whole business process.

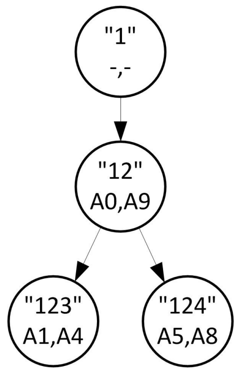


Figure 6.7: Tree of levels

ii. **Allocating the test points.** Using the tree of levels built in Step (i) above, this task performs a recursive process over the levels to allocate the test points. The algorithm in Figure 6.8 shows this recursive process in order to clarify the execution of this task.

   In detail, the algorithm is executed beginning in the level which is the root of the tree of levels. The algorithm traverses the levels recursively, allocating test points from lower to upper levels. The allocation of test points takes place in *currentLevel* if:

   - *currentLevel* includes more activities than the permitted amount per cluster, and (i) it is a leaf of the tree (lines 4 and 5) or (ii) the test points have already been allocated on its children (lines 15 and 16).
   - *currentLevel* includes the same number of activities as the maximum permitted per cluster (line 19).

   The most important statements, with the marks (1), (2), and (3) in the algorithm, are explained in the following.

   (1) Allocating test points in the input and outputs of a level: when it comes to the allocation of test points in a level, the aim is to isolate the activities of that level from the rest of the activities in the business process. Therefore, this statement entails the action of allocating test points after the previous activity and before the subsequent activity of the level.

```
procedure GreedyMethod(String currentLevel)
    if there are more activities in currentLevel than the permitted activities per cluster then
        if currentLevel is a leaf of the tree of levels then
            (1) allocate test points in the input and outputs of currentLevel
            (2) allocate test points in the activities of currentLevel
        else
            for all children c of currentLevel in the tree do
                recursive call: run this algorithm over activities in level c
                if any test point is allocated in level c then
                    (3) reduction of the business process
                else
                    //activities in c will be taken into account in the upper level
                end if
            end for
            (1) allocate test points in the input and outputs of currentLevel
            (2) allocate test points in the activities of currentLevel
        end if
    else if the number of activities in currentLevel is equal to the maximum activities per cluster then
        (1) allocate test points in the input and outputs of currentLevel
    else
        //activities in currentLevel will be taken into account in the upper level
    end if
end procedure
```

Figure 6.8: Greedy recursive algorithm for the allocation of test points

For example, if test points must be allocated in the business process in Figure 6.6 and the maximum number of activities per cluster is four, then the recursive algorithm begins in level ”1” (root of the tree in Figure 6.7) that contains ten activities (the whole business process). Since this level contains too many activities and it is not a leaf in the tree, the algorithm moves forward to the only child in the tree: level ”12”. The number of activities in this level (eight activities) is also larger than the maximum per cluster, and hence this level must be studied independently of the rest of the activities in the business process and must be isolated. To this end, three test points are allocated: one in the input of the level (output of $A0$) and two in the outputs of the level (outputs of $A4$ and $A8$).

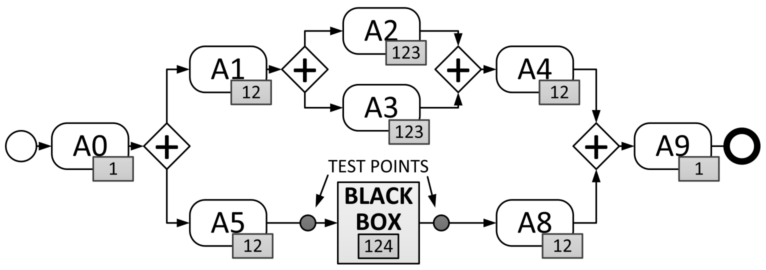


Figure 6.9: Reduced business process

(2) Allocating test points in the activities of a level: either because the level is a leaf or because it has already been isolated. This statement entails the action of allocating test points in the activities of a level using the complete search for the corresponding CSP, as explained at the beginning of this Subsection (6.3.4).

(3) Reduction of the business process: once the test points have been allocated in a level, this level can be considered as a *black box* in upper levels. Therefore the business process is effectively reduced in the allocation of test points over the whole business process since the activities of that level are not taken into account.

For example, supposing that, in the business process in Figure 6.6, the level ”124” has been isolated and the test points have already been allocated on this level. This level can therefore be replaced by a *black box* delimited by test points, as shown in Figure 6.9.

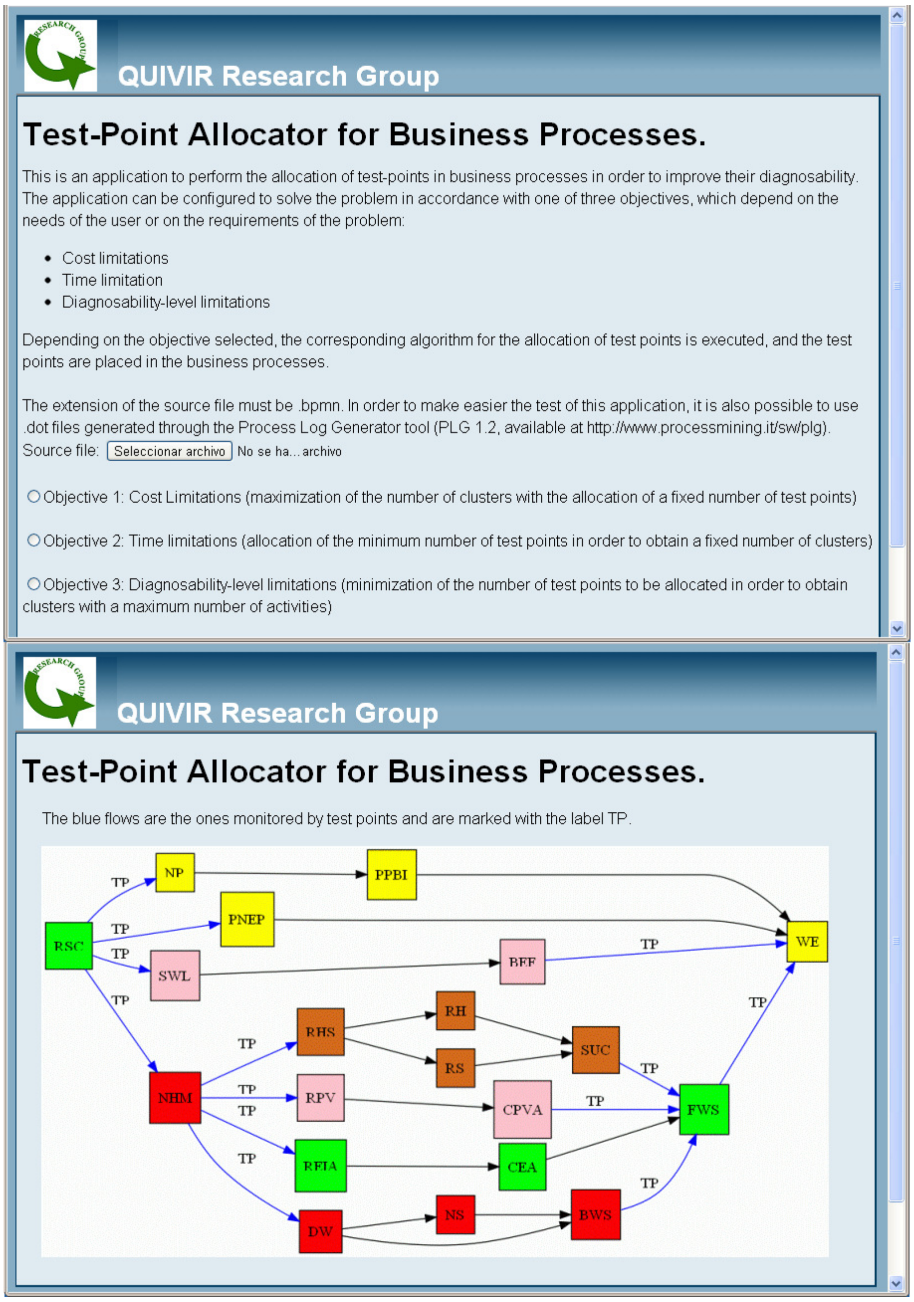
6.4 Implementation and Empirical Evaluation

In this section, the diagnosability level obtained is discussed and the execution time results of our experimental evaluation of the three objectives detailed in Section 6.3 are presented.

6.4.1 Implementation

The Test-Point Allocator tool is implemented, which carries out the allocation of test points in business processes and provides options in accordance with the three objectives detailed in this chapter. The tool takes BPMN 2.0 models (OMG, 2011) as input and translates them into CSPs in accordance with the process presented in this chapter. In order to solve these CSPs, the tool uses the ILOG JSolver^{TM}

(IBM, 2003). The results are visualized on the screen using the Graphviz/dot library (Graphviz/Dot, 2011).


QUIVIR Research Group

Test-Point Allocator for Business Processes.

This is an application to perform the allocation of test-points in business processes in order to improve their diagnosability. The application can be configured to solve the problem in accordance with one of three objectives, which depend on the needs of the user or on the requirements of the problem:

- Cost limitations
- Time limitation
- Diagnosability-level limitations

Depending on the objective selected, the corresponding algorithm for the allocation of test points is executed, and the test points are placed in the business processes.

The extension of the source file must be .bpmn. In order to make easier the test of this application, it is also possible to use .dot files generated through the Process Log Generator tool (PLG 1.2, available at http://www.processmining.it/sw/plg).
Source file: Seleccionar archivo No se ha... archivo

○ Objective 1: Cost Limitations (maximization of the number of clusters with the allocation of a fixed number of test points)

○ Objective 2: Time limitations (allocation of the minimum number of test points in order to obtain a fixed number of clusters)

○ Objective 3: Diagnosability-level limitations (minimization of the number of test points to be allocated in order to obtain clusters with a maximum number of activities)

QUIVIR Research Group

Test-Point Allocator for Business Processes.

The blue flows are the ones monitored by test points and are marked with the label TP.

Figure 6.10: Screenshots of Test-Point Allocator

Figure 6.10 shows screenshots of the Test-Point Allocator tool: the initial form to upload the input file and choose the options and objective; and the result obtained for one of the tested examples for the process in Figure 6.1. The various colours of the activities indicate the cluster where they are located.

6.4.2 Experimental Design

Further to the diagnosability-level analysis, another purpose of the experimental evaluation is to determine the execution time from start to completion of a test-point allocation process.
With the aim of performing the execution-time measurements, our algorithms are executed over a set of randomly generated test cases obtained by using the *Process*

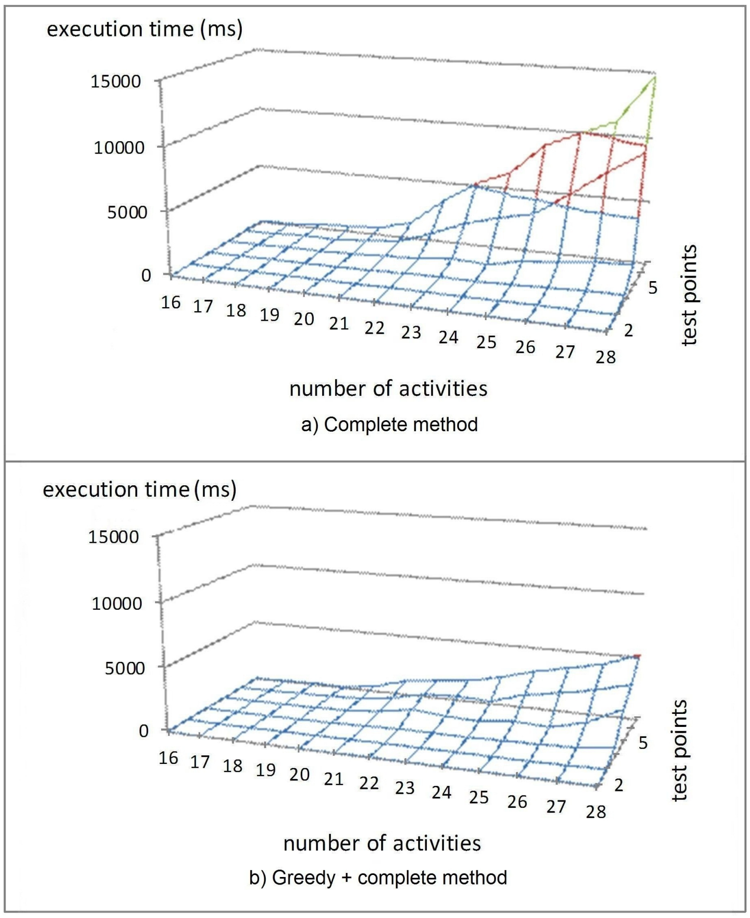


Figure 6.11: Execution time for Objective 1

Log Generator (PLG, (Burattin and Sperduti, 2010)). This mechanism enables the generation of realistic business processes in accordance with certain specific user-defined parameters.

In order to take the measurements, the test cases are composed of business processes, each with a different number of activities and flows, and several topologies are used.

The test cases are measured using a PC with CPU Intel Xeon 2,4GHz - 8GB RAM.

6.4.3 Performance Results

The worst-case complexity for the solution of COPs is high. Solving COPs takes exponential time due to the dependency of their complexity on the number of values each variable can take (Dechter, 1992; Traxler, 2008). It is therefore logical to empirically evaluate the performance of the developed methods. In order to assess whether, in practise, the time it takes to allocate test points is acceptable. Hence, in this subsection, the execution times for Objectives 1 and 3 are measured, where the execution time of the complete and greedy methods are compared.

Figure 6.11 a) shows the execution time necessary for the complete algorithm to reach Objective 1. This execution time can be compared with the performance measurement shown in Figure 6.11 b), which corresponds to the execution using the greedy method. It is possible to notice a minor increase in the execution time when there is a greater number of activities in the business processes. On compar-

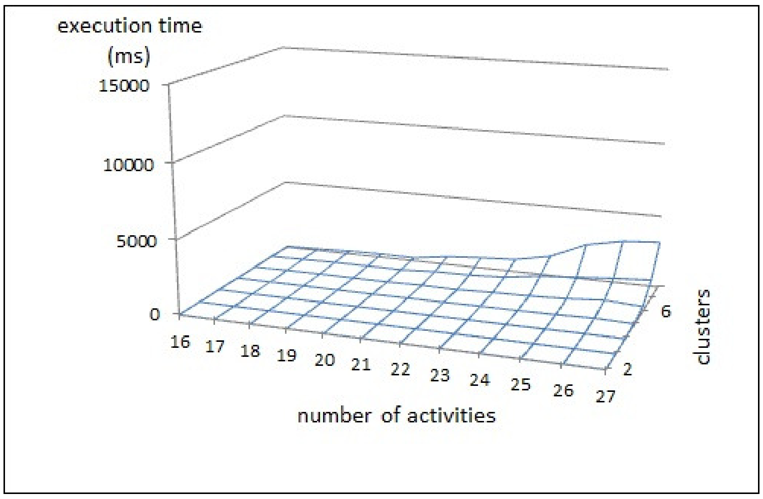


Figure 6.12: Execution time for Objective 2

ing the results, it is observed that the execution of the complete CSP algorithm, without the bound provided by the greedy method, takes three times as long as it would with this bound.

