Constraint-based Planning and Scheduling Techniques for the Optimized Management of Business Processes

Irene Barba Rodríguez, 48861238-S

irenebr@us.es

Supervised by Dr. Carmelo Del Valle Sevillano

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)

Departamento de Lenguajes y Sistemas Informáticos
E.T.S. Ingeniería Informática. Universidad de Sevilla

Avda Reina Mercedes s/n. 41012 Sevilla
Teléfono +34 95 455 27 71 **Fax** +34 95 455 71 39
E-mail carmelo@us.es **Web** www.lsi.us.es

e.t.s.
ingeniería
informática

Dr. Carmelo Del Valle Sevillano, con DNI: 29775324F, profesor Titular de Universidad adscrito al departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla,

CERTIFICA QUE:

Dña. Irene Barba Rodríguez, con DNI: 48861238S, ha realizado bajo su supervisión el trabajo de investigación titulado:

CONSTRAINT-BASED PLANNING AND SCHEDULING TECHNIQUES FOR THE OPTIMIZED MANAGEMENT OF BUSINESS PROCESSES

Una vez revisado, autoriza la presentación del mismo como tesis doctoral en la Universidad de Sevilla y estima oportuna su presentación al tribunal para su valoración. Dicha tesis ha sido realizada dentro del programa de doctorado Tecnología e Ingeniería del Software, con mención de excelencia MEE2011-0129 del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla. Igualmente, autoriza la presentación para obtener la acreditación de Doctorado Internacional.

En Sevilla, a 14 de Diciembre de 2011

Informe Favorable.

Fdo. Carmelo Del Valle Sevillano

Agradecimientos

A través de estas líneas quiero expresar mi agradecimiento a todas aquellas personas que me han apoyado durante estos años.

En especial, agradecer al Dr. Carmelo Del Valle Sevillano, director de esta tesis, la orientación, el seguimiento y la supervisión continua de la misma, además de su paciencia y el apoyo recibido a lo largo de esta investigación.

Agradezco especialmente a la Dra. Barbara Weber la gran ayuda desinteresada recibida, el interés mostrado por mi trabajo, sus valiosas aportaciones a esta investigación, y los buenos consejos recibidos.

Quisiera hacer extensiva mi gratitud a mis compañeros y amigos del Departamento de Lenguajes y Sistemas Informáticos por tantas risas, confidencias, y momentos compartidos tanto dentro como fuera del trabajo. Un sincero agradecimiento a los miembros del grupo de investigación Quivir por su amistad, apoyo, ayuda en todo momento, y colaboración.

Un agradecimiento muy especial merece la comprensión, paciencia y el ánimo recibidos de mi familia y amigos, que me han acompañado tanto en los momentos de crisis como en los momentos de felicidad. El desarrollo de esta tesis nunca hubiera sido posible sin el amparo incondicional de mis padres, Manolo y Carmen, de mi hermano Dani y Josefi, y sin las continuas sonrisas de mis niños Rocío, Sara y Dani Jr. Gracias a Rafa que, de forma incondicional, entendió mis ausencias y mis malos momentos siempre con una sonrisa, gracias por hacer fácil lo difícil.

Muchas gracias a todos.

Acknowledgements

Through these lines I want to express my gratitude to all those who have supported me over these years.

Especially thank Dr. Carmelo Del Valle Sevillano, advisor of this thesis, for his guidance and continuous supervision of the thesis, as well as for his patience and support during this research.

I am especially grateful to Dr. Barbara Weber for her disinterested help, her interest in my work, her valuable contributions to this research, and her good advices.

I would like to extend my gratitude to my colleagues and friends of the Departamento de Lenguajes y Sistemas Informáticos for many laughter, confidences, and shared moments both inside and outside of work. A sincere thanks to the members of Quivir research group for their friendship, support, help at all time, and collaboration.

A special thanks to my family and friends for their understanding, patience and encouragement, who have accompanied me both in times of crisis and in times of happiness. The development of this thesis would never have been possible without the unconditional support of my parents, Manolo and Carmen, my brother Dani and Josefi, and without the constant smiles of my children Rocio, Sara and Dani Jr. Thanks to Rafa who, unconditionally, understood my absences and my bad times always with a smile, thanks for making the difficult easy.

Thank you very much everyone.

Resumen

Un proceso de negocio (business process, BP) se puede definir como un conjunto de actividades que se ejecutan de forma coordinada en un entorno organizativo y técnico, y que conjuntamente alcanzan un objetivo de negocio. Hoy en día, existe un interés creciente en la alineación de los sistemas de información de forma orientada a procesos, y por lo tanto se considera de vital importancia la gestión eficaz de los BPs (business process management, BPM). Una instancia de un proceso de negocio es análoga a un plan en inteligencia artificial (AI). Además, en BPM, un plan también debe incluir una asignación adecuada de recursos a las actividades del proceso (scheduling). Por lo tanto, existe un interés creciente en aplicar técnicas de planning y scheduling (P&S) para la mejora del ciclo de vida de BPM. Teniendo en cuenta que en general los problemas de P&S incluyen restricciones y la optimización de ciertas funciones objetivo, la programación con restricciones (constraint programming, CP) proporciona un framework adecuado para modelar y resolver este tipo de problemas. Además, existe bastante paralelismo entre CP y los lenguajes de modelado de BPs basados en restricciones. En la presente memoria de Tesis, se aplican técnicas de P&S basadas en restricciones en diferentes etapas del ciclo de vida de BPM de forma coordinada para mejorar así el proceso completo de gestión de los BPs.