Likewise, Figure 6.12 shows the performance measurement for Objective 2.

And finally, the difference between the execution time for the complete method and that for the greedy method in the solution for Objective 3 can be observed in Figure 6.13. It is possible to notice the difference between the increase in the execution time for the complete method and the temporal complexity when the greedy algorithm is used for the establishment of a bound in the number of test points.

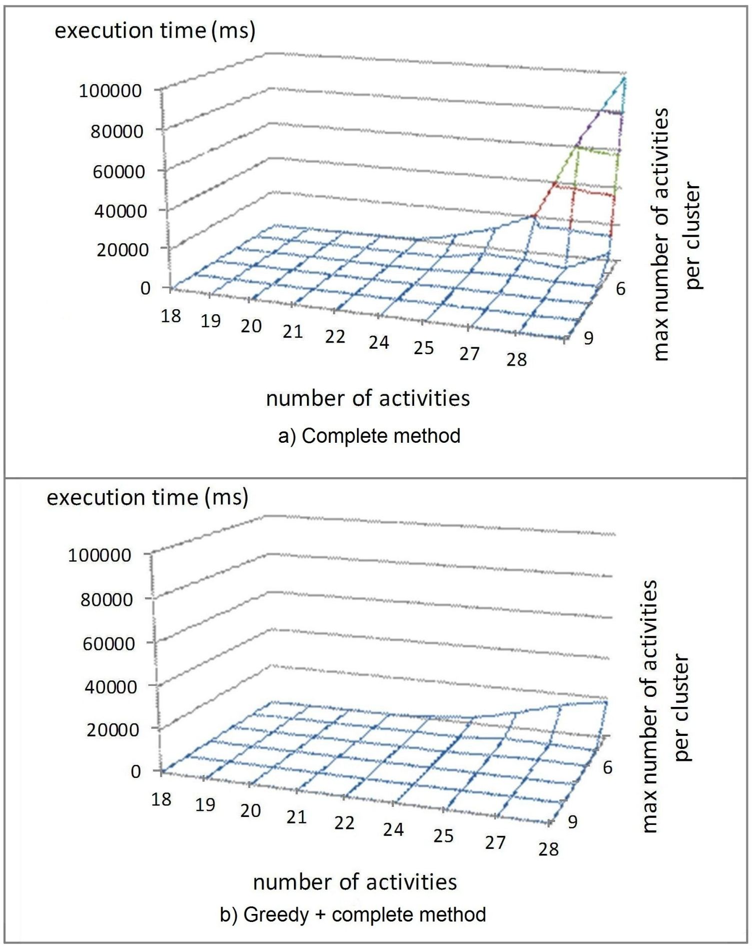


Figure 6.13: Execution time for Objective 3

6.5 Summary

This chapter provides a technique to improve the diagnosability of business processes. This is performed by the allocation of test points to guarantee the observability of the data flow at certain parts of the processes, and getting a division of the original business process into several clusters of activities, facilitating the later isolation of possible faults.
To this end, the business processes are modelled by using the Constraint Programming paradigm, which enables to fully automate the allocation of test points in an efficient manner.
The allocation of test points is carried out in accordance with three objectives which have been selected in terms of the most common time and cost requirements on behalf of the users, considering both the enhancement of the diagnosability of the business processes and the improvement of the computational complexity of the allocation and the later diagnosis process.
As a result, the Test-Point Allocator tool, available on the Internet, is able to allocate test points in business processes modelled in BPMN 2.0, providing options to select and configure the objective to achieve.

Part V

Contributions III and IV: Fault Diagnosis of Business Processes at Runtime

Chapter 7

Diagnosis of Business Processes based on Business Data Constraints

7.1 Introduction

In accordance with the ideas previously stated in the introduction of the current Thesis Dissertation, although the business process model used to configure and enact a business process is verified correct, the process may not behave correctly at run-time since the activities cannot be working as expected. The faults detected during or after the execution of process instances are derived from the goals which have not been reached due to some activity or activities which are not behaving correctly. Then, it becomes necessary to diagnose the activities by analysing process instances in order to determine which activity is not working as it was modelled.

In the case of the data flow perspective of the business process model available, including the description of the company policies as business data constraints in a contract, this contract can be used to automatically diagnose the behaviour of the activities at run-time. To this end, this chapter presents a contribution which proposes an automatic diagnosis method for this kind of malfunction, providing the functionality of the task *Quantitative Diagnosis based on Business Compliance Rules* of the diagram presented in Figure 1.2.

The proposed method is inspired in the classic point of view of model-based diagnosis that compares the expected behaviour (model) with the observations (model-based diagnosis). The discrepancies detected will be used in the diagnosis process. Not all the activities in a business process participate in every single exe-

cution, due to there are gateways that enable the execution of several branches for a varied number of times, since it is a runtime process only the activities involved in the executed instance should be considered for the diagnosis.
This contribution provides two automatic solutions which consider the trade-off between the obtaining of the minimal diagnosis and the performance. Both solutions find out the minimal diagnosis, with some differences: (1) the first solution, based on te use of reified constraints and Max-CSPs, only provides one solution to the diagnosis problem, and is better in terms of performance when it comes to find single faults; and (2) the second solution, based on the obtaining of Minimal Unsatisfiable Subsets, provides all solutions to the diagnosis problem, reaching first the solutions with multiple faults.

7.1.1 Motivation

The data flow perspective of a business process model can be described by means of business compliance rules, which define the company behaviour and policies, so that they work as the contract or semantic behaviour of business processes. As it was aforementioned in the background to business processes is Chapter 2, a subset of the business compliance rules are the business data constraints, which represent the semantic relation between the data values that are introduced, read and modified during the business process instances.
Since each organization defines its own policies about things such as prices, costs, numbers of employee, deadlines of tasks and so on, it is possible to specify a different set of business data constraints for each business process, even for different activities within the same business process.
The organizations must comply with an increasing number of internal and external regulations that are generally assured through regular audits and reviews. Due to continuously changing market conditions, an automatic approach to compliance management is desirable. There are some mechanisms to check whether business processes are compliant with the rules, but the key point is how we can specify the business processes model including business data constraints (i.e. relations between data, and values) in an easy way.
It is considered that the business process contract consists of the following:

- The data model that includes the business data constraints for these data;
- A business process model which, in this case, is a BPMN 2.0 diagram (OMG, 2011).

After a business process is modelled and verified, it gets enacted. During this phase of the lifecycle, the activities executed in each process instance transform the data of the business objects by introducing, reading or writing values, thereby being necessary to check whether the compliance rules are satisfiable after that data transformation. In the negative case, an abnormal behaviour is detected regarding the non-compliance of the rules defined in the model. Therefore, it becomes necessary to precisely locate the activity or activities which are not working as they were modelled. They are responsible for a malfunction and should be identified.

7.1.2 Contribution

The current chapter presents a solution based on the description of the activities' behaviour with business data constraints as contracts. Extensive literature research regarding business process compliance has been presented (Sadiq et al., 2007; Namiri and Stojanovic, 2007; Ghose and Koliadis, 2007). However, the main question arising in this contribution is whether it is possible to infer when or where an error has occurred with respect to the business policies and procedures in terms of the data-flow values. This suggests an adaptation of the typical fault model-based diagnosis approach to business process diagnosis, which is what is pursued in this chapter.

Business compliance rules contain business knowledge that describes the policies and procedures related to business processes. Business rules represent a natural step in the application of computer technology aimed at enhancing productivity in the workplace. The adoption of business rules adds another tier to systems that automate business processes. Compared to traditional systems, this approach has the following major advantages, as analysed in a greater depth by Weber et al. (2009): it lowers the cost incurred in the modification of business logic; it shortens development time; rules are externalized and easily shared among multiple applications; changes can be made faster and with less risk.

This contribution focuses on the use of business data constraints to describe the data semantics of a business process for the representation of the relations between data values. These business data constraints can be used for describing both the behaviour of each activity of the process independently (pre and postconditions) and the behaviour of the overall process in a global way. Both kind of rules can be associated to determined activities in terms of the data they manage. Therefore, one of our objectives is to establish the possible transformations of real-world concepts to constraints and to automate the process of diagnosis, representing as constraints the business data constraints associated to the different activities.

These constraints are linear or polynomial equations or inequations over data-flow variables, related by a boolean combination (*and/or*), according to the grammar defined in Definition 4.7.

According to one of the business rules taxonomies, current business rules are classified into integrity rules, derivation rules, reaction rules, production rules and transformation rules. This contribution focuses on the production rules, whose representation is the *if Condition then Consequence* format, and are used to validate the contract of the business process, where *Condition* represents the constraint to validate and *Consequence* the evaluation of the compliance rule. By using constraints, the information is represented at a more abstract level, since languages based on constraints include and improve all the capacity of representation of current rules engines, such us Drools, Fair Isaac Blaze Advisor, ILOG JRules and Jess. Likewise, representing the business data constraints as constraints we can get the automation of the contract-based diagnosis process.

By using business data constraints and Artificial Intelligence techniques oriented towards diagnosis, it is possible to describe the contract of the process to validate the correctness and determine any incorrect activities. In the business process area, the diagnosis process has two conditions that render the task of diagnosis more complicated: (a) Depending on the process instance and the input data, certain activities will work towards obtaining the objective of the process. This implies that in the diagnosis process, it is necessary to know which activities have participated. (b) A branch loop is a possible scenario, where a set of activities are executed several times, thereby the different executions of the loop have to be taken into account. Both strategies proposed in this chapter, based on Max-CSPs and MUSes respectively, take into account these depicted problems when it comes to perform the model-based fault diagnosis of incorrect activities.

The remainder of the chapter is organized as follows. Next subsection presents the specification of an illustrative example of business process and business data constraints that is used all along the chapter to describe the diagnosis process. Section 7.2 gives an overview to the fault diagnosis method proposed, based on business data constraints. Section 7.3 contains the main part of the contribution, describing the two proposed strategies, and including an improved algorithm based on minimal unsatisfiable subsets to find any incorrect activities by taking into account the executed activities and the possible loops. Section 7.4 presents the experimental results, and Section 7.5 concludes the chapter by summarizing the main concepts.

7.1.3 Illustrative Example

In order to illustrate the model-based fault diagnosis method presented in this chapter, the example about the handling of a conference, already detailed in Chapter 5, is used (shown again in Figure 7.1). The example is supposed to be already verified may and must-correct. However, some faults may be detected after its enactment and execution. Besides the data already mentioned as managed by this process, certain new variables are needed, all of them collected in Table 7.1

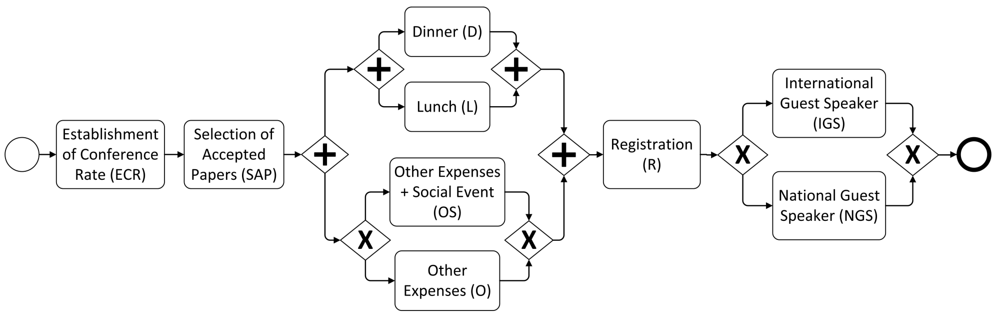

Dinner (D)
Lunch (L)
Establishment of Conference Rate (ECR)
Selection of Accepted Papers (SAP)
Other Expenses + Social Event (OS)
Other Expenses (O)
Registration (R)
International Guest Speaker (IGS)
National Guest Speaker (NGS)

Figure 7.1: Example of the business process for the organization of a conference

Table 7.1: Data input domain for the example in Figure 7.1

Variable	Domain	Meaning
totalCost	{25000..70000}	Total cost of the overall conference
totalExpenses	{25000..70000}	Total expenses of the overall conference
costPerAtt	{120..375}	Cost per conference attendee
regFee	{200..390}	Conference registration fee
sponsorship	{0..15000}	External contributions to support the event
numPapers	{50..80}	Number of accepted papers
dinner	{60..100}	Gala dinner cost
lunch	{10..30}	Cost of each lunch
others	{30..185}	Money for other expenses, like social events
confAtt	{75..170}	Number of conference attendees
guestSpeaker	{0..10000}	Money to spend in inviting a guest speaker

The semantics of the business process in the example is modelled as a contract whose business data constraints are detailed in Table 7.2 below.

Table 7.2: Business data constraints for the example in Figure 7.1

Business data constraints
$totalCost \leq totalExpenses$
$totalCost \geq totalExpenses * 0.8$
$totalCost = confAtt * costPerAtt + guestSpeaker$
$totalExpenses = confAtt * regFee + sponsorship$
$regFee * 0.1 \leq dinner$
$dinner \leq regFee * 0.35$
$regFee * 0.1 \leq 3 * lunch$
$3 * lunch \leq regFee * 0.35$
$confAtt \leq 75 \Rightarrow others \leq 0.2 * regFee + 0.05 * sponsorship$
$confAtt \leq 75 \Rightarrow others \geq 0.05 * regFee + 0.05 * sponsorship$
$confAtt > 75 \Rightarrow others \leq 0.25 * regFee$
$confAtt > 75 \Rightarrow others \geq 0.05 * regFee$
$costPerAtt = 3 * lunch + dinner + others$
$numPapers * 1.8 \geq confAtt$
$numPapers * 0.5 \leq confAtt$
$sponsorship \leq 2000 \Rightarrow guestSpeaker \geq 0.2 * sponsorship$
$sponsorship \leq 2000 \Rightarrow guestSpeaker \leq sponsorship + 0.1 * regFee * confAtt$
$sponsorship > 2000 \Rightarrow guestSpeaker \geq 0.4 * sponsorship$
$sponsorship > 2000 \Rightarrow guestSpeaker \leq sponsorship$

During the handling of the conference, some activities of the business process are executed, but it is possible that some of them are working incorrectly according to the main goals of the conference committee, for instance ”Do not spend more money than obtained with registrations and sponsorships; and a high participant satisfaction degree with the conference organization”.

It is possible that, after an enactment of the process, and due to the values assigned to some variables during the execution of certain activities, the business data constraints are not satisfied. For each instance, a diagnosis process must be performed to find out which activity or activities are not working as it was modelled and therefore expected.