Concretamente, en primer lugar, se propone la aplicación de técnicas de P&S a especificaciones de procesos de negocio declarativas para generar planes optimizados de ejecución de BPs. Dichos planes pueden ser utilizados para asistir a los usuarios durante diferentes etapas del ciclo de vida de BPM, de forma que ciertas funciones objetivos sean optimizadas. Estos planes de ejecución optimizados pueden ser utilizados en varias aplicaciones innovadoras, por ejemplo, (1) asistir a los usuarios durante la ejecución de BPs flexibles en la optimización de ciertas funciones objetivo mediante la generación de recomendaciones, ya que incrementar la flexibilidad típicamente implica decrementar la guía para el usuario, y por tanto la ejecución de BPs declarativos supone en general un reto significativo para los usuarios que lo ejecutan; y (2) generar automáticamente modelos de BPs optimizados, ya que la especificación imperativa y manual de los modelos de los BPs puede ser un problema muy complejo, consumir gran cantidad de recursos

temporales y humanos, causar algunos errores, y puede dar lugar a modelos no optimizados.

En segundo lugar, la presente memoria de Tesis incluye una propuesta para el modelado y la ejecución de BPs que conllevan la selección y el orden de las actividades a ejecutar (planning), además de la asignación adecuada de recursos (scheduling), considerando la optimización de varias funciones objetivo y el alcance de ciertos objetivos. La principal novedad es que todas las decisiones (incluso la selección de actividades) se toman en run-time considerando los valores reales de ejecución, y por lo tanto los BPs se gestionan de forma flexible y eficiente.

Abstract

A business process (BP) consists of a set of activities which are performed in coordination in an organizational and technical environment, and which jointly realize a business goal. Nowadays, there exists a growing interest in aligning information systems in a process-oriented way as well as in the effective management of BPs (Business Process Management, BPM). An instance of a BP is analogous to a plan in Artificial Intelligence (AI). In BPM, a plan also includes allocation of resources and target start and end times (scheduling). Therefore, the application of planning and scheduling (P&S) techniques to enhance the BPM life cycle has been analyzed in many research works in past years. Since P&S problems include constraints and certain objective functions need to be optimized, constraint programming (CP) supplies a suitable framework for modelling and solving these problems. Furthermore, several parallels between CP and constraint-based BP modelling languages exist. In the current Thesis Dissertation, constraint-based P&S techniques are applied at different stages of the BPM life cycle in a coordinated way to improve overall system functionality.

Specifically, in a first place, the application of P&S techniques to declarative process specifications to generate optimized BP enactment plans is proposed. These optimized plans can then be used in several interesting and innovative applications, e.g., (1) assisting users during flexible process execution to optimize performance goals of the processes through recommendations, since increasing flexibility typically implies decreased user guidance by the BPM and thus poses significant challenges to its users; and (2) automatic generation of optimized BP models, since the manual specification of imperative BP models can form a very complex problem, can consume a great quantity of time and human resources, may cause some failures, and may lead to non-optimized models.

Secondly, the current Thesis Dissertation presents a proposal for modelling and enacting BPs that involve the selection and the ordering of the activities to be executed (planning), besides the resource allocation (scheduling), considering the optimization of several objective functions and the reach of some goals. The main novelty of this proposal is that all decisions (even the activity selection) are taken

in run-time considering the actual parameters of the execution, and hence the BP is managed in an efficient and flexible way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Generalities

Nowadays, a growing interest in aligning information systems in a process-oriented way exists (Dumas et al., 2005; Weske, 2007) as well as in the effective management of business processes (BPs). A BP consists of a set of activities which are performed in coordination in an organizational and technical environment (Weske, 2007), and which jointly realize a business goal. Business Process Management (BPM) can be seen as supporting BPs using methods, techniques, and software to design, enact, control and analyze operational processes involving humans, organizations, applications, and other sources of information (van der Aalst et al., 2003). Similarly, Workflow Management Systems (van der Aalst and van Hee, 2002; Georgakopoulos et al., 1995) consist of methods and technologies for managing the flow of work in organizations. In a related way, BPM Systems (BPMSs) are software tools that support the management of the BPs.

Traditional BPM life cycle (Weske, 2007) includes several phases: (1) Process Design & Analysis, i.e., BPs are identified, reviewed, validated, and represented by business process models, (2) System Configuration, i.e., BPs are implemented by configuring a BPM system, (3) Process Enactment, i.e., the BPM system controls the execution of BP instances as defined in the BP model, and (4) Evaluation, i.e., information regarding the BP enactment is evaluated in order to identify and improve the quality of the BP model and their implementations.

An instance of a business process is analogous to a plan in AI. In AI planning (Ghallab et al., 2004), the activities to be executed are not established, i.e., before generating a plan. Therefore, it is necessary to select the activities to be executed from a set of alternatives and to establish an ordering.

Moreover, the execution of most BPs entails, in some way, scheduling decisions since the activities to be executed may compete for some shared resources.

In these cases, it is necessary to allocate the resources in a suitable way, usually optimizing some objectives. During process execution, scheduling decisions are typically made by the BPM systems (BPMSs), by automatically assigning activities to resources (Russell et al., 2005). The area of scheduling (Brucker and Knust, 2006; Pinedo, 2008) includes problems in which it is necessary to determine an enactment plan for a set of activities related by temporal constraints. Furthermore, the execution of every activity requires the use of resources, hence they may compete for limited resources.

Taking into account the parallels between Planning & Scheduling (P&S) and BPM, currently there exists a growing interest in the application of P&S techniques to enhance different stages of the BPM life cycle. However, from our point of view, several connections between both disciplines remain to be exploited. In the current Thesis Dissertation, P&S techniques are applied at different stages of the BPM life cycle in a coordinated way to improve overall system functionality.

Since a P&S problem includes constraints and the optimization of certain objective functions, constraint programming (CP) (Rossi et al., 2006) supplies a suitable framework for modelling and solving problems involving P&S aspects (Salido, 2010). Furthermore, a wide scope of BP constraint-based modelling languages are used and analyzed in many research works. Several parallels between CP and constraint-based BP modelling languages exist, and hence CP seems to be promising for modelling and solving problems related to BPM. In the current Thesis Dissertation, several constraint-based proposals are analyzed for modelling and solving P&S problems related to BPM.