7.2 Contract-based Diagnosis of Business Processes

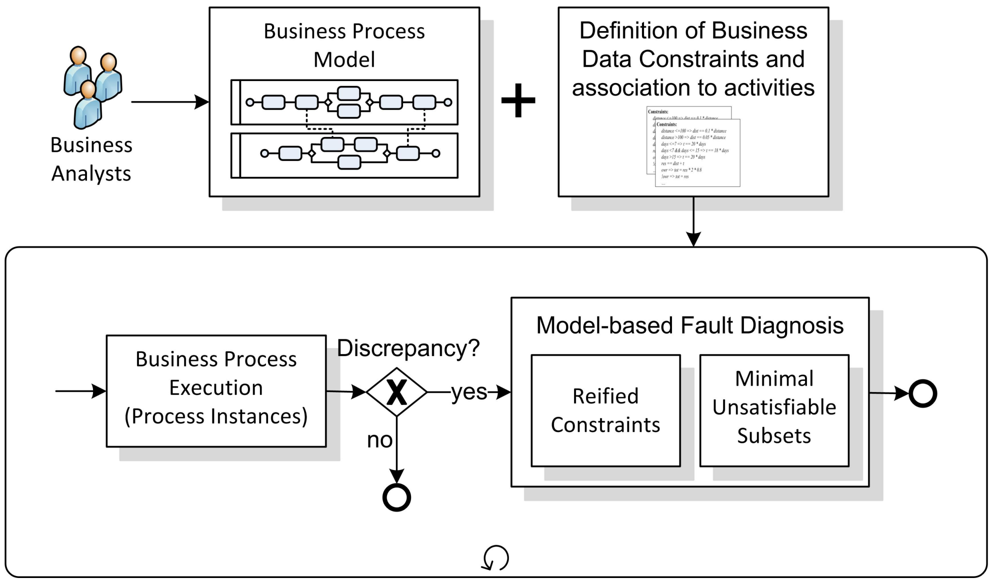

Business Analysts
Business Process Model
Definition of Business Data Constraints and association to activities
Business Process Execution (Process Instances)
Discrepancy?
yes
no
Model-based Fault Diagnosis
Reified Constraints
Minimal Unsatisfiable Subsets

Figure 7.2: Contract-based diagnosis process

In classic model-based diagnosis, in order to determine which component is working incorrectly, it is necessary to compare the model of the system with the observable variables. In the business process area, details of the model may be unavailable due to company privacy, although the policy of the companies and what business rules should be satisfied by the data flow must be known. This allows the user to diagnose possible faulty activities and to understand why a particular instance of the business process is not working. However, determining the root cause of a failure is an even greater challenge, since the topology of the business process and the involved activities for each instance must all be taken into account. Along these lines, we propose a diagnosis process based on the analysis of business output data and business data constraints to diagnose a business process instance. The model of the business process and the diagnosis is based on these rules represented by constraints, referred to as business data constraints.

Figure 7.2 gives an overview of the proposed diagnosis method. The process starts from the business process model together with the business data constraints (and their associations with the activities) defined by analysts, and performs the diagnosis process every time a discrepancy is detected after the execution of a process instance.

The business data constraints are checked against the logical states that can be traversed by the process. A naive way of checking compliance is hence to enumerate all those states. Clearly, given that the number of states is (in general) exponential to the size of the process, such an approach is undesirable.
When a business process model using business data constraints is enacted, some inconsistencies can be identified. In order to determine these inconsistencies, three types of information about the business process instance are necessary:

- Business data constraints that should be satisfied during the execution of the business process instance, and the activities associated to them.
- The activities that have participated in the process instance, since the various conditions associated to the control flow operators can determine separate execution paths.
- The values of the variables of the data flow that participate in the business data constraints. The variables of the data flow are those which can be instantiated during the execution of the process.

With this information, the diagnosis process is composed of three steps:

i. Selection of the business data constraints which are related to the process instance in accordance with the log file (Compliance Constraint/Activity Mapping).

ii. Detection of whether the business process has worked correctly during this instance.

iii. If an inconsistency is detected, the diagnosis process is used to determine the activity or activities which are responsible for the non-compliance. This step is carried out by building a Constraint Satisfaction Problem (CSP) with the involved business data constraints and the data-flow instance.

In order to perform the aforementioned determination of inconsistencies and the later diagnosis, the business data constraints and the data flow are mapped into a CSP. This way, the diagnosis task can be performed by using two proposals of diagnosis techniques, as explained in Section 7.3.

7.3 Diagnosis Process by using Business Data Constraints

The previous section presents the proposed diagnosis method composed of several steps. The diagnosis method to attain the objective of this contribution, deploys two strategies in order to provide two different diagnosis solutions, both of which are based on the Constraint Programming paradigm, since it enables to model the business data constraints as constraints in accordance with the grammar in Definition 4.7.

Due to the different control flow patterns that form the structure of a business process, it is necessary to take the instance that has been executed into account in order to determine the constraints that will compose the final CSP to be solved, in the same way as for the input data introduced during the execution of the process. That is, the information about the input data and the activities that have been executed are of interest since they define which business data constraints are related to that instance and whether these constraints are satisfiable for the data-flow values in that instance of the process.

Therefore, the diagnosis module in Figure 7.2 receives the activities that have been executed in order to build the correct CSP. The kinds of control flow patterns that influence that execution trace are:

- **Parallel Split (AND).** Since all branches are executed, if the execution of the business process reaches this structure, then all the activities, and consequently all their related business data constraints, should be taken into account in the diagnosis process.

- **Multi-choice (OR/XOR).** The CSP to solve should be composed of only the business data constraints associated to the activities in the executed branches.

- **Loop.** This structure enables a group of activities of the business process to be executed at least once depending on a logical condition. With regard to the incorporation of business data constraints into our CSP, and depending on the kind of tasks performed by the activities within the loop, two different cases can come up: (1) when only the most recent execution of the activities in the loop should be considered, since previous executions only performed tasks that are later cancelled or deleted by the final execution; (2) when all the executions of the loop must be taken into account in the building of the

CSP, since all the executions of the activities perform relevant actions with the input data.

In the first case, the business data constraints and observed data incorporated to the CSP are those related to the activities executed in the last iteration of the loop.

In the second case, the constraints derived from the business data constraints of the activities in the loop must be repeated in the CSP as many times as they are executed. In order to differentiate between the input data introduced in each iteration of the loop, the variables instantiated though the input data are renamed so that they have different names for each iteration of the loop, both as variables in the CSP and as variables in the different repeated constraints.

Once the constraints of the CSP are determined, based on the execution trace of the business process, then the diagnosis can be performed with either of these two proposed solutions:

- **Diagnosis using reified constraints and Max-CSP.** This proposal takes the relation between each business data constraints and one or more activities of the business process into account. Each business data constraint is transformed into a reified constraint, giving rise to an overconstrained *Constraint Satisfaction Problem*. Due to not existing any solution, there is an error. Using the reified constraints to build a Max-CSP, we try to minimize the number of unsatisfied constraints, finding the minimal sets of faulty activities. It performs a top-down strategy that provides better results when the diagnosis results are single faults.

- **Diagnosis based on the attainment of the MUSes.** The idea of this proposal is to find the Minimal Unsatisfiable Subsets of constraints (*MUSes*, (de la Banda et al., 2003)) in order to find what is wrong in an overconstrained CSP instance. By finding these subsets, and calculating their minimal hitting sets, the activities responsible for rendering the business data constraints of the business process inconsistent can be identified (Reiter, 1987). This proposal obtains the minimal diagnosis of the activities by means of a bottom-up strategy that first reaches the solutions with multiple faults.

Both proposals for diagnosis are given in greater detail in following subsections.

7.3.1 Contract-based Diagnosis Using Reified Constraints and Max-CSP

As mentioned above, each business data constraints is associated to the activities related to the participating variables. This association is performed in an automatic way, since it is only necessary to determine the variables related to each activity in order to be able to relate business data constraints and activities.

This proposal builds reified constraints based on each business data constraints. A reified constraint consists of a constraint and a variable which denotes its truth value. This way, starting from a CSP composed of the business data constraints, it is necessary to add new variables to this CSP. They are boolean variables, one for each business data constraints, which are used to build reified constraints, associating each constraint to a boolean through an equality.

Applying this idea to the example in Figure 7.1 gives rise to 19 new boolean variables, one for each constraint ($C1, C2, C3, \ldots$). The compliance constraints are related to these boolean variables, forming reified constraints as, for example:

$$C1 == (totalCost <= totalExpenses)$$

$$C2 == (totalCost >= totalExpenses * 0.8)$$

Then, the objective function is added to the Max-CSP to be solved. Its aim is to maximize the number of activities which are not responsible for the unsatisfactibility of the CSP. That is, the objective function maximizes the number of boolean variables instantiated to *true*.

Once this maximization is performed, it is necessary to know the activities responsible for the inconsistency. To this end, we take the sets of activities related to the constraints whose associated boolean variable has been instantiated to false, and find out the minimal collection of activities which cover all those sets. According to diagnosis theory, the best way to find that activity or activities is by calculating the hitting sets and minimal hitting sets, which provide us with the minimal diagnosis for a business process instance.

As an example, for the reified constraints of the example in Section 7.1.3, if the organizing committee of the conference receives less money that expected from the sponsors, and the CSP is solved, the activities instantiated to false are *ECR* and *IGS*. It means that the lack of money must be solved increasing the conference rate, or decreasing the expenses of the invitation of the international guest speaker.

7.3.2 Determination of MUSes for Contract-based Diagnosis

This solution is based on the attainment of all the *MUSes*, which represent the most succinct explanations, in terms of the number of involved constraints, of infeasibility. Indeed, when we check the consistency of a CSP, it is preferable to know which constraints are contradicting one another rather than only knowing that the CSP as a whole is inconsistent.
Due to the computational complexity of the attainment of all *MUSes*, some existing approaches were designed to derive only some of the *MUSes* and not all of the *MUSes* of an overconstrained CSP. Our proposal is based on an improvement in Gasca et al. (2007), which is grounded in structural analysis in order to determine all the *MUSes* efficiently. In that paper, several techniques are presented, which improve the complete technique in several ways depending on the structure of the constraint network. These techniques make use of the powerful concept of the structural lattice of the constraints and neighbourhood-based structural analysis to boost the efficiency of the exhaustive algorithms. Since systematic methods for solving hard combinatorial problems are too computational expensive, structural analysis offers an alternative approach for fast generation of all *MUSes*. Accordingly, experimental studies of these new techniques outperform the best exhaustive techniques since they avoid the necessity of solving a high number of CSPs with exponential complexity. However they do add certain new procedures with polynomial complexity.
In the case of this contribution, two techniques are applied so that their computational complexity can be compared:

- **Exhaustive technique.** It attains the *MUSes* in accordance with the process described in Algorithm 7.3. It uses th queue Q (line 3) to initially store the inconsistent constraints. The process uses this queue to exhaustively check the satisfactibility of every possible combination of constraints. In the case a subset of constraints is proved unsatisfiable (checked in line 10), it is stored in the list MUS (line 13).

- **Variable-Based Neighbourhood technique.** It uses the knowledge about the Variable-Based Neighbourhood explained in Gasca et al. (2007). To this end, this technique relates business data constraints as neighbours if they share certain *Non-Observable Variables*, which are those variables whose values remain unknown until the end of the execution of the process since they are not determined by the role of the pool.

```
1:  procedure ObtainMUSes1($Constraints\ C$)
2:      //Being $C$ the business data constraints that present the inconsistency:
3:      $Q :=$ Queue initialized with each constraint in $C$
4:      $MUS :=$ List that will contain the $MUSes$, initialized to empty
5:      while $Q$ is not empty do
6:          $\{c_i \dots c_j\} := Q.poll()$ //retry and remove the head of $Q$
7:          for $c_k \in \{c_{j+1} \dots c_n\}$ do
8:              if NOT $\exists SubSet^{1 \dots n-1}_{\{c_i \dots c_j\}} \cup c_k \in MUS$ then
9:                  // being $n$ the cardinality of $\{c_i \dots c_j\}$
10:                 if $\{c_i \dots c_j\} \cup c_k$ is a satisfiable CSP then
11:                     $Q.add(\{c_i \dots c_j\} \cup c_k)$
12:                 else
13:                     $MUS.add(\{c_i \dots c_j\} \cup c_k)$
14:                 end if
15:             end if
16:         end for
17:     end while
18: end procedure
```

Figure 7.3: Algorithm to obtain the *MUSes* (I)

Algorithm 7.4 shows the details of this process, with is similar to the process described in Algorithm 7.3, but with the difference of the generation of neighbours to propagate the search of MUSes (line 8).

As an improvement upon the technique presented in Gasca et al. (2007) when it comes to calculate the neighbours of a set of business data constraints, not all the neighbours are generated: an order is established between the constraints, so that only those neighbours that are subsequent to all the constraints in the set are generated. In this way we succeed in generating only *new* neighbours, thereby avoiding the repeated study of the satisfiability of the same collection of constraints.

Once all *MUSes* have been obtained, each *MUS* is then associated to the activities related to the constraints that it contains. That is, for every *MUS* obtained, we count on a set of activities related to that *MUS*, in such a way that the activity or activities responsible for the unexpected behaviour is contained in those groups. As in the technique based on reified constraints, the minimal hitting sets are used to find out that activities. Again, the calculation of these minimal hitting sets of activities provides us with the minimal diagnosis for a business process instance.

```
1: procedure ObtainMUSes2(Constraints C)
2:     //Being C the business data constraints that present the inconsistency:
3:     $Q$ := Queue initialized with each constraint in $C$.
4:     //The data structure selected determines the search strategy.
5:     $MUS$ := List that will contain the MUSes, initialized to empty
6:     while $Q$ is not empty do
7:         $\{c_i \dots c_j\} := Q.poll()$ //choose an element of $Q$
8:         $neighbours := expand(\{c_i \dots c_j\})$
9:         //generate neighbours according to the Variable-Based Neighbourhood
10:        for all $c_k \in neighbours$ do
11:            if $\{c_i \dots c_j\} \cup c_k$ is a satisfiable CSP then
12:                $Q.add(\{c_i \dots c_j\} \cup c_k)$
13:            else
14:                $MUS.add(\{c_i \dots c_j\} \cup c_k)$
15:            end if
16:        end for
17:    end while
18: end procedure
```

Figure 7.4: Algorithm to obtain the *MUSes* (II)

7.4 Empirical Evaluation

As stated in previous chapter, due to the exponential complexity of solving the worst-case of a CSP, this section shows the empirical evaluation of the performance of the proposed solutions to asses that the execution time is acceptable in practise.

7.4.1 Experimental Design

In order to perform the empirical evaluation, certain tests have been performed using different test cases of business processes with different number of business data constraints.
Those tests are developed from satisfiable solutions of the obtained CSPs, and changing some input data in order to cause inconsistencies. For instance, for the example in Figure 7.1, the inconsistencies generated are:

- Test case 1: Establishing a very low quantity of accepted papers (*numPapers* = 10).

- Test case 2: With only 60 participants (*confAtt* $= 60$).
- Test case 3: Not receiving much money from the sponsors, and spending too much money in inviting an international speaker (*sponsorship* $= 2001$, *guestSpeaker* $= 4250$).

The test cases are measured using a Windows 7 machine, with an Intel Core 2 Duo processor, 1.86GHz and 2.0Gb RAM.

7.4.2 Performance Results

In this subsection, the results obtained in both proposals are presented. To this end, and due to the temporal complexity of the diagnosis using reified constraints is negligible (always lower than 1 millisecond), only the temporal complexity of both algorithms for obtaining the *MUSes* in Subsection 7.3.2 are compared. Moreover, the minimal diagnosis results obtained by the two techniques are presented.

Both techniques to calculate the *MUSes* obtain the same diagnosis results, and do not present any false positive. Therefore, Table 7.3 presents only the computational complexity comparison. The table shows the average time spent by each technique for each test case and the iterations performed (that is, the number of subsets whose satisfactibility is checked). It is possible to notice that the exhaustive technique spends rather more time than the Variable-Based Neighbourhood technique. The reason for this is that the exhaustive technique tries to find the *MUSes* by checking all the possible subsets of constraints, whereas the Variable-Based Neighbourhood technique is confined to the subsets formed by neighbours.

Table 7.4 shows the different diagnoses obtained from the different techniques, using the different test cases detailed before for the example of business process shown in Figure 7.1.

As mentioned above, both techniques find out the minimal diagnosis. Regarding the differences in the temporal complexities, it is due to the search strategy followed by each technique. On the one hand, the technique based on reified constraints is better when it comes to find single faults. On the other hand, the technique based on the attainment of the *MUSes* performs a search tree that first reaches the solutions with multiple faults. Therefore, the technique to use must be selected depending on the kinds of faults (single or multiple) that use to appear in the process to diagnose.

Table 7.3: Comparison between MUSes algorithms

#business data constraints	Exhaustive technique		Variable-Based Neighbourhood technique	
	time (ms)	iter.	time (ms)	iter.
9	2,588,891	502	12,547	15
9	1,031,110	256	15,437	24
9	1,004,750	256	12,953	15
9	336,000	129	63,844	16
9	114,265	67	48,235	16
9	114,703	67	46,531	12
11	3,354,115	951	32,313	37
11	426,234	131	10,109	16
11	742,188	157	17,438	22
13	2,326,453	514	15,156	21
13	3,004,571	725	21,063	31

7.5 Summary

This chapter provides a method to diagnose business processes whose models are correct but present abnormal behaviour after their execution. These business processes count on a contract represented by constraints to describe the behaviour/semantics of their activities, being the main question arising this contribution whether it is possible to infer what or where a failure has occurred with respect to the business policies and procedures in terms of the data-flow values.

To this end, an adaptation of the typical fault model-based diagnosis approach to business process diagnosis have been developed in this chapter. The diagnosis process has been fully automated by representing as constraints the business data constraints of the processes to diagnose. Those constraints are defined following a grammar to avoid ambiguities.

Likewise, a diagnosis method has been proposed, which performs the automatic diagnosis after detecting the non-compliance of the rules at run-time. The diagnosis process has evidence about the data flow and activities executed in each business process instance, so that only those activities are considered during the

Table 7.4: Diagnosis results

test case	Minimal Diagnosis
1	Selection of Accepted Papers
2	Registration
3	International Guest Speaker

diagnosis process. The diagnosis process is able to both detect and diagnose the inconsistencies in the execution.
To deal with this diagnosis problem, two different strategies have been proposed: (1) use of reified constrains and Max-CSPs to relate each business data constraint to the set of activities whose execution affects to its satisfactibility; and (2) the attainment of minimal unsatisfiable subsets of business data constraints to search for the minimal set of activities which make the contract inconsistent. Both strategies obtains the minimal diagnosis of the processes in an efficient way.

Chapter 8

Diagnosis of Business Processes based on Structural Analysis

8.1 Introduction

As it was aforementioned in the introduction to the current Thesis Dissertation, it is not always possible to count on semantics to define the behaviour of a business process, since the data flow perspective and the behaviour represented by means of business data constraints are not always available, or it cannot be modelled as numeric data constraints to be automatically treated.

Although business processes are configured and enacted from a model that is verified correct, certain errors may occur at run-time during their execution, being desirable to determine which activity or activities are not working as they were modelled. This diagnosis is even more difficult when different processes interact and perform a business-to-business collaboration.

In order to determine the activities whose behaviour is incorrect, it would be necessary to resort to the structural analysis of the collaborations, since the functional dependencies between activities make possible to infer the activity or activities which are responsible for a malfunction. In order to perform this analysis, and since there is no data flow perspective of the model available to compare between the expected and observed values of the data, it is necessary to count on a signature matrix (coming from complaints of users on customers) with qualitative information indicating the achievement of the business goals.

Business-to-business collaborations are quite complex, and any failure might have an immediate effect on the operational business of the involved companies. To deal with this problem, this chapter presents a contribution to diagnose faulty activities in business-to-business collaborations, based on FDI techniques. The approach provides the functionality of the task *Qualitative Diagnosis based on Structural Analysis* on the diagram in Figure 1.2 by using only structural analysis of the choreographies when no other kind of semantics of the processes are available to perform any automatic diagnostic reasoning.

8.1.1 Motivation

When business processes which take part in a business-to-business collaboration are enacted, any failure in the overall collaboration could have important negative effects on the operational activities over the companies that are collaborating. In these cases, the customers may present complaints over the accuracy if their requests are not satisfied.
The fault management procedures in business-to-business collaboration require that the faults are diagnosed and localized as precisely and efficiently as possible to ensure proper corrective actions. Thereby, when the faults occur at run-time, it is even more desirable to automatically detect system malfunctions and diagnose in a timely and efficient manner to reduce the impact of the faults on the utility of business-to-business collaborations. In the types of systems proposed in this chapter, where the detection of incorrect behaviours is only based on the analysis of the information provided by the oracle, the main problem arises when the diagnosis has to be performed without a model to compare.
If the models of the processes in the collaboration are verified correct, the detected faults are necessarily caused by an activity or a set of activities which are not working as it was specified in its model. Moreover, when those models do not count on any data flow perspective due to the semantics is not available or it cannot be modelled as business data constraints, the diagnosis process may only be based on the control flow perspective of the business-to-business collaboration, and the data provided by the oracle.

8.1.2 Contribution

Since there is no semantic information available, or the one that is available cannot be modelled as a contract, and besides the global structural information of the overall choreography of a business-to-business collaboration is distributed, our contribution proposes to perform the diagnosis by using a set of local diagnosers,

each one associated to a business process in the collaboration and counting on partial knowledge of the overall model of the choreography, i.e., the part of the model which corresponds to the business process it is associated to.
Therefore, the diagnosis of this kind of collaborations should be necessarily performed via structural analysis of their topologies. This way, the dependencies between the business processes, their activities, and the attainment of the business goals are studied in order to determine the source of an abnormal behaviour of the processes.
The problem of finding all diagnoses (i.e. all faulty activities that explain an abnormal behaviour) is costly in terms of execution time. In order to improve the performance of the distributed diagnosis, the model of each business process is simplified by each diagnoser in a previously compiled model, attained off-line, getting a diagnosis method to isolate the activity or activities which are not working as they were modelled, in two phases:

- As the model does not change for the different instances, a preprocessing step can be done to discover the structural dependencies between the activities. Then, during the first phase, which is performed off-line, a precompiled model of each business process is built by each local diagnoser by analysing the structure of each process and the external collaborations with other processes (by means of messages).

- Then, if an abnormal behaviour is detected by receiving a complaint, the second phase is carried out on-line, where the diagnosis is performed over the pre-compiled model to determine and isolate the faulty activities whose incorrect behaviour explains the incorrect results of the overall collaboration.

An innovation of our contribution is that it does not require a complete computational treatment of all business process instances to know which activities are not working correctly, but it only uses the trace of the instance which finished with abnormal results.
To describe those diagnosis tasks mentioned above, the remainder of this chapter is organized as follows. Next subsection presents an example to illustrate the diagnosis process along the chapter. Section 8.2 defines concepts related to business-to-business collaborations and structural analysis. Section 8.3 details the methodology used in the diagnosis process, detailing its phases and algorithms. Section

8.4 gives implementation details about the proposed algorithms and shows experimental results. And finally, a summary of the contribution is outlined in Section 8.5.

8.1.3 Illustrative Example

In order to explain our contribution, the example adapted from Weske (2007), shown in Figure 8.1, is used. It depicts a business-to-business collaboration consisting of four business processes from different organizations, (*BP1, BP2, BP3* and *BP4*).

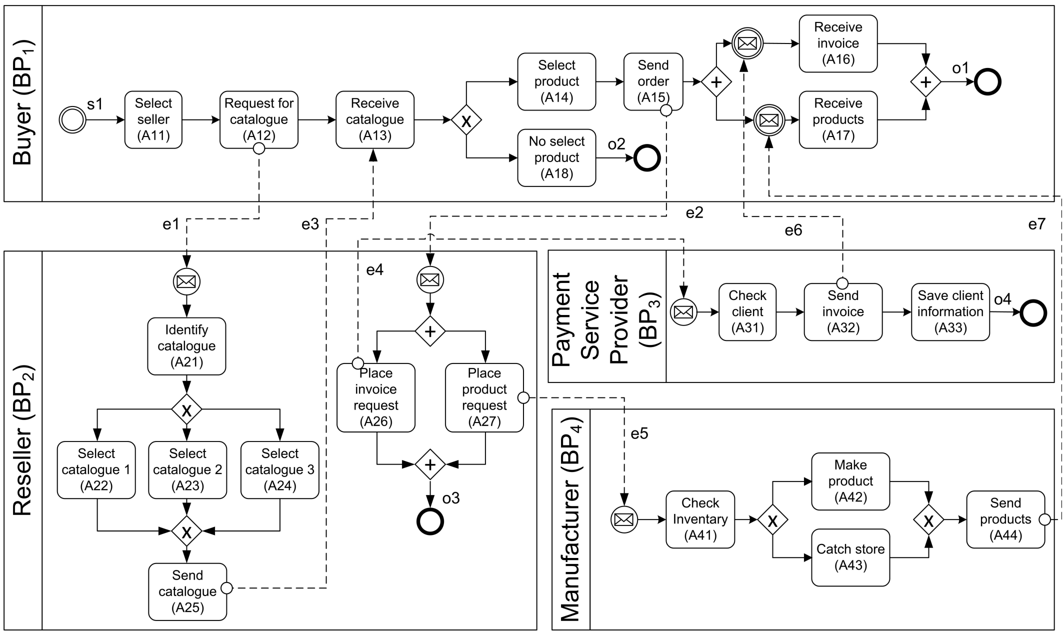


Figure 8.1: Example of Business-to-Business Collaboration

Although the models of the business processes in a collaboration are not available in a global way, in the example, the graphical representation of the four business processes are shown together to clarify the interactions between them. Those different business processes with their respective information are:

- *BP1* (Buyer), with the start event {*s1*} and {*e1, e2, e3, e6, e7*} as message flows (i.e. interactions with the other business processes).
- *BP2* (Reseller), with {*e1, e2, e3, e4, e5*} as message flows.
- *BP3* (Payment Service Provider), with {*e4, e6*} as message flows.

- *BP4* (Manufacturer), with $\{e5, e7\}$ as message flows.

As can be seen in Figure 8.1, the control flow perspective of the model of this business-to-business collaboration is available. However, the data flow perspective cannot be represented through numeric constraints (hence, through business data constraints), so that the only available information from this perspective comes from the oracle indicating if the outputs of the overall collaboration (represented by the end events $\{o1, o2, o3, o4\}$ in the example) are correct or incorrect (qualitative information).
Regarding the control flow perspective, the flows between the activities within each business process remain unobservable from the outside but are only known within the own process internally.
Each business process does not know how the rest business processes in the choreography are formed, only knows from and towards what business processes has to receive or send messages.

8.2 Main Concepts for the Structural Diagnosis of Business-to-Business Collaborations

In order to clarify the explanations along this chapter, we use the following notation in Table 8.1 with regard to the elements which participate in a business-to-business collaboration.

Table 8.1: Notation for the elements in a collaboration

Notation	Description
$BP_i, i \in \{1..n\}$	Every one of the *n* business processes in a collaboration
$\{s_1, \ldots, s_q\}$	Start events in the collaboration
$\{o_1, \ldots, o_r\}$	End events in the collaboration, which inform about the correctness of each output
$\{e_1, \ldots, e_p\}$	Message flows between different processes, called **external interactions**
$\{i_1, \ldots, i_m\}$	Sequence, conditional and default flows between activities within each business process, called **internal interactions**

It is considered that business-to-business collaboration design ensures that the participant business processes are compatible to interact successfully according to the overall collaboration. In our case, since the model is considered correct, this implies:

- No erroneous interaction between the business processes in the collaboration.
- The messages that can be sent by a participant business process correspond to messages that other business processes in the collaboration can receive (structural compatibility).
- The partner business processes involved in a collaboration need to agree on their interactions to avoid deadlock and livelock. Then, as was aforementioned, in this phase we suppose a correct model.

With these assumptions, and considering that there is no entity with a global view of the overall collaboration, the key idea of this contribution for diagnosing faults in business-to-business collaborations is to provide each business process with a local diagnoser. Those local diagnosers are in charge of the diagnosis performance for its associated business process, having knowledge of its orchestration model (its internal interactions) and counting on the Business Process Interaction Model (BPIM).

Definition 8.1. *Diagnoser$_i$: Diagnoser associated to the i-th business process in the collaboration (BP_i) that can perform local diagnosis tasks and send, receive and process information with other diagnosers.*

Definition 8.2. *Business Process Interaction Model (BPIM) of a business process BP_i in a business-to-business collaboration represents the external interactions of BP_i with the rest business processes in the collaboration. The BPIM is known by the Diagnoser$_i$.*

In order to get a more efficient diagnosis of faulty activities, each local diagnoser is able to attain a Compiled Orchestration Model (COM) of its associated business process in an off-line way. This way, the intermediate calculations of the diagnosis process can be performed only once for the overall collaboration, avoiding the repetition of these calculations for every instance. The COM is defined as follows.

Definition 8.3. *Compiled Orchestration Model (COM) of a business process BP_i in a business-to-business collaboration is a simplified representation of the control flow perspective of the model of BP_i.*

Our main contribution to diagnose business-to-business collaborations is based on this off-line COM obtained by using structural analysis. The COM is used to perform all the diagnosis tasks, getting a better computational efficiency. This equivalent model represents the relations among the activities of a business process and their external interactions, so that the diagnosis can be performed without any information about the internal interactions in a global way.

8.3 Logic and Distributed Algorithm

The diagnosis method proposed in this contribution is depicted in the diagram in Figure 8.2, where the steps of the process are detailed. First, after the business process analysts have modelled the business processes participating in the collaboration, and those models are verified correct, the *Diagnoser* associated to each business process performs the first phase, which is an off-line processing to attain the COMs, containing only one step called *(1) Determining the COMs*. This phase is performed only once, since after the COMs have been built, they are used in all subsequent diagnosis.
Then, the collaboration can be enacted and executed. If some abnormal behaviour is detected after the execution of a single process instance, the second phase of the diagnosis process is carried out in an on-line way, in order to propagate information and perform local diagnosis respectively, containing two steps called *(2) Propagation Phase* and *(3) Local Diagnosis Phase* in Figure 8.2.
As a result, the activities responsible for the abnormal behaviour of the collaboration are determined.
Therefore, the complete diagnosis process is composed of three steps, detailed in the following:

- **Off-line analysis**, performed only once from a verified correct model of the business-to-business collaboration.
  - **(1) Determining the COMs:** In each business process BP$_i$ involved in the process, an off-line orchestration analysis is executed by the *Diagnoser*$_i$. The result of this step is the COM for each business process, which is an equivalent model used in the next steps to improve the temporal efficiency of the diagnosis process.
- **On-line process**, performed if a discrepancy between the observed and expected results is detected and the users or customers complaint about it.