1.2 Motivation and Contributions

Typically, business processes are specified in an imperative way, i.e., an activity sequence that will result in obtaining the related corporate goal is defined. However, declarative BP models are increasingly used and their usage allows the user to specify what has to be done instead of having to specify how it has to be done. Declarative BP model specifications facilitate the human work involved, avoid failures, and obtain a better optimization, since the tacit nature of human knowledge is often an obstacle to eliciting accurate process models (Ferreira and Ferreira, 2006).

The advantages of using declarative languages for BP modelling instead of imperative languages are discussed in several studies, e.g., (Wainer et al., 2004; Pesic et al., 2007; Rychkova et al., 2008a; Fahland et al., 2009, 2010; Pichler et al., 2011). Such advantages includes support for partial workflows (Wainer et al., 2004), absence of over-specification (Pesic et al., 2007), and provision of more maneuvering room for end users (Pesic et al., 2007).

Due to their flexible nature, frequently several ways to execute declarative process models exist, i.e., different enactment plans can be related to the same declarative BP model. Each one of these plans leads, in general, to get different values for several objective functions (e.g., overall completion time or cost) so that certain enactment plans are considered optimal regarding to some objective functions.

In this way, an AI-based method for **generating optimized BP enactment plans from declarative process specifications** (cf. Chapter 3) is proposed in order to optimize the performance of a process, according to objective functions like minimizing the overall completion time. For the generation of these optimized enactment plans, activities to be executed have to be selected and ordered (planning problem (Ghallab et al., 2004)) considering both control-flow and resource constraints (scheduling problem (Brucker and Knust, 2006; Pinedo, 2008)). For P&S the activities such that the process objective function is optimized, a constraint-based approach is proposed which is in charge of determining how it has to be done in order to satisfy the constraints imposed by the declarative problem specifications, and to attain an optimization of certain objective functions. The generation of optimized BP enactment plans from declarative process specifications can greatly improve the overall BPM life cycle (Weske, 2007), e.g., the optimized plans can be used for simulation (Rozinat et al., 2009), time prediction (van der Aalst et al., 2011), recommendations (Schonenberg et al., 2008; Haisjackl and Weber, 2010; Barba et al., 2011), and generation of optimized BP models (R-Moreno et al., 2007; Alves et al., 2008; González-Ferrer et al., 2009; Barba and Del Valle, 2011b), which are innovative and interesting topics to be addressed in BPM environments nowadays.

Specifically, these optimized BP enactment plans are used for **giving users of flexible BPMSs recommendations during run-time** (cf. Chapter 4) in the Process Enactment phase so that performance objective functions of processes are optimized. As mentioned, due to their flexible nature, frequently several ways to execute declarative process models exist. Typically, given a certain partial trace (reflecting the current state of the process instances), users can choose from several enabled activities which activity to execute next. This selection, however, can be quite challenging since performance goals of the process should be considered, and users often do not have an understanding of the overall process. Moreover, optimization of objective functions requires that resource capacities are considered. Therefore, recommendation support is needed during BP execution, especially for inexperienced users (van Dongen and van der Aalst, 2005). In our proposal, recommendations on possible next steps are generated taking the partial trace and the optimized plans into account. Furthermore, in the proposed approach, replanning is supported if actual traces deviate from the optimized enactment plans.

Other interesting application of the generated optimized BP enactment plans which is addressed in the current work is supporting process analysts in the BP Design & Analysis phase by automatically **generating optimized imperative BP models** (cf. Chapter 5). The BP Design & Analysis phase has the goal to generate a BP model, i.e., to define the set of activities and the execution constraints between them (Weske, 2007), by formalizing the informal BP description using a particular BP modelling notation. This phase plays an important role in the BPM life cycle, since it greatly influences the remaining phases of this cycle. Business process models are usually defined manually by business analysts through imperative languages considering activity properties, constraints imposed on the relations between the activities as well as different performance objectives. Furthermore, allocating resources is an additional challenge since scheduling may significantly impact business process performance. Therefore, the manual specification of process models can be very complex and time-consuming, potentially leading to non-optimized models or even errors can be generated. Moreover, the result of process modelling is typically a static plan of actions, which is difficult to adapt to changing procedures or to different business goals. To overcome these problems, the automatic generation of optimized imperative BP models from constraint-based specifications is proposed through creating optimized BP enactment plans (cf. Chapter 3). In this way, process models can be adapted to changing procedures or to different business goals, since imperative process models can dynamically be generated from static constraint-based specifications. Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way. Therefore, a wide study of several aspects can be carried out, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by starting from different declarative specifications.

On the other hand, the execution of most BPs entails, in some way, scheduling decisions since the activities to be executed may compete for some shared resources. In these cases, it is necessary to allocate the resources in a suitable way, usually optimizing some objectives. During process execution, scheduling decisions are typically made by the BPM systems (BPMSs), by automatically assigning activities to resources (Russell et al., 2005). To lesser measure, planning problems are present in BP executions when, in some points, several possible execution branches exist, and the selection of the suitable one depends on the BP goal and/or on the optimization of some functions. Since BP models are typically specified in an imperative way, most of the planning decisions are taken in the modelling phase. Specifically, the ordering and the selection of the activities to be executed (planning) in the BP enactment are specified in the BP design time, when only estimated values for several parameters can be analyzed. However, there are BPs which entail complex planning decisions which can greatly be in-

fluenced by the values of several unpredictable parameters, whose actual value is known in run time. In this way, a proposal for **modelling and enacting BPs that involve P&S decisions** (cf. Chapter 6) is presented. The main contribution is that both P&S decisions are taken in BP run-time, providing the process management with efficiency and flexibility, and avoiding the drawbacks of taking these decisions during the design phase. As an example, a complex and representative problem including P&S, the repair planning problem, is managed through the proposed approach. For solving this problem, a constraint-based approach is proposed. Moreover, a PDDL specification together with the results obtained by a generic planer are analyzed.