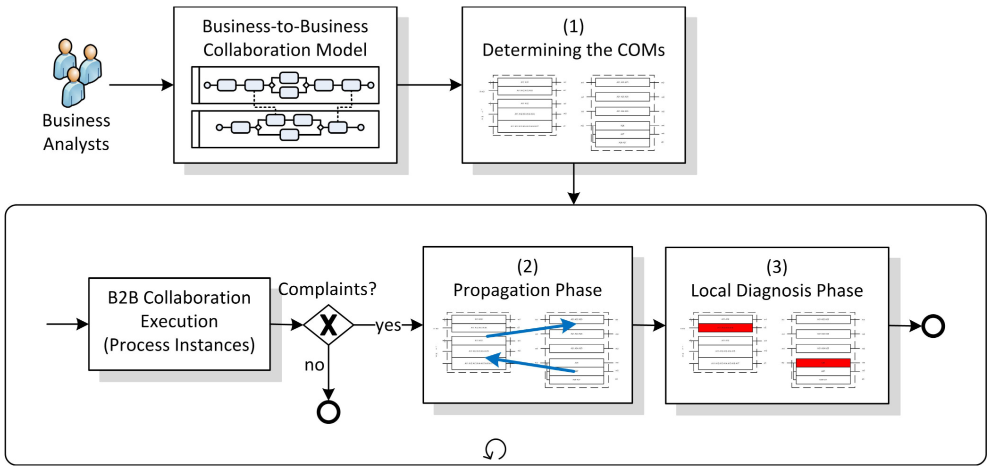

Business Analysts
Business-to-Business Collaboration Model
(1) Determining the COMs
B2B Collaboration Execution (Process Instances)
Complaints?
yes
no
(2) Propagation Phase
(3) Local Diagnosis Phase

Figure 8.2: Phases of the diagnosis process

– **(2) Propagation phase:** After the *Diagnoser$_i$* receives information about a discrepancy detected at some observable point (either at the end events or by some test point reading), an internal algorithm is run in order to decide which flows could be correct or incorrect. As a result of this algorithm, each *Diagnoser$_i$* sends outgoing information to the *Diagnosers* associated to the business processes which interact with BP$_i$ informing about the incorrect behaviour detected. This information traverses the *Diagnosers* in the opposite direction of the flow during a normal execution of the collaboration. That is, from end events to start events.

– **(3) Local diagnosis phase:** Depending on the received information, each *Diagnoser* has to decide if the local diagnosis of its associated business process is necessary.

These steps of the diagnosis process are detailed in the next subsections.

8.3.1 Determining the Compiled Orchestration Model

In order to obtain the COM of the business-to-business collaboration, a orchestration analysis takes place. Each *Diagnoser$_i$* must analyse the structure of the control flow of BP$_i$, in such a way that it has to determine the different paths (i.e. cluster of activities) that may be executed considering every possible process instances.

Thereby, within a business process, those different paths are determined by:

- Considering a business process as a graph, where the activities represent the nodes, and the different interactions between them the edges, the different connected components of the graph give rise to different paths.
- If, within a connected component of the graph, exists an XOR-split with n options, it gives rise to n paths, since n options of an XOR or OR imply n different possible execution paths for the workflow.

Once the paths of the business processes are calculated, the way to know if an activity is working correctly is by analysing its inputs, outputs or other activities related to it by external or internal interactions. Within each path, there exist internal subsets (*IS*). The idea of an *IS* is to collect a set of activities where the external and internal interactions between them relate more than one activity in the *IS*, to derive the diagnosis using orchestration analysis. In order to clarify this idea, new notations are used:

External(A_{ij}) are the external interactions which are inputs or outputs of the activity A_{ij}.

Internal(A_{ij}) are the internal interactions which are inputs or outputs of the activity A_{ij}.

Definition 8.4. *Internal Subset (IS) for a path in a business process* BP_i *is a set of activities, where for each one of its activities* A_{ij}:

$\forall v \mid v \in Input(A_{ij})$: $v \in \{s1, \ldots, sq\}$
$\vee$ ($v \in$ *Internal*(A_{ij}) $\wedge \exists A_{ik} \neq A_{ij} \mid A_{ik} \in IS \mid v \in$ *Output*(A_{ik}))
$\vee$ ($v \in$ *External*(A_{ij}) $\wedge \exists A_{ik} \neq A_{ij} \mid A_{ik} \in IS \mid v \in$ *Output*(A_{ik}))
$\vee$ ($v \in$ *External*(A_{ij}) $\wedge v \in$ *External*(A_{jk}) $\mid A_{jk} \in BP_j \neq BP_i$)

(i.e. the input v of A_{ij} *should be: (1) start event; (2) an internal ((3) external) interaction with another activity in the same IS; or (4) an external interaction with an activity from a different business process)*

$\forall v \mid v \in Output(A_{ij})$: $v \in \{o1, \ldots, or\}$
$\vee$ ($v \in$ *Internal*(A_{ij}) $\wedge \exists A_{ik} \neq A_{ij} \mid A_{ik} \in IS \mid v \in$ *Input*(A_{ik})) $\vee$ ($v \in$ *External*(A_{ij}))

(i.e. the output v of A_{ij} *should be: (1) end event; (2) internal interaction with another activity in the same IS; or (3) external interaction with any other activity)*

and for the set of activities that form each internal subset IS:

$\forall v \mid v \in Input(IS)$: $v \in Input(A_{ij}) \wedge A_{ij} \in IS \wedge (v \in \{s1, \ldots, sq\} \vee v \in External(A_{ij}))$

(i.e. the input of an IS is the input v of an activity A_{ij} in the IS, being v a start event or an external interaction of A_{ij})

$\forall v \mid v \in Output(IS)$: $v \in Output(A_{ij}) \wedge A_{ij} \in IS$
$\wedge (v \in \{o1, \ldots, or\} \vee (v \in External(A_{ij}) \wedge \nexists A_{ik} \neq A_{ij} \mid A_{ik} \in IS \wedge v \in External(A_{ik})))$

(i.e. the output of an IS is the output v of an activity A_{ij} in the IS, being v an end event or an external interactikon between A_{ij} and A_{ik}, also in this same IS)

For the example in Figure 8.2, the different paths calculated by the *Diagnosers* are:

- *Diagnoser*$_1$: $\{A11, A12, A13, A18\}$ and $\{A11, A12, A13, A14, A15, A16, A17\}$
- *Diagnoser*$_2$: $\{A21, A22, A25\}$, $\{A21, A23, A25\}$, $\{A21, A24, A25\}$ and $\{A26, A27\}$
- *Diagnoser*$_3$: $\{A31, A32, A33\}$
- *Diagnoser*$_4$: $\{A41, A42, A44\}$ and $\{A41, A43, A44\}$

As an example, the activities $\{A26, A27\}$ form a path with three internal subsets as it is shown in Figure 8.3. This path (like every path in a collaboration) works as a black box for the rest of *Diagnosers*, and its activities can be connected to other paths through external interactions.
In general, the paths and *IS*s calculated by the *Diagnosers* give rise to the COMs of the collaboration, as it is shown in Figure 8.4 for our example. This COM, customized for the diagnosis process, is built off-line, and is used to perform the next diagnosis phases in an efficient way.
Once the COM has been obtained and stored, the collaboration can be enacted and executed.

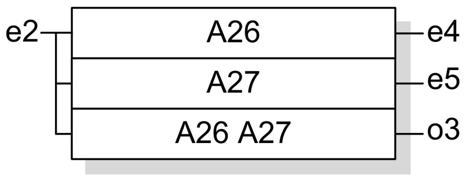


Figure 8.3: Graphical representation of a local path with three Internal Subsets

8.3.2 Propagation Phase

A business-to-business collaboration depends on the users or customers, who are able to determine the discrepancies between the observed and expected behaviour, detecting possible faults.
Once the collaboration has been enacted, if some discrepancies are detected after the execution of a process instance, each *Diagnoser* receives information about the end events of the collaboration where the discrepancies occurred (**incorrect** end events) and the end events where no anomaly were detected (**correct** end events). It must be taken into account that the presence of XOR-splits and OR-splits in the business processes causes that not all the end events have to be reached at every process instance, since an XOR implies the execution of only one path out of several branches, so that the end events located in no selected paths are not reached during that execution.
In the case all reached end events are correct, no propagation is necessary, since all the activities are working correctly. In the case at least one discrepancy has been detected, the propagation phase starts in order to determine which activities are responsible for the fault. To this end, each $Diagnoser_i$ knows the path that was executed in BP_i during the execution of the process instance. For example, BP_2 in the example has three different paths depending on the XOR gateway because it determines the activity that was executed (A22, A23 or A24) and hence it determines which path should be considered in the propagation phase.
In order to explain the next steps of the diagnosis process, these new definitions have to be introduced:

Definition 8.5. *Possible Incorrect Interaction (PII): It is an external interaction related to an incorrect end event of the collaboration. An interaction is related to an end event if when the interaction changes, the end event also changes.*

Definition 8.6. *Correct Interaction (CI): It is an external interaction related to a correct end event. If an interaction is related to a correct end event and an incorrect end event simultaneously, the interaction is defined as a Correct Interaction,*

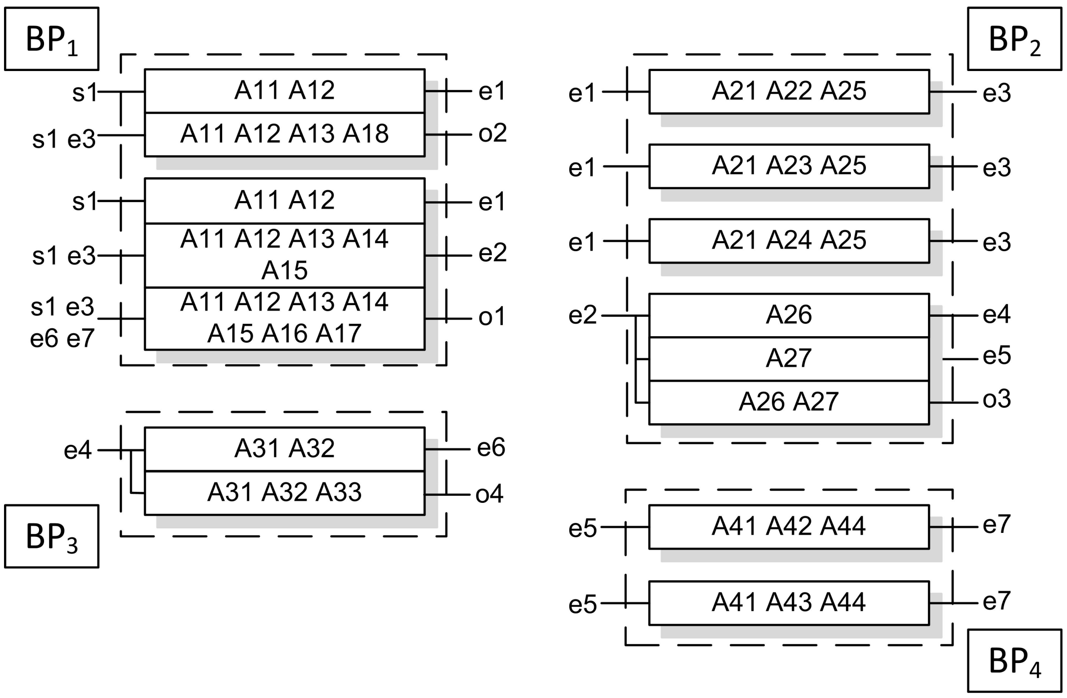


Figure 8.4: Paths and Internal Subsets of the Business-to-Business Collaboration

since we suppose that the effects on two incorrect activities cannot be compensated to generate a correct end event.

Depending on the end events, it is possible to know if the overall collaboration is working correctly. In order to know which activities are failing, the BPIM is used to infer which interactions are incorrect. For example, for the local path shown in Figure 8.3, suppose $e5$ is a *PII* and $o3$ a correct end event. It means that the activity $A27$ is failing, but it is not possible because if the end event $o3$ is correct, the activity $A27$ would be working correctly. It means that the external interaction $e5$ considered as a *PII* actually is not an incorrect interaction. If an interaction is correct all the activities associated to the *IS* of this interaction are correct. A *PII* becomes an Incorrect Interaction (*II*) if it is related to at least one activity that is not a correct activity. This idea is described in the following definition:

Definition 8.7. *Incorrect Interaction (II): Given the set External(IS_i) that includes the output interactions of IS_i,* **e** *is an II if it is a PII and* $\exists\ A_{kl} \in IS_i \mid \forall\ IS_j : j \in 1 \ldots n \wedge i \neq j$ *not* $\exists$ **e**' : **e**' *is a CI* $\wedge$ **e**' $\in$ *External(IS_j)* $\wedge\ A_{kl} \in IS_j$

Therefore, according to the previous definition, the definition of *CI* is reinforced with this new concept, since a *CI* is also a message that is not an Incorrect Interaction.
On this phase of the process, some information is exchanged between the *Diagnosers* in order to get to the point where all of them are aware of the *CIs* and *PIIs* in the collaboration, so that the knowledge has been propagated. These *Diagnosers* are able to infer what external interactions are *CIs* or *IIs* according to Definition 8.7, and to send this information to the business processes to which they are linked through their inputs. To this end, the message traffic flows in the opposite direction to the normal flow of execution of the process instances.
This information shared by the *Diagnosers* are composed of the following fields:

- **Correct interactions** field: the external interactions labelled as *CIs* by the *Diagnoser* of the business process source of the information.
- **Incorrect interactions** field: the external interactions labelled as *PIIs* by the *Diagnoser* of the business process source of the information.
- **Source** field: indicates the source of the information. It may be: (1) a neighbour *Diagnoser*; or (2) users or customers which performed a complaint to prompt the beginning of the diagnosis process.