1.3 Structure

The rest of the document is organized as follows:

- Chapter 2 includes background related to the areas which are addressed in the current Thesis Dissertation, i.e., (1) business process management, (2) planning & scheduling, and (3) constraint programming. Moreover, the connections and parallels which are given between these areas are analyzed.

- Chapter 3 details the constraint-based approach which is used for P&S the BP activities so that optimized enactment plans are generated from constraint-based specifications.

- Chapter 4 includes how the generated optimized enactment plans are used to improve the Process Enactment phase by generating recommendations during run-time.

- Chapter 5 describes how the generated optimized BP enactment plans are used to enhance the Process Design & Analysis phase by automatically generating optimized imperative BP models.

- Chapter 6 details our proposal for modelling and enacting BP that involve planning and scheduling decisions in run-time so that Process Design & Analysis and Process Enactment phases are leveraged.

- Chapter 7 summarizes the main conclusions which were obtained during the development of this thesis.

- Lastly, Chapter 8 shows some future work which is intended to be addressed.

1.4 Publications

During the development of the current Thesis Dissertation, some research works have been published in different Workshops, Conferences and Journals. These publications[1] support the validation of the scientific quality of this thesis.

1. *Irene Barba, Barbara Weber, Carmelo Del Valle.* "Supporting the Optimized Execution of Business Processes through Recommendations". 7th International Workshop on Business Process Intelligence (BPI 2011, **Ranked as CORE C in ERA Conference Ranking**), **BPM 2011 Workshops, Part I, Springer LNBIP Vol. 99, pp. 135-140**, 2012.

2. *Irene Barba, Carmelo Del Valle.* "A Constraint-based Approach for Planning and Scheduling Repeated Activities". ICAPS 2011 **21st International Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2011, Ranked as CORE B in ERA Conference Ranking), pp. 55-62**, 2011.

3. *Irene Barba, Carmelo Del Valle.* "A Planning and Scheduling Perspective for Designing Business Processes from Declarative Specifications". **3rd International Conference on Agents and Artificial Intelligence (ISBN: 978-989-8425-40-9) (ICAART 2011, Ranked as CORE C in ERA Conference Ranking), Vol. 1, pp. 562-569**, 2011.

4. *Irene Barba, Carmelo Del Valle.* "Planning and Scheduling of Business Processes in Run-Time: A Repair Planning Example". 19th International Conference on Information Systems Development (ISD 2010, **Ranked as CORE A in ERA Conference Ranking**), **Information Systems Development, Business Systems and Services: Modeling and Development (ISBN: 978-1-4419-9645-9 e-ISBN: 978-1-4419-9790-6), Springer, pp. 75-88**, 2011.

5. *Carmelo Del Valle, Antonio Márquez, Irene Barba.* "A CSP Model for Simple Non-reversible and Parallel Repair Plans". **Journal of Intelligent Manufacturing (ISSN: 0956-5515 e-ISSN: 1572-8145, Impact Factor: 1.081 (15/38 Q2)), Vol. 21(1), pp. 165-174**, 2010.

6. *Irene Barba, Carmelo Del Valle.* "A Job-Shop Scheduling Model of Software Development Planning for Constraint-based Local Search". **International Journal of Software Engineering and Its Applications (ISSN: 1738-9984), Vol. 4(4), pp. 1-16**, 2010.

[1] The publications has been chronologically ordered starting from the most recent publications, and ending with the oldest publications.

7. *Irene Barba, Carmelo Del Valle.* "Una Propuesta PDDL para la Planificación de la Reparación como Proceso de Negocio". Apoyo a la Decisión en Ingeniería del Software (ADIS2010) **Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (ISSN: 1988-3455), Vol. 4(1), pp. 11-22**, 2010.

8. *Irene Barba, Carmelo Del Valle, Diana Borrego.* "A Multiobjective Constraint Optimization Model for Multimode Repair Plans". **Proceedings of the 6th International Conference on Informatics in Control, Automation and Robotics (ISBN: 978-989-8111-99-9) (ICINCO 2009), INSTICC Press, Vol. 1, pp. 355-358**, 2009.

9. *Irene Barba, Carmelo Del Valle, Diana Borrego.* "A Constraint-based Model for Multi-objective Repair Planning". **14th IEEE International Conference on Emerging Technologies and Factory Automation (ISBN: 978-1-4244-2728-4, ISSN: 1946-0759) (ETFA 2009), art. no. 5347038**, 2009.

10. *Diana Borrego, Rafael M. Gasca, María Teresa Gómez-López, Irene Barba.* "Choreography Analysis for Diagnosing Faulty Activities in Business-to-Business Collaboration". **20th International Workshop on Principles of Diagnosis (DX 2009), pp. 171-178**, 2009.

11. *Irene Barba, Carmelo Del Valle, Diana Borrego.* "PDDL Specification for Multi-objective Repair Planning". **CAEPIA 2009 Workshop on Planning, Scheduling and Constraint Satisfaction, Springer LNCS Vol. 1548, pp. 21-33**, 2009.

12. *Irene Barba, Carmelo Del Valle, Diana Borrego.* "A Constraint-based Job-Shop Scheduling Model for Software Development Planning". Apoyo a la Decisión en Ingeniería del Software (ADIS2010) **Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (ISSN 1988-3455), Vol. 3(1), pp. 1-12**, 2009.

13. *Irene Barba, Carmelo Del Valle, Diana Borrego.* "A Job-Shop Scheduling Model for Constraint-Based Local Search". **IBERAMIA 2008 Workshop on Planning, Scheduling and Constraint Satisfaction (ISBN: 972886207-5), pp. 7-18**, 2008.