In order to proceed with the propagation, each *Diagnoser* needs to receive the information about all the outputs of a path, or at least about some correct end event, before inferring whether the external interactions of that path are *CIs* or *IIs*. Therefore, the propagation takes place as the information of the outputs of the paths is known by the *Diagnosers*. This is not a linear process since several business processes may have paths with a dependency between them. For instance, if a business process BP_a has the paths P_x and P_y, and another business process BP_b has the path P_z, it is possible that P_x depends on P_z, and P_z depends on P_y. In this example, to carry out the propagation, *Diagnoser*$_a$ waits for the information about the outputs of P_x to propagate to P_z. In the same way, *Diagnoser*$_b$ will propagate from P_z to P_y when it has the information about P_z. So, both *Diagnosers* interchange information in both directions.
The procedures in the algorithms in Figures 8.5, 8.6 and 8.7 describe the behaviour of a *Diagnoser* after receiving information about correct or incorrect end events.
For instance, for the example in Figure 8.2, if $o1$ is correct and $o3$, $o4$ are incorrect the propagated information is as follows:

```
procedure ReceiveInteractionInformation($X_{OK}$, $X_{KO}$, $BP_i$)
    label as $CIs$ the external interactions which are in $X_{OK}$
    for each external interaction $e$ in $X_{KO}$ do
        if $e$ is not labelled as $CI$ then
            label $e$ as $PII$
        end if
    end for
    if the Diagnoser has received the incoming information about all the outputs of
      the path $P$, and $P$ is on the executed instance then
        checkInteractions($P$)
        propagateInformation($P$)
    else
        for each external interaction $e$ in $X_{OK}$, being $IS_e$ the $IS$ whose output is e, do
            store the inputs of $IS_e$ in an (initially empty) set $X'_{OK}$
            label as $CI$ every external interaction $e'$ related to $IS'_e$ so that $IS'_e \subset IS_e$
            store the inputs of $IS'_e$ in $X'_{OK}$
        end for
        send the Information($X'_{OK}$, - , this) to the Diagnosers that are related to this
          BP by means of the set on interactions in $X'_{OK}$ (only if this information has
          not been sent before to those Diagnosers)
        wait for the rest of incoming information
    end if
end procedure
```

Figure 8.5: Algorithm executed when a local diagnoser receives information

- The user sends complaints to *Diagnoser*$_1$, *Diagnoser*$_2$ and *Diagnoser*$_3$ indicating that $o1$ is correct and $o3$, $o4$ are incorrect end events.

- *Diagnoser*$_1$ labels $o1$, $e1$ and $e2$ as *CIs* and propagates to *Diagnoser*$_2$, *Diagnoser*$_3$ and *Diagnoser*$_4$ that $e3$, $e6$ and $e7$ are correct respectively.

- *Diagnoser*$_2$ labels $o3$ as *PII* and, since it has received that $e3$ is correct, labels it as *CI* and propagates that $e1$ is correct.

- *Diagnoser*$_3$ has received that $o4$ is incorrect and $e6$ is correct, so that it labels $e6$ as *CI* and propagates that $e4$ is possibly incorrect to *Diagnoser*$_2$.

- *Diagnoser*$_4$ has received that $e7$ is correct, so that it labels it as *CI* and propagates $e5$ as correct to *Diagnoser*$_2$.

```
procedure CHECKINTERACTION(Path P)
    /*Definition 8.7*/
    for each interaction $e$ labelled as a PII do
        if $IS_i \in P$ and $A_{kl}$ is an activity of $IS_i$ and it does not exist another $IS_j$ that
        contains $A_{kl}$ with a message $e'$ labelled as a CI and $e'$ is different from $e$
        and $e'$ belongs to $Output(IS_j)$ then
            label $e$ as an II
        else
            label $e$ as a CI
        end if
    end for
    for each output interaction $e$ labelled as II do
        label as PII the interactions which are inputs of the ISs where $e$ is an output
    end for
end procedure
```

Figure 8.6: Procedure for labelling interactions as correct or incorrect

- The process continues until all *Diagnosers* have propagated all the knowledge about the inputs of all their paths, so that the local diagnosis phase can start.

8.3.3 Local Diagnosis Phase

Using the information collected from the previous step, the local diagnosis is performed by the *Diagnosers* whose external interactions are labelled as *II*.
The local diagnosis process also has off-line and on-line phases:

- To build a signature matrix: With the information obtained from the COM, a signature matrix is created. This matrix relates each external interaction and end events of each business process with the activities of the *ISs* where these interactions participate. The matrix has as rows as *number of interactions* and as columns as *number of activities*. This process is performed only once, storing pre-compiled information. The construction of the matrix is an adaptation of the algorithm presented in Ceballos et al. (2007) for business processes. An example is shown in Figure 8.8(a) that represents the signature matrix for one of the paths of BP_2 in the example.

- When the *IIs* are known, only the part of the matrix related to the possible incorrect activities is analysed. It means that only the rows of the *IIs* and

```
procedure PROPAGATEINFORMATION(Path P)
    for all $BP_i$ related to $P$ by the set of interactions $X$ do
        if there exists a subset of interactions $X_i$ which are inputs of $P$ and outputs
        of $BP_i$ and are labelled as PIIs then
            send the information ($X - X_i$, $X_i$, this) to $Diagnoser_i$
        else
            send the information ($X$, - , this) to $Diagnoser_i$ (only if this information
              has not been sent before to the $Diagnoser_i$)
        end if
    end for
end procedure
```

Figure 8.7: Procedure for propagating information

	A26	A27
e4	X	
e5		X
o3	X	X

a)

	A27
e5	X
o3	X

b)

Figure 8.8: Signature matrix of a path of BP_2

the activities not related to any *CI* participate in the local diagnosis. With this subset of relations among interactions, end events and activities, the set of activities which are not working correctly must be found. According to diagnosis theory, the best way to do that is by calculating the minimal hitting sets.

Figure 8.8(b) shows the part of the signature matrix of the path analysed to perform the local diagnosis if $e4$ is *CI* and $e5$, $o3$ are *IIs*.

Finally, the minimal hitting sets are the diagnosis of the business processes. For the example in Figure 8.8(b) there is a minimal hitting set, which is $\{A27\}$.

The local diagnosis can be improved storing all the final diagnoses according to the *IIs*. That is, if an *II* has already been analysed, the diagnosis will be very efficient. Each *Diagnoser* counts on a local list where it stores the fault signature from the local diagnosis executed until that moment, so that when a *Diagnoser* has to diagnose its activities after a new execution, previously it checks whether it has already performed the same diagnosis before.
Each element of the lists contains two fields with the next information:

i. External interactions labelled as *IIs*.

ii. Activities which can fail according to the *IIs* detected.

Local diagnosis results depend on the subset of external interactions which are *IIs*, since two different analysis with the same set of *IIs* obtain the same minimal hitting sets of activities. Using a local list to store previous results enables a faster process of diagnosis, because of the re-utilization of the information obtained so that it is not necessary to carry out the same analysis twice.
Finally, the global diagnosis D_g can be seen as the union of all local diagnoses: $D_g = D_1 \cup D_2 \cup ... \cup D_n$, being n the number of business processes in the business-to-business collaboration.

8.4 Empirical Evaluation

The execution complexity of diagnosing collaborations between business processes is a problem of distributed nature, which entails achieving coherence or consistency among the local diagnosers that take part in the communications during the propagation phase. Thereby, to reduce this complexity, our solution proposes the elaboration of a previously compiled model.
Therefore, we wish to empirically evaluate the performance of the developed diagnosis method, so that we can ensure it takes acceptable time to diagnose business-to-business collaborations.

8.4.1 Experimental Design

This section shows the temporal results of performing several tests to the algorithm previously explained. This tests have been carried out over business-to-business collaborations generated randomly (by using the *Process Log Generator* tool (PLG), detailed in Burattin and Sperduti (2010)), which count on different number of business processes (from 4 to 10 processes per collaboration). The tests were performed both with pre-compiled and without pre-compiled model.
The aim of these tests is the comparison between the execution time necessary to perform the diagnosis with and without the previous analysis that obtains the pre-compiled model. For this reason, we measure the execution time spent by the algorithms in performing the diagnosis process for business-to-business collaborations composed by different number of business processes, specifically from 4 to 10. This execution time is measured in milliseconds since the discrepancy between observed and expected behaviour is detected, until the obtaining of the

local diagnosis indicating the activities responsible for the abnormal working of the collaboration.
For each different number of business processes (4 to 10), 10 collaboration of processes have been generated, and 100 random messages have been sent to each one. Therefore, 70 different processes have been diagnosed 100 times. These tests have been executed in several computers with similar characteristics: Intel Core2 1,86GHz processor, and 2GB RAM memory.

8.4.2 Performance Results

In this subsection, the execution time of the diagnosis process is measured and presented in Table 8.2, whose content is organized as follows:

- Each row shows the measured times divided into the *number of business processes (BPs)* that compose the different randomly generated collaborations.
- The columns present the differences between pre-compiled and not pre-compiled model, appearing as data the average time (AV) spent by the diagnosis process for processes of a determined size, and the average time spent in the tests performed to the collaboration that has had the best time (AV_{min}) and the worse time (AV_{max}) in the diagnosis process. The last column shows the delay of the no pre-compiled execution with respect to the pre-compiled one.
- The *delay* in the last column shows the extra time spent by the diagnosis process without pre-compiled model. Specifically, this time is spent in the analysis necessary to obtain the equivalent structure used in the diagnosis process, since whether the pre-compiled model is not used, this analysis must be performed each time that a discrepancy is detected.

In order to facilitate the visualization, Figure 8.9 shows a graphic representation of the results. Two different lines appear representing the diagnosis processes with and without pre-compiled model.
It is possible to see the delay that a no pre-compiled model causes in the diagnosis process. Likewise, it is appreciable that the greater number of business processes interact in the collaboration, the more time saved by the diagnosis process by means of calculating the pre-compiled model off-line. The improvement that is get for a scenario may be more or less significant than the one represented in

Table 8.2: Temporal results

number	Pre-compiled Model			No Pre-compiled Model			
of BPs	AV	AV_{min}	AV_{max}	AV	AV_{min}	AV_{max}	*delay*
4	102.83	68.60	142.28	170.02	138.31	207.02	67.19
5	178.97	166.98	192.25	270.41	233.43	319.59	91.44
6	193.11	171.20	213.59	306.53	253.53	468.48	113.42
7	257.05	203.86	313.29	384.82	319.93	467.79	127.77
8	302.79	291.77	313.17	445.68	427.96	455.15	142.89
9	314.31	270.27	344.38	482.77	412.70	524.63	168.46
10	407.25	253.69	465.30	605.44	552.67	646.98	198.19

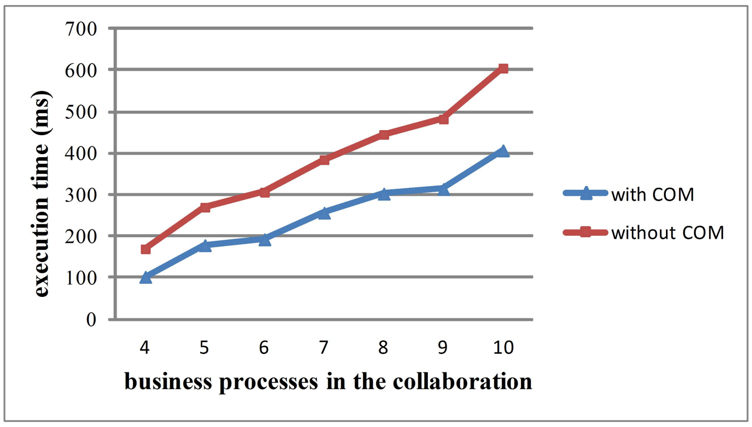


Figure 8.9: Graphic representation of the results

Figure 8.9 That depends on the structural complexity degree of the collaborations to diagnose. Nevertheless, the execution time is always improved to a greater or lesser degree.

8.5 Summary

This chapter provides a method to diagnose business-to-business collaborations at runtime only using structural analysis. The diagnosis process starts from the assumption that model of the collaboration, which only counts on control flow perspective, has already been verified correct. Therefore, the faults to diagnose

are those which appears after the execution of a process instance due to an activity (or a set of them) is not working as it was modelled.
Due to there is no entity with a global knowledge about the overall model of the collaboration, our contribution uses a set of local diagnosers, one per business process in the choreography, which know the local model of their associated processes. Moreover, those diagnosers count on the Business Process Interaction Model in order to manage the interaction between the different business processes.
In order to improve the computational complexity, which is costly in terms of execution time, the local diagnosers analyse the model of the local processes to attain an auxiliary compiled model (COM), in an off-line way. This model is the base for the rest of the diagnosis process, since it is use for both propagation and local diagnosis tasks.
As a result, our contribution can diagnose the set of activities which are responsible for a malfunction in a business-to-business collaboration (1) without needing an entity with global knowledge of the overall collaboration; and (2), although the information regarding the model of collaborations is distributed, our solution gets satisfactory results in a very accurate and efficient way due to the COM attained off-line and the reusing of previously obtained diagnosis results.

Part VI

Conclusions and Future Work

Chapter 9

Final Remarks

Since companies need to ensure the quality of their processes to become more competitive, the current Thesis Dissertation proposes the adaptation of fault diagnosis techniques, widely analysed for other areas of knowledge, to business processes, performing new methodologies in accordance with the features of this kind of processes.

In order to facilitate a quick identification of the activities where the processes fail, either at early stages to isolate faults in the model, or at run-time when the faults determine a wrong behaviour of the processes during their execution, we provide different proposals to improve the quality of the processes by automatically applying diagnosis techniques at different stages of the business process lifecycle.

To this end, both perspectives of the model have been considered, since both of them are necessary to describe the behaviour of business processes, taking into account that the data flow perspective is not always available or modelled as a contract.

In this way, the current Thesis Dissertation verifies the correctness of business process models which count on Business Data Constraints to describe the data semantics of a business process for the representation of the relations between data values. These Business Data Constraints are understood as a subset of business compliance rules which represent the semantic relation between the data values that are introduced, read and modified during the business process instances. Therefore, their analysis has been used to locate faults in the modelling of the behaviour of the activities from the data flow point of view.

Likewise, since some diagnosis methods applied in this Thesis Dissertation are based on Model-based Diagnosis techniques, it is necessary to count on an ob-

servational model which is able to provide with sufficient observations the actual behaviour of business processes. This makes possible the comparison between observed and expected behaviour, getting a more precise diagnosis. Therefore, also a contribution to improve the diagnosability by determining the locations where the observations should be performed has been proposed.
In order to carry out all these diagnosis tasks, we propose their insertion between the basic stages of the business process lifecycle, giving rise to a framework including four modules to provide diagnosis functionalities in the four contributions presented in this Thesis Dissertation:

- In detail, with reference to the diagnosis of the model, Chapter 5 presents a contribution to **diagnose semantic business process models** as regards the data flow perspective. To that end, we have proposed workflow data graphs as formalization of semantic business process models together with two correctness notions, may and must-correctness, that can be verified for workflow data graphs. Workflow data graphs model semantic business process by including pre and postconditions for the activities to describe their individual behaviours. This contribution also proposes two correctness notions for semantic business process models, called may and must-correctness, and presents a diagnosis approach for their verification, which consists of several phases.

  First, preprocessing is applied to detect basic data anomalies. Then, the workflow data graph is translated into an IP formulation that models the executable instances, and into a CSP formulation that models the data states acceptable according to the pre and postconditions of the activities. The combined IP and CSP model can be efficiently solved using Constraint Programming techniques. In case of an error, feedback is provided in the form of an error path showing where the workflow gets stuck under certain conditions over the data flow. Such feedback provides valid information for the business process designer to fix future errors before the workflow is deployed. The approach is complete, so it always generates accurate feedback in case of an error.