14. *Diana Borrego, Rafael M. Gasca, María Teresa Gómez-López, Irene Barba.* "Diagnosing Business Processes Execution using Choreography Analysis". Apoyo a la Decisión en Ingeniería del Software (ADIS2010) **Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos (ISSN 1988-3455), Vol. 2, No. 3, pp. 13-24**, 2008.

15. *Irene Barba, Diana Borrego, Carmelo Del Valle, Rafael M. Gasca.* "Inferencia de Crónicas Temporales con Programación Lógica Inductiva para Predicción de Evoluciones". **XII Conferencia de la Asociación Española para la Inteligencia Artificial (ISBN: 978-84-611-8846-8) (CAEPIA 2007), Vol. 1, pp. 347-356**, 2007.

16. *Rafael M. Gasca, Carmelo Del Valle, Víctor Cejudo, Irene Barba.* "Improving the Computational Efficiency in Symmetrical Numeric Constraint Satisfaction Problems". **XI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA 2005), Springer LNAI Vol. 4177, pp. 269-279**, 2006.

1.5 Research Projects

The development of the current Thesis Dissertation has been framed in and funded by some research projects[2].

1. **Técnicas para la diagnosis, confiabilidad y optimización en los sistemas de gestión de procesos de negocio.** Ministerio de Ciencia e Innovación TIN2009-13714 *(01/01/2010 - 31/12/2012)*.

2. **OPBUS: Mejora de la calidad de procesos mediante tecnologías de optimización y tolerancia a fallos.** Consejería de Innovación, Ciencia y Empresa *(13/01/2009 - 12/01/2011)*.

3. **Automatización de la detección, diagnosis y tolerancia a fallos en sistemas con incertidumbre y en sistemas distribuidos.** Ministerio de Educación y Ciencia DPI2006-15476-C02-00 *(01/10/2006 - 30/09/2009)*.

4. **TELMEDIA: Monitorización y detección remota de desviaciones en terapias con técnicas inteligentes.** CITIC *(15/06/2005 - 30/05/2006)*.

[2]The research projects has been chronologically ordered starting from the most recent projects, and ending with the oldest projects.

Chapter 2

Background

The current Thesis Dissertation combines aspects of Planning and Scheduling (P&S) and Constraint Programming (CP) in order to improve different stages of the Business Process Management (BPM) life cycle. Section 2.1 provides backgrounds regarding BPM, Section 2.2 gives an overview of planning and scheduling, and Section 2.3 describes the constraint programming paradigm. Finally, Section 2.4 includes how P&S techniques can be applied in order to enhance different stages of a typical BPM life cycle, and summarizes the most related works.

2.1 Business Process Management

In the last years, the effective management of business processes (cf. Def. 1) in organizations became more important, since they need to adapt to the new commercial conditions, as well as to respond to competitive pressures, considering the business environment and the evaluation of their information systems. Moreover, in enterprizes an increasing interest in the management of businesses by using processes exists.

Definition 1. *A **Business Process** (BP) can be defined as a set of related structured activities whose execution produce a specific service or product required by a particular customer.*

In order to use and manage business processes, business analysts need to specify the BPs through BP models (cf. Def. 2) by using a BP modelling language. In the literature, a wide spectrum of paradigms for BP modelling are presented, each one entailing different levels of accuracy in the BP elicitation, e.g., declarative and imperative paradigms (cf. Sect. 2.1.2). Some examples of BP models are shown in Figs. 2.2, 2.3, which are explained later.

Definition 2. *A **business process model** consists of a set of activity models and execution constraints between them (Weske, 2007).*

The modelling of the processes plays an important role in the overall management of BPs (Business Process Management, BPM, cf. Def. 3) (Davenport, 1993; Georgakopoulos et al., 1995). In the current business world, the economic success of an enterprize increasingly depends on its effectiveness in the management of its BPs, and hence BPM is an interesting research area which is being widely analyzed nowadays. In a related way, BPM Systems (cf. Def. 4) are software tools that support the management of the BPs.

Definition 3. ***Business Process Management** (BPM) can be seen as supporting BPs using methods, techniques, and software to design, enact, control and analyze operational processes involving humans, organizations, applications, documents and other sources of information (van der Aalst et al., 2003).*

Definition 4. *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).*

Similarly to BPMSs, Workflow Management Systems (van der Aalst and van Hee, 2002; Georgakopoulos et al., 1995) consist of methods and technologies that allow managing the flow of work in organizations. In some cases, the software BP management tools use temporal information and ignore, in some ways, the resources to be used, considering them unlimited. This may not be adequate in different situations, for example when limited resources can be required by different activities at overlapped periods of time. In this way, resource allocation is only considered to a limited degree in existing BPMSs, and is typically done during run-time by assigning work to resources. In this Thesis Dissertation, resource perspective is also considered in the BP modelling step, since resource allocations and scheduling may significantly impact business process performance.

2.1.1 BPM Life Cycle

Traditional BPM Life Cycle (Weske, 2007) (cf. Fig. 2.1) includes several stages which are related to each other. These stages are organized in a cyclical structure which shows the logical dependencies between them:

- Process Design & Analysis: In this stage, BPs are identified, reviewed, validated, and represented by business process models (cf. Def. 2), so that the informal BP description is formalized using a particular BP modelling notation. Two steps are considered to create a BP model: (1) draw an initial

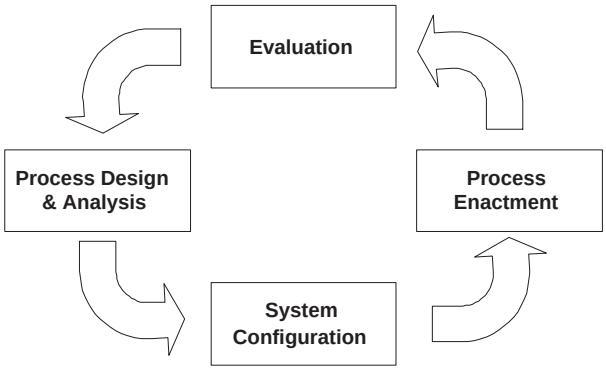


Figure 2.1: Typical BPM Life Cycle.

BP model, and (2) improve this initial model by simulation or BP redesign techniques. Traditionally, this phase is mostly a human activity. In some cases, process models can be verified against inconsistencies and errors (van Dongen, 2007).

- System Configuration: In this phase, BP models are implemented by configuring a BPM system. There are different ways to do so, e.g., by stating a set of policies and procedures. Service-oriented architectures as well as web services for their implementation have gained increasing popularity for BPMSs implementations recently. Moreover, data-driven approaches to the flexible enactment of BPs are considered for enactment of human interaction BPs using data dependencies to control process enactment.

- Process Enactment: After completing system configuration stage, BP instances can be enacted. In this stage, the BPM system controls the execution of BP instances as defined in the BP model. As execution proceeds, the enactment information must be analyzed due to the possible appearance of unexpected events.

- Evaluation: In this stage, information regarding the BP enactment is evaluated in order to identify and improve the quality of the BP model and their implementations. Traditionally, enactment logs are analyzed by using BP activity monitoring and BP mining techniques.

After the Evaluation phase, BP models are corrected and improved in the BP Design & Analysis phase if necessary by considering the evaluation information, and hence closing the cycle which shows the logical dependencies between the phases of the BPM life.

In this Thesis Dissertation, the BP Design & Analysis phase is widely analyzed since this phase plays an important role in the BPM life cycle for any improvement initiative, and it greatly influences the remaining phases of this cycle. The BP Design & Analysis phase has the goal to generate a BP model, i.e., to define the set of activities and the execution constraints between them (Weske, 2007), by formalizing the informal BP description using a particular BP modelling notation. Section 2.1.2 introduces the main paradigms which are used for BP modelling.

2.1.2 Process Modelling

In the literature, a wide spectrum of paradigms for BP modelling are presented. These different paradigms can be roughly categorized into the two following classes:

- The declarative BP paradigm focuses on what has to be done instead of having to specify how it has to be done. Declarative BP models capture the regulatory and internal directives of the business processes in order to restrict possible options of activity execution. Due to their flexible nature, frequently several ways to execute declarative process models exist. Different declarative approaches have been proposed, such as the use of constraints, rules, event conditions or other (logical) expressions (e.g., (Dourish et al., 1996; Joeris, 2000; Wainer and De Lima Bezerra, 2003; van der Aalst et al., 2009; Goedertier and Vanthienen, 2009)).

- The imperative BP paradigm focuses on defining a precise activity sequence which establishes how a given set of tasks has to be performed (e.g., (Zisman, 1977; Ellis and Nutt, 1993; BPMN, 2011)).

In our proposals, we consider declarative and imperative models. Regarding declarative models, we focus on constraint-based models, and regarding imperative languages, we consider BPMN.

Constraint-based BP Models

Recently, constraint-based approaches have received increased interest (Vanderfeesten et al., 2008; Pesic, 2008), since they suggest a fundamentally different way of describing BPs which seems to be promising in respect of the support of highly dynamic processes (Vanderfeesten et al., 2008; Pesic, 2008). Irrespective of the chosen approach, requirements imposed by the BPs need to be reflected by the process model. This means that desired behavior must be supported by the process model, while forbidden behavior must be prohibited (Pesic et al., 2007;

van der Aalst et al., 2009; Montali, 2009). While imperative process models specify exactly how things have to be done, declarative process models focus on what should be done.

In the literature, several rule-based and constraint-based languages for declarative BP modelling are proposed, among which the work (Glance et al., 1996) proposes BP grammars for the definition of rules in order to deal with activities and documents. In (Dourish et al., 1996), the prototype Freeflow is presented, which includes a constraint-based process modelling formalism, and the use of declarative dependency relationships. Moreover, (Wainer and De Lima Bezerra, 2003) presents a constraint-based proposal for rules which constrain the BP enactment through preconditions and postconditions of the activities, together with additional conditions which must hold in general. In a similar way, (Joeris, 2000) presents a flexible workflow enactment based on event-condition-action (ECA) rules. Furthermore, (Lu et al., 2006) proposes a temporal constraint network for BP execution in order to define selection and scheduling constraints through the use of temporal intervals.

In our proposal we use ConDec (van der Aalst and Pesic, 2006a) as a basis for the BP control-flow specification. ConDec is a graphical language based on Linear Temporal Logic (LTL) (Clarke Jr. et al., 1999) for modelling and enacting dynamic business processes. It uses an open set of templates, i.e., parameterized graphical representations of LTL formulas, for the definition of relationships between activities. These relationships related to the templates must be satisfied during the execution phase. Each template has a name, a graphical representation and a semantics given by a LTL formula. LTL (Clarke Jr. et al., 1999) is a widely used temporal logic for specifying temporal properties that includes four operators in addition to classical logical operators, i.e., *always*, *eventually*, *until* and *next time*. Besides ConDec (Pesic, 2008), some research works concerning BPs based on LTL can be found. As an example, (Liu et al., 2007) proposes a method for the automatic verification of BP models that are translated to finite state machines through compliance rules that are translated to LTL. Similarly, (Hallé and Villemaire, 2008) presents an algorithm for the runtime monitoring of BP properties, expressed in an extension of LTL. Moreover, (Awad et al., 2011) proposes an approach for synthesizing process templates out of compliance requirements expressed in LTL, and these templates are used as a basis for negotiation among business and compliance experts. Furthermore, (Elgammal et al., 2011) performs a comparative analysis between three languages (LTL, CTL and FCL) which can be used as the formal foundation of BP compliance requirements.