  The implementation of the approach has been performed by extending the DiagFlow tool (Eshuis and Kumar, 2010) initially conceived to verify only the control flow perspective. The tool diagnoses business process models in an extended XPDL format. The XPDL extension is needed to store the semantic information of each business process, including the information about the data flow by adding pre and postconditions in the activities. Performance evaluation of the tool shows that the algorithms scale well for large workflow models with data flows, despite the high worst-case complexity of solving constraints satisfaction programs.

- As regards the attainment of sufficient observations in the observational model, Chapter 6 presents a contribution to **improve the diagnosability in business process models**. The proposal allocates test points to make certain parts of the data flow observable, by dividing the overall business process into clusters of activities to facilitate the isolation of the activity or activities which are responsible for future abnormal behaviour.

  In order to perform the allocation of test points, the business processes are modelled as Constraint Satisfaction Problems so that the problem can be efficiently solved using Constraint Programming techniques.

  This contribution presents three different objectives to be achieved, which depend on the requirements of the user or the problem specification. They consider both the enhancement of the business processes diagnosability and the improvement of the computational complexity of isolating faults in a later diagnosis process.

  On the topic of allocating test points in business processes in order to improve their diagnosability, to the best of our knowledge, only our proposals (Borrego et al., 2010b,c) have been published. Likewise, the approach has been implemented in the Test-Point Allocator tool (Borrego et al., 2011). The tool allocates test points in business processes designed using BPMN 2.0, in accordance with the three selected objectives.

- Regarding the diagnosis at run-time of business processes which count on data flow perspective modelled as contracts describing the behaviour, Chapter 7 provides a contribution to **diagnose business processes based on business data constraints**. The contribution presents a method to diagnose business process instances in terms of a set of business data constraints and for input and output data.

  To this end, the diagnosis method in the contribution takes into account the different gateways which influence in the execution trace of every instance, so that, during the diagnosis process, it only considers the activities involved on the trace of the business process instance where the fault was detected.

In order to perform the diagnosis, the business data constraints are described and modelled as Constraint Satisfaction Problems to also use well-known solvers to get a solution efficiently. Two types of techniques based on constraint programming are herein defined for the attainment of the minimal diagnosis in an efficient way: (1) modelling of the problem by using of reified constraints to get a more efficient solution in terms of execution time; and (2) determination of Minimal Unsatisfiable Subsets of constraints to ensure the obtaining of the minimal diagnoses.

This contribution has been previously addressed in Borrego et al. (2010a,d).

- Finally, a forth contribution is proposed to provide a method to diagnose business processes whose data flow perspective is not available or even its semantics cannot be modelled as contracts. Thereby, Chapter 8 presents a solution to **diagnose business processes by means of structural analysis** of the control flow perspective, based on Model-based diagnosis principles.

  The contribution details a method to perform the diagnosis of business-to-business collaborations. To this end, we propose to associate a local diagnoser to each process in the collaboration, which count on a Business Process Interaction Model (BPIM) to manage the communications between processes. Moreover, in order to get a more efficient diagnosis, the BPIM and the control flow perspective on the business process model are analysed off-line to obtain the Compiled Orchestration Model (COM), which is an auxiliary structure used in the diagnosis process so that the execution time needed to perform the global diagnosis is improved.

  During the diagnosis process, the local diagnosers work with public and private information (external and internal interactions), and the diagnosis process is performed without needing to know neither all the information exchanged between the activities nor the overall business process model. By means of the use of previously compiled knowledge, a high temporal efficiency is obtained, since the preparation of the Compiled Orchestration Model (COM) is performed off-line and used at different stages of the process.

  This proposal of fault diagnosis of business-to-business collaborations is an innovative solution to the diagnosis of business processes without the availability of the data flow perspective, although it does not find the global minimal diagnosis, since the set of activities found as a solution is not a global minimal hitting set (Borrego et al., 2008b,a, 2009).

In short, the current Thesis Dissertation provides diagnosis methods which come to reinforce the business process lifecycle with new policies to get high-quality business processes. The proposals offer solutions to different possible scenarios at different stages of the business processes development, always considering the efficiency as an important factor to attain, developing their evaluations through performance measurements to test their validity in terms of the accuracy of their results and execution times.

Chapter 10

Future Work

The current Thesis Dissertation presents four contributions to facilitate the identification of causes which produce abnormal behaviours in business processes. Those research works can be extended to include aspects which have not been explored before, or even to provide new functionalities.

In the first instance, regarding the verification of semantic business process models, the contribution provided in Chapter 5 is intended to be extended with the next ideas:

- To provide the business analysts with additional feedback in case of a violation, in order to make easier the job of fixing the problem which causes the incorrectness of the model. To this end, it would be interesting to perform an analysis of the pre and postconditions of the activities involved in a process instance whose simulation has been verified incorrect, as well as the domains of the involved data, in order to accurately indicate the points to be checked to fix the problem. It entails the use of techniques to minimize the number of preconditions, postconditions and/or data whose incorrect modelling can explain the incorrectness of the overall business process model.

- To consider the conditions in the gateways in order to simulate instances of the business processes. This way, the valuations of data also influence in the selection of branches which compose an instance, which would no longer be based only on the topology of the business processes. As a result, it is also possible to determine if there is any branch which would never be reached according to the conditions of the gateways, due to the set of gate conditions which should be satisfied to reach a branch may become overconstrained.

Related to the improvement of the diagnosability of business processes (cf. Chapter 6), there are certain extensions regarding to the criterion for allocating test points which are intended to be achieved:

- Allocation of test points in accordance with the reliability of each activity, so that the test points should be mostly allocated in flows where there are more unreliable activities. To this end, it is necessary the definition of heuristics to determine the reliability level of an activity in a business process, considering factors such as information about the percentage of faults presented by each activity in previous executions of the process.

- Allocation of test points in accordance with the diagnosability of each activity in an independent way. That is, it is easier to identify the fault in an activity whose behaviour is accurately modelled by means of pre and post-conditions, than to determine the incorrect behaviour of an activity without that kind of model. Therefore, it would be more useful to perform observations of the data managed by the later activity in order to count on more information for the diagnosis process.

- Since, the diagnosability level of a business process is directly related to the ability to discriminate between faults when they occur, several metrics can be used in order to perform a comparison of the diagnosability level of a business process before and after the allocation of test points, such as simply the number of activities per cluster of the longest chain of activities that can be executed in a process instance without any observation of the data flow between these activities. However, since the allocation of test points aims to facilitate a later diagnosis process, it is necessary the definition of a metric to obtain the number of fault modes which cannot be discriminated in the worst case, in order to obtain the largest set of activities whose instances of abnormal behaviour are detected by a certain set of the allocated test points.

Consequently, the combination of the goals pursued in the three different objectives presented in the contribution in Chapter 6 with these two new ideas described above gives rise to multi-objective problems, which would also be faced up as future work.
Regarding the diagnosis of business processes at run-time, the contributions in Chapters 7 and 8 are also intended to be extended by considering the information derived from the execution of several instances to perform the respective diagnosis processes. In this way, the feedback provided after the diagnosis could be

even more accurate to help business analysts to determine the exact cause of the incorrect behaviour. In detail:

- In the case of the contribution for the diagnosis based on compliance rules (cf. Chapter 7), it is intended to adapt the proposed methodology to perform the diagnosis of business processes before the instance is still in execution. The idea is to observe the data flow at the locations where the test points are allocated, and use these data to determine that the process instance in progress will finish its execution presenting some kind of inconsistency between the compliance rules and the managed data.

- Also, regarding the presented contribution in diagnosis based on structural analysis in Chapter 8, the combination of information from different instances can help provide a minimal diagnosis by enabling to discard some parts as possible responsible of the detected faults.

And finally, although each contribution has been implemented and tested independently, two of them even counting on their own tools, it would be desirable to develop a global framework supported by a tool including the functionality provided by each contribution. This way, the tool would complement business process management systems by providing them with capabilities to reinforce the basic business process lifecycle, adding verification and diagnosis at different stages.

Bibliography

Affane, M. S., Bennaceur, H., 1998. A weighted arc consistency technique for max-csp. In: Proc. of the 13th ECAI. pp. 209–213.

Alpern, B., Wegman, M. N., Zadeck, F. K., 1988. Detecting equality of variables in programs. In: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. San Diego, California, pp. 1–11.

Ardissono, L., Console, L., Goy, A., Petrone, G., Picardi, C., Segnan, M., Dupre, D., 2005. Enhancing web services with diagnostic capabilities. In: Web Services, 2005. ECOWS 2005. Third IEEE European Conference on. IEEE, pp. 10–pp.

Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M., 2006. Fault tolerant web service orchestration by means of diagnosis. In: EWSA. pp. 2–16.

Awad, A., Smirnov, S., Weske, M., 2009. Towards resolving compliance violations in business process models. GRCIS. CEUR-WS. org.

Becker, J., Ahrendt, C., Coners, A., Wei, B., Winkelmann, A., 2011. Modeling and analysis of business process compliance. In: Nttgens, M., Gadatsch, A., Kautz, K., Schirmer, I., Blinn, N. (Eds.), Governance and Sustainability in Information Systems. Vol. 366 of IFIP Publications. Springer, pp. 259–269.

Bistarelli, S., Montanari, U., Rossi, F., 1995. Constraint solving over semirings. In: Mellish, C. (Ed.), IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence. Montreal.

Biswas, G., Cordier, M.-O., Lunze, J., Trave-Massuyes, L., Staroswiecki, M., 2004. Diagnosis of complex systems: Bridging the methodologies of the fdi and dx communities. Systems, Man, and Cybernetics, Part B, IEEE Transactions on 34 (5), 2159–2162.

Biteus, J., Nyberg, M., Frisk, E., 2008. An algorithm for computing the diagnoses with minimal cardinality in a distributed system. Engineering Applications of Artificial Intelligence 21, 269–276.

Bocconi, S., Picardi, C., Pucel, X., Dupré, D. T., Travé-Massuyès, L., 2007. Model-based diagnosability analysis for web services. In: AI*IA'07. pp. 24–35.

Booch, G., Rumbaugh, J., Jacobson, I., 2005. Unified Modeling Language User Guide, The (Addison-Wesley Object Technology Series). Addison-Wesley Professional.

Borrego, D., Gasca, R. M., Gómez-López, M. T., Barba, I., 2009. Choreography analysis for diagnosing faulty activities in business-to-business collaboration. In: 20th International Workshop on Principles of Diagnosis. DX-09. Stockholm, Sweden, pp. 171–178.

Borrego, D., Gasca, R. M., Gómez-López, M. T., Parody, L., 2010a. Contract-based diagnosis for business process instances using business compliance rules. In: Proceedings of the 21st International Workshop on Principles of Diagnosis. (DX-10). pp. 169–176.

Borrego, D., Gómez-López, M. T., Gasca, R. M., 2008a. Diagnosing distributed systems using only structural and qualitative information. International Transactions on Systems Science and Applications (to appear).

Borrego, D., Gómez-López, M. T., Gasca, R. M., 2011. Test-point allocator tool. In: `http://www.lsi.us.es/~quivir/index.php/Main/Downloads`.

Borrego, D., Gómez-López, M. T., Gasca, R. M., Barba, I., 2008b. Diagnosing business processes execution using choreography analysis. In: XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008). pp. 13–24.

Borrego, D., Gómez-López, M. T., Gasca, R. M., Ceballos, R., 2010b. Determination of an optimal test points allocation for business process analysis. In: IEEE/Ifip Network Operations and Management Symposium Workshops. Workshop on Business Driven It Management. IEEE Communications Society, pp. 159–160.

Borrego, D., Gómez-López, M. T., Gasca, R. M., Ceballos, R., 2010c. Improving the diagnosability of business process management systems using test points. In: 6th Workshop on Business Process Intelligence (BPI 2010).

Borrego, D., Gómez-López, M. T., Gasca, R. M., Parody, L., 2010d. Diagnosis de errores en la gestión de procesos software con programación con restricciones. In: X Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010). pp. 23–34.

BOS, 2011. Bonita open solution. In: `http://www.bonitasoft.org`. BonitaSoft.

Bruni, R., 2003. Approximating minimal unsatisfiable subformulae by means of adaptive core search. Discrete Appl. Math. 130 (2), 85–100.

Burattin, A., Sperduti, A., 2010. Plg: a framework for the generation of business process models and their execution logs. In: In Proceedings of the 6th International Workshop on Business Process Intelligence (BPI 2010). Stevens Institute of Technology; Hoboken, New Jersey, USA.

C. Commault, J.-M. D., Agha, S. Y., 2006. Structural analysis for the sensor location problem in fault detection and isolation. In: Proceedings of the IFAC Symposium SAFEPROCESS' 2006. Beijing, China, pp. CD–ROM.

Cardoso, J., july 2005. Evaluating the process control-flow complexity measure. In: Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference. Vol. 2. pp. xxxiii–856.

Cassar, J., Staroswiecki, M., 1997. A structural approach for the design of failure detection and identification systems. IFAC-IFIP-IMACS Conf. on Control of Industrial Processes, Belfort, France.

Çatalyürek, V., Aykanat, C., 1999. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems 10 (7), 673–693, cited By (since 1996): 103.

Ceballos, R., Cejudo, V., Gasca, R. M., Valle, C. D., 2005. A topological-based method for allocating sensors by using csp techniques. In: CAEPIA. Vol. 4177 of Lecture Notes in Computer Science. Springer, pp. 62–68.

Ceballos, R., Gómez-López, M. T., Gasca, R. M., del Valle, C., 2006. Integración de técnicas basadas en modelos para la determinación de la diagnosis mínima de un sistema. In: Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial No. 31. pp. 41–51.

Ceballos, R., Gómez-López, M. T., Gasca, R. M., del Valle, C., 2007. A compiled model for faults diagnosis based on different techniques. AI Communications 20 (1), 7–16.

Cetin, S., Altintas, N., Solmaz, R., 2006. Business rules segregation for dynamic process management with an aspect-oriented framework. In: Eder, J., Dustdar, S. (Eds.), Business Process Management Workshops. Vol. 4103 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp. 193–204.

Console, L., Dressler, O., 1999. Model-based diagnosis in the real world: Lessons learned and challenges remaining. In: IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1393–1400.

Console, L., Picardi, C., Dupré, D. T., 2007. A framework for decentralized qualitative model-based diagnosis. In: IJCAI 2007. pp. 286–291.

Console, L., Picardi, C., Ribaudo, M., 2000. Diagnosis and diagnosability analysis using process algebra. In: In Proc. 11th Int. Workshop on Principles of Diagnosis (DX).

Cordier, M.-O., Dague, P., Dumas, M., Lévy, F., Montmain, J., Staroswiecki, M., Travé-Massuyès, L., 2000. A comparative analysis of ai and control theory approaches to model-based diagnosis. In: ECAI. pp. 136–140.

Cytron, R., Ferrante, J., Rosen, B. K., Wegman, M. N., Zadeck, F. K., 1991. Efficiently computing static single assignment form and the control dependence graph. ACM Transactions on Programming Languages and Systems 13, 451–490.

D. Maquin, M. L., Ragot, J., 1997. Fault detection and isolation and sensor network design. European Journal of Automation 31 (2), 393–406.