We consider ConDec to be a suitable language, since it allows the specification of BP activities together with the constraints which must be satisfied for correct BP enactment and for the goal to be achieved. Moreover, ConDec allows to specify a wide set of BP models of varied nature, flexibility and complexity in

a simple way. Furthermore, ConDec has been widely referenced in past years in the context of BPM (e.g., (Lamma et al., 2007; Ly et al., 2008; Montali, 2009; Chesani et al., 2009)). ConDec is based on constraint-based BP models (cf. Def. 5), i.e., including information about (1) activities that can be performed as well as (2) constraints prohibiting undesired process behavior.

Definition 5. *A **constraint-based process model** $S = (A, C_{BP})$ consists of a set of activities A[1], and a set of constraints C_{BP} prohibiting undesired execution behavior. For each activity $a \in A$, resource constraints can be specified by associating a role with that activity. The activities of a constraint-based process model can be executed arbitrarily often if not restricted by any constraints.*

Constraints can be added to a ConDec model to specify forbidden behavior, restricting the desired behavior (for a description of the complete set of constraints, cf. (van der Aalst and Pesic, 2006a)). ConDec basic templates can be divided into 3 groups:

1. **Existence** constraints: unary relationships concerning the number of times one activity is executed. As an example, Exactly(N, A) specifies that A must be executed exactly N times.

2. **Relation** constraints: positive binary relationships used to establish what should be executed. As an example, Precedence(A, B) specifies that to execute activity B, activity A needs to be executed before.

3. **Negation** constraints: negative binary relationships used to forbid the execution of activities in specific situations. As an example, NotCoexistence(A, B) specifies that if B is executed, then A cannot be executed, and vice versa.

An interesting extension which has been considered for ConDec templates is the inclusion of **Choice** constraints (Pesic, 2008), which are n-ary constraints expressing the need of executing some activities belonging to a set of possible choices, independently of the other constraints.

In ConDec, while unary relationships describe constraints related to one activity (e.g., existence constraints), binary constraints describe relationships between activities (e.g., precedence constraints). Binary templates are composed by a source activity (cf. Def. 6) and a sink activity (cf. Def. 7), which correspond to the beginning and the end of the arrow related to the specific template in the graphical notation of ConDec, respectively.

[1] In this Thesis Dissertation, the BP activities are considered to be primitive, i.e., they are not composed by other BP activities.

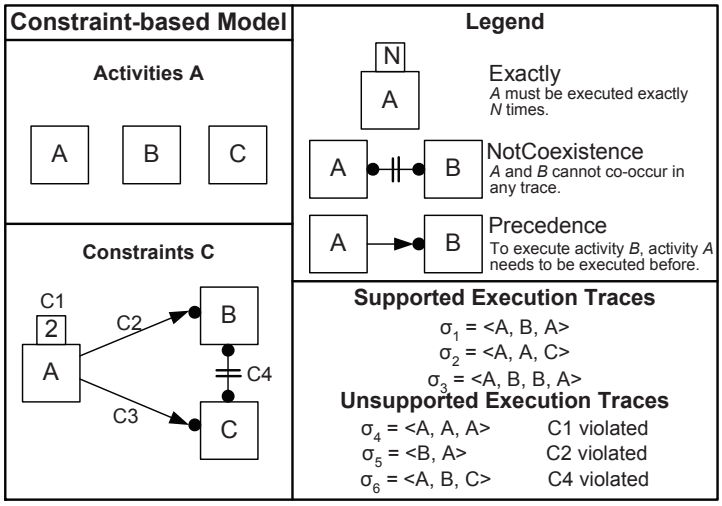


Figure 2.2: Simple Constraint-based Model.

Definition 6. *A source activity of a binary template is an activity which appears in the first parameter of the template. For templates which state precedence relations between activities, a source activity is a predecessor activity.*

Definition 7. *A sink activity of a binary template is an activity which appears in the second parameter of the template. For templates which state precedence relations between activities, a sink activity is a successor activity.*

Figure 2.2 shows a simple constraint-based model which is composed by activities *A*, *B*, and *C*, and constraints *C*1 (Exactly(N,A)), *C*2 (Precedence(A, B)), *C*3 (Precedence(A, C)), and *C*4 (NotCoexistence(B, C)).

In ConDec, binary constraints can be extended by defining branched templates, as described in (Pesic, 2008). The branched templates for the binary templates can be established between several BP activities in the following way:

- The branched constraint is established between several source activities and one sink activity, so that the relation is given between *at least* one of the sources and the sink[2].
- The branched constraint is established between one source activity and several sink activities, so that the relation is given between the source and *at least* one of the sinks[3].

[2]These branched templates consider only the disjunction of conditions related to the sources, since the conjunction can be obtained by including the associated non-branched template between each source and the sink activity.

[3]These branched templates consider only the disjunction of conditions related to the sinks, since the conjunction can be obtained by including the associated non-branched template between the source and each sink activity.

In ConDec (van der Aalst and Pesic, 2006a), the fact of considering atomic activities is recognized as being a major problem. Similar to ConDec, the languages ConDec++ (Montali, 2009) (an extension of ConDec) and Saturn (Demeyer et al., 2010) are constraint-based workflow definition languages based on LTL which, unlike ConDec, consider non-atomic activities that can be started, completed or cancelled at a later time, and overlapped with other activities.

Once a BP is modelled through a constraint-based modelling language, the BP can be executed. As the execution of a BP model proceeds, information regarding the executed activities can be recorded in an execution trace (cf. Def. 8). Information related to past process execution can be very valuable, since it can be used for many purposes, e.g., process mining (van der Aalst et al., 2011) or generating estimates (cf. Chapter 3).