Darwiche, A., 1998. Model-based diagnosis using structured system descriptions. Journal of Artificial Intelligence Research 8, 165–222.

Davis, R., 1984. Diagnostic reasoning based on structure and behavior. In Artificial Intelligence 24, 347–410.

de Kleer, J., Mackworth, A. K., Reiter, R., 1992. Characterizing diagnoses and systems. Artif. Intell. 56 (2-3), 197–222.

de la Banda, M. J. G., Stuckey, P. J., Wazny, J., 2003. Finding all minimal unsatisfiable subsets. In: PPDP '03: Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming. ACM, pp. 32–43.

Dechter, R., 1992. Constraint networks. Encyclopedia of Artificial Intelligence, Second Edition, 276–285.

Dechter, R. (Ed.), 2003. Constraint Processing. Morgan Kaufmann Publisher.

Dressler, O., Struss, P., 2003. A toolbox integrating model-based diagnosability analysis and automated generation of diagnostics. In: the 14th International Workshop on Principles of Diagnosis (DX03). Washington, D.C., USA, pp. 99–104.

Dulmage, A. L., Mendelsohn, N. S., 1959. A structure theory of bi-partite graphs of finite exterior extension. Transactions of the Royal Society of Canada 53 (3), 1–13.

Dynamic Decision Technologies, 2011. Dynadec web page.
URL `http://dynadec.com/`

Éric Grégoire, Mazure, B., Piette, C., 2007. Local-search extraction of muses. Constraints 12 (3), 325–344.

Eshuis, R., Kumar, A., 2008. An integer programming based approach for diagnosing workflows. Tech. rep., Beta Working Paper Series, WP 264, Eindhoven University of Technology.

Eshuis, R., Kumar, A., August 2010. An integer programming based approach for verification and diagnosis of workflows. Data Knowl. Eng. 69, 816–835.

Eshuis, R., Wieringa, R., July 2004. Tool support for verifying uml activity diagrams. IEEE Transactions on Software Engineering 30 (7), 437–447.

Fattah, Y. E., Holzbaur, C., 1994. A clp approach to detection and identification of control system component failures. In: Proceedings of the Workshop on Qualitative and Quantitative Approaches to Mode-Based Diagnosis, Second International Conference on Intelligent Systems Engineering. pp. 97–107.

Freuder, E., Wallace, R., 1992. Partial constraint satisfaction. Artificial Intelligence (58), 21–70.

Frisk, E., Krysander, M., 2007. Sensor placement for maximum fault isolability.

Frohlich, P., de Almeida Mora, I., Nejdl, W., Schroeder, M., 1997. Diagnostic agents for distributed systems. In: ModelAge Workshop. pp. 173–186.

G. Verfaillie, M. L., Schiex, T., 1996. Russian doll search. In: Proc. of the 13th AAAI. pp. 181–187.

Gasca, R. M., Valle, C., Gómez-López, M. T., Ceballos, R., 2007. Nmus: Structural analysis for improving the derivation of all muses in overconstrained numeric csps, 160–169.

Genesereth, M. R., 1984. The use of design descriptions in automated diagnosis. Artif. Intell. 24 (1-3), 411–436.

Ghose, A., Koliadis, G., 2007. Auditing business process compliance. In: ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing. Springer-Verlag, Berlin, Heidelberg, pp. 169–180.

Gómez-López, M. T., Ceballos, R., Gasca, R. M., Del Valle, C., 2009. Developing a labelled object-relational constraint database architecture for the projection operator. Data Knowl. Eng. 68 (1), 146–172.

Gómez-López, M. T., Ceballos, R., Gasca, R. M., Valle, C. D., 2004. Constraint databases technology for polynomial models diagnosis. In: 15th International Workshop on Principles of Diagnosis. pp. 215–220.

Gómez-López, M. T., Gasca, R. M., 2010. Run-time monitoring and auditing for business processes data using contraints. In: 6th Workshop on Business Process Intelligence (BPI 2010).

Graphviz/Dot, 2011. In: `http://www.graphviz.org`.

Ha, B., Bae, J., Park, Y., Kang, S., 2006. Development of process execution rules for workload balancing on agents. Data & Knowledge Engineering 56 (1), 64–84.

Hay, D., Healy, K. A., Hall, J., Bachman, C., Breal, J., Funk, J., Healy, J., Mcbride, D., Mckee, R., Moriarty, T., et al., 2000. Defining business rules. what are they really? the business rules group. Business, 4–5.

Heller, U., Struss, P., 2001. G+de - the generalized diagnosis engine. In: DX-2001, 12th International Workshop on Principles of Diagnosis. pp. 79–86.

Heravizadeh, M., Mendling, J., Rosemann, M., 2009. Dimensions of business processes quality (qobp). In: Business Process Management Workshops. Springer, pp. 80–91.

Huang, S.-M., Chu, Y.-T., Li, S.-H., Yen, D. C., 2008. Enhancing conflict detecting mechanism for web services composition: A business process flow model transformation approach. Inf. Softw. Technol. 50 (11), 1069–1087.

Hyun Son, J., Ho Kim, M., 2001. Improving the performance of time-constrained workflow processing. Journal of Systems and Software 58 (3), 211–219.

IBM, 2003. JSolver 2.1, Reference manual, *unpublished.*

Jablonski, S., Bussler, C., 1996. Workflow management - modeling concepts, architecture and implementation. International Thomson.

Junker, U., August 2001. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In: IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1). Seattle, WA, USA.

Kanellakis, P. C., Kuper, G. M., Revesz, P. Z., 1992. Constraint query languages.

Karamanolis, C., Giannakopoulou, D., Magee, J., Wheater, S. M., 2000. Model checking of workflow schemas. In: Proceedings of IEEE EDOC 2000. pp. 170–179.

Karypis, G., Aggarwal, R., Kumar, V., Shekhar, S., 1999. Multilevel hypergraph partitioning: applications in vlsi domain. Ieee Transactions On Very Large Scale Integration Vlsi Systems 7 (1), 69–79.

Kask, K., 2000. New search heuristics for max-csp. Principles and Practice of Constraint Programming–CP 2000, 262–277.

Kleer, J. D., Williams, B., 1987. Diagnosing multiple faults. Art. Int.

Kumar, V., 1992. Algorithms for constraint satisfaction problems: A survey. AI Magazine 13 (1), 32–44.

Larrosa, J., Meseguer, P., 1999. Partition-based lower bound for max-csp. In: Principles and Practice of Constraint Programming–CP99. Springer, pp. 303–315.

Liffiton, M., Sakallah, K., 2008. Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning 40 (1), 1–33.

Liffiton, M. H., Sakallah, K. A., 2005. On finding all minimally unsatisfiable subformulas. In: In International Conference on Theory and Applications of Satisfiability Testing (SAT'05). pp. 173–186.

Ly, L., Rinderle-Ma, S., Göser, K., Dadam, P., 2009. On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers 1, 25.

Madron, F., Veverka, V., 1992. Optimal selection of measuring points in complex plants by linear models 38 (2), 227–236.

Mark Liffiton Michael, M. D. M., Pollack, M. E., 2007. Identifying conflicts in overconstrained temporal problems.

Mauss, J., Tatar, M., 2002. Computing minimal conflicts for rich constraint languages.

Mears, C., Garcia De La Banda, M., Wallace, M., 2009. On implementing symmetry detection. Constraints 14 (4), 443–477.

Moffitt, M. D., Pollack, M. E., 2005. Applying local search to disjunctive temporal problems. In: IJCAI. pp. 242–247.

Mozetic, I., Holzbauer, C., 1993. Controlling the complexity in model–based diagnosis.

Namiri, K., Stojanovic, N., 2007. A semantic-based approach for compliance management of internal controls in business processes. In: CAiSE Forum.

Narendra, N., Varshney, V., Nagar, S., Vasa, M., Bhamidipaty, A., 2008. Optimal control point selection for continuous business process compliance monitoring. In: Service Operations and Logistics, and Informatics, 2008. IEEE/SOLI 2008. IEEE International Conference on. Vol. 2. IEEE, pp. 2536–2541.

Oh, Y., Mneimneh, M. N., Andraus, Z. S., Sakallah, K. A., Markov, I. L., 2004. Amuse: a minimally-unsatisfiable subformula extractor. In: DAC '04: Proceedings of the 41st annual conference on Design automation. ACM, New York, NY, USA, pp. 518–523.

OMG, 2011. Object Management Group, Business Process Model and Notation (BPMN) Version 2.0. OMG Standard.

Pencolé, Y., Cordier, M.-O., 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. Artificial Intelligence 164 1-2, 121–170.

Petri, C. A., 1962. Kommunikation mit automaten. Ph.D. thesis, Universitt Hamburg.

Provan, G., 2002. A model-based diagnosis framework for distributed system. In: 13th International Workshop on Principles of Diagnosis. pp. 16–24.

Reiter, R., 1987. A theory of diagnosis from first principles. Artificial Intelligence 32 (1), 57–95.

Roos, N., ten Teije, A., Witteveen, C., 2003. Multi-agent diagnosis with semantically distributed knowledge. In: Proceedings of the Belgium-Netherlands Artificial Intelligence Conference (BNAIC).

Ross, R. G., 2009. What is a business rule? practicable business rules. Business.

Rossi, F., van Beek, P., Walsh, T. (Eds.), 2006. Handbook of Constraint Programming. Elsevier.

Roychoudhury, I., Biswas, G., Koutsoukos, X., 2009. Designing distributed diagnosers for complex continuous systems. Automation Science and Engineering, IEEE Transactions on 6 (2), 277–290.

Sadiq, S. W., Governatori, G., Namiri, K., 2007. Modeling control objectives for business process compliance. In: BPM. pp. 149–164.

Sadiq, S. W., Orlowska, M. E., Sadiq, W., Foulger, C., 2004. Data flow and validation in workflow modelling. In: Schewe, K.-D., Williams, H. E. (Eds.), ADC. Vol. 27 of CRPIT. Australian Computer Society, pp. 207–214.

Sadiq, W., Orlowska, M. E., 2000. Analyzing process models using graph reduction techniques. Information Systems 25, 117–134.

Scheer, A., 2000. ARIS–business process modeling. Springer Verlag.

Schiex, T., Fargier, H., Verfaillie, G., 1995. Valued constraint satisfaction problems: Hard and easy problems. In: Mellish, C. (Ed.), IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence. Montreal.

Shah, I., 2011. Direct algorithms for finding minimal unsatisfiable subsets in overconstrained csps. International Journal on Artificial Intelligence Tools 20 (1), 53.

Sidorova, N., Stahl, C., Trcka, N., 2011. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. Information Systems 36 (7), 1026–1043.

Spanache, S., Escobet, T., Travé-Massuyès, L., 2004. Sensor placement optimisation using genetic algorithms. In: Proceedings of the15th International Workshop on Principles of Diagnosis, DX-04. pp. 179–183.

Sponsor, N., 2008. Business rules and business processes. Information Systems Journal 1 (10), 20–24.

Staroswiecki, M., Declerk, P., 1989. Analytical redundancy in non linear interconnected systems by means of structural analysis. In: IFAC Advanced Information Processing in Automatic Control (AIPAC-89). pp. 51–55.

Sun, S. X., Zhao, J. L., Nunamaker, J. F., Sheng, O. R. L., 2006. Formulating the data-flow perspective for business process management. Information Systems Research 17 (4), 374–391.

Sun, S. X., Zhao, J. L., Sheng, O. R. L., 2004. Data flow modeling and verification in business process management. In: AMCIS. Association for Information Systems.

Thompson, G., Goodale, J., 2006. Variable employee productivity in workforce scheduling. European journal of operational research 170 (2), 376–390.

Touré, F., Baïna, K., Benali, K., 2008. An efficient algorithm for workflow graph structural verification. In: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:. OTM '08. Springer-Verlag, Berlin, Heidelberg, pp. 392–408.

Travé-Massuyès, L., Escobet, T., Milne, R., 2001. Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem. In: Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 551–556.

Travé-Massuyès, L., Escobet, T., Olive, X., 2006. Diagnosability analysis based on component-supported analytical redundancy relations. IEEE Transactions on Systems, Man, and Cybernetics, Part A 36 (6), 1146–1160.

Traxler, P., 2008. The time complexity of constraint satisfaction. In: Proceedings of the 3rd international conference on Parameterized and exact computation. IWPEC'08. Springer-Verlag, Berlin, Heidelberg, pp. 190–201.

Tsai, C., Huang, K., Wang, F., Chen, C., 2010. A distributed server architecture supporting dynamic resource provisioning for bpm-oriented workflow management systems. Journal of Systems and Software 83 (8), 1538–1552.

van der Aalst, W., 2004. Business process management demystified: A tutorial on models, systems and standards for workflow management. Lectures on Concurrency and Petri Nets, 21–58.

van der Aalst, W., Hirnschhall, A., Verbeek, H. M. W., 2002. An alternative way to analyze workflow graphs. In: Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02), volume 2348 of Lecture Notes in Computer Science. Springer-Verlag, pp. 535–552.

van der Aalst, W. M. P., 2007. Challenges in business process analysis. In: Filipe, J., Cordeiro, J., Cardoso, J. (Eds.), ICEIS (Selected Papers). Vol. 12 of Lecture Notes in Business Information Processing. Springer, pp. 27–42.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Weske, M., 2003. Business process management: A survey. In: van der Aalst, W. M. P., ter Hofstede, A. H. M., Weske, M. (Eds.), Business Process Management. Vol. 2678 of Lecture Notes in Computer Science. Springer, pp. 1–12.

Wallace, R. J., 1995. Directed arc consistency preprocessing. In: Constraint Processing, Selected Papers. Springer-Verlag, London, UK, pp. 121–137.

Weber, B., Sadiq, S. W., Reichert, M., 2009. Beyond rigidity - dynamic process lifecycle support. Computer Science - R&D 23 (2), 47–65.

Weber, I., Hoffmann, J., Mendling, J., 2010. Beyond soundness: on the verification of semantic business process models. Distributed and Parallel Databases 27 (3), 271–343.

Weske, M., 2007. Business process management. concepts, languages, architectures. Springer.

WFMC, 2005. Workflow management coalition workflow standard: Process definition interface – xml process definition language. Tech. Rep. WFMC-TC-1025, Workflow Management Coalition.

Yassine, A. A., Ploix, S., Flaus, J.-M., 2008. A method for sensor placement taking into account diagnosability criteria. Applied Mathematics and Computer Science 18 (4), 497–512.

Yassine, A. A., Rosich, A., Ploix, S., 2010. An optimal sensor placement algorithm taking into account diagnosability specifications. International Conference on Automation, Quality and Testing, Robotics 2, 1–6.

Zhang, Y., Chen, X., Liu, G., Qiu, J., Yang, S., 2009. Optimal test points selection based on multi-objective genetic algorithm. In: Testing and Diagnosis, 2009. ICTD 2009. IEEE Circuits and Systems International Conference on. IEEE, pp. 1–4.

Zhao, J., Stohr, E., 1999. Temporal workflow management in a claim handling system. ACM SIGSOFT Software Engineering Notes 24 (2), 187–195.