Definition 8. *Let $S = (A, C_{BP})$ be a constraint-based process model with activity set A and constraint set C_{BP}. Then: A **trace** σ is composed by a sequence of starting and completing events $< e_1, e_2, ...e_n >$ regarding activity executions a_i, $a \in A$, i.e., events can be:*

1. *$start(a_i, r_{jk}, t)$, i.e., the i-th execution of activity a using k-th resource with role j is started at time t.*

2. *$comp(a_i, t)$, i.e., the i-th execution of activity a is completed at time t.*

Related to the process execution trace, is the concept of process instance. Specifically, a process instance (cf. Def. 9) represents a concrete execution of a constraint-based model and its execution state is reflected by the execution trace.

Definition 9. *Let $S = (A, C)$ be a constraint-based process model with activity set A and constraint set C. Then: A **process instance** $I = (S, \sigma)$ on S is defined by S and a corresponding trace σ.*

A running process instance I is in state **satisfied** if its current partial trace σ satisfies all constraints stated in C. Furthermore, an instance is in state **violated**, if the partial trace violates any constraint stated in C and there is no suffix that can be added to satisfy them. Figure 2.2 includes examples of traces of satisfied and violated instances[4] for a constraint-based model.

During the BP enactment, considering a constraint-based model and a specific related process instance, only certain activities are enabled to be executed next (cf. Def. 10). This selection, however, can be quite challenging since performance goals of the process (e.g., minimization of overall completion time)

[4]For the sake of clarity, only completed events for activity executions are included in the trace representation.

should be considered, and users often do not have an understanding of the overall process. Moreover, optimization of performance goals requires that resource capacities are considered. Therefore, recommendation support is needed during BP execution, especially for inexperienced users (van Dongen and van der Aalst, 2005) (cf. Chapter 4).

Definition 10. *Let $S = (A,C)$ be a constraint-based process model with activity set A and constraint set C, and $I = (S,\sigma)$ be a corresponding process instance with partial trace σ. Then: An activity a_i of instance I is **enabled** at time T iff a_i can be started and the instance state of I is not violated afterwards; i.e., for $\sigma =< e_1, e_2, ...e_n >$, we obtain $\sigma'_l =< e_1, e_2, ...e_n, start(a_i, Rj_k, T) >$ afterwards and instance (S,σ') is not in state violated.*

For example, for the partial trace σ_1 of Fig. 2.2, B is enabled, while A is not enabled due to $C1$, and C is not enabled due to $C4$.

Due to the flexible nature of constraint-based models, frequently several ways to execute constraint-based process models exist. Therefore, there are different ways to executed a constraint-based process model in such a way that all constraints are fulfilled, i.e., the process goal is reached (cf. Def. 11). For example, traces[5] $< AAB >$ and $< AAC >$ are two valid ways of executing the constraint-based model of Fig. 2.2, while trace $< AABC >$ is invalid due to $C4$.

Definition 11. *The **goal** of a BP is specified through the constraints which must be satisfied in the BP enactment.*

The different valid execution alternatives, however, can vary greatly in respect to their quality, i.e., how well different performance objective functions (cf. Def. 12) like minimizing cycle time can be achieved.

Definition 12. *The **objective function** of a BP is the function to be optimized during the BP enactment.*

An objective function which is considered in this Thesis Dissertation is the overall completion time of processes, i.e., time needed to complete all process instances which were planned for a certain period.

Business Process Model and Notation

The Business Process Model and Notation (BPMN) (BPMN, 2011) is a standard for modelling BP flows and web services, and provides a graphical notation for

[5]For the sake of clarity, traces represent sequences of activities so that no parallelism is considered in the examples, i.e., only completed events for activity executions are included in the trace representation.

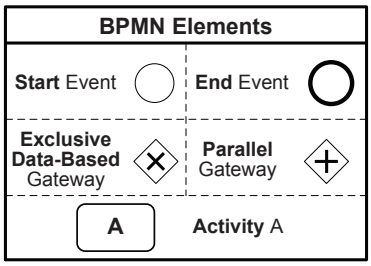


Figure 2.3: Some BPMN elements.

the specification of BP models. A BPMN model is composed of events, gateways, activities and swim lanes, among other elements (cf. Fig. 2.3). An event represents something that happens during the enactment of a BP and affects its execution flow, specifically the start event initiates the flow of the process, while the end event finishes this flow. Gateways are in charge of controlling how sequence flows interact as they converge or diverge within a process. Specifically, the *exclusive data-based gateway* can either be used as a decision point where several outgoing sequence flows are possible but only one sequence will be selected for the execution, or as a way to merge several sequence flows into one; while the *parallel gateway* provides a mechanism to both fork and synchronize the flows. Swim lanes are graphical ways of organizing and categorizing the BP activities, whereby pools represent the participants in a BP, and lanes are used to organize the activities within a pool according to roles and resources.

Figure 2.4 shows a toy example of a BPMN model, which contains three lanes, S, BM1 and BM2. After starting the enactment, the activity Receive Request is executed using the resource *S*. After that, two activities are executed in parallel (parallel gateway): (1) activity Hotel Search using the resource BM1, and (2) activity Airline Search using the resource BM2. After executing Hotel Search activity, only one of the following activities is executed (exclusive data-based gateway): (1) activity Failed Hotel using resource BM1, or (2) activity Book Hotel using resource BM1. In a similar way, after executing Airline Search activity, only one of the following activities is executed (exclusive data-based gateway): (1) activity Failed Airline using resource BM2, or (2) activity Book Airline using resource BM2. Moreover, after Failed Airline activity, only one of the following flows is given (exclusive data-based gateway): (1) activities Compensation and Notify Failure are executed using resource S, or (2) the BP enactment finishes. Furthermore, if at least one of the activities Book Hotel or Book Airline is executed (exclusive data-based gateway), activities Credit Card and Notify Booked are then executed. After that, the BP enactment finishes.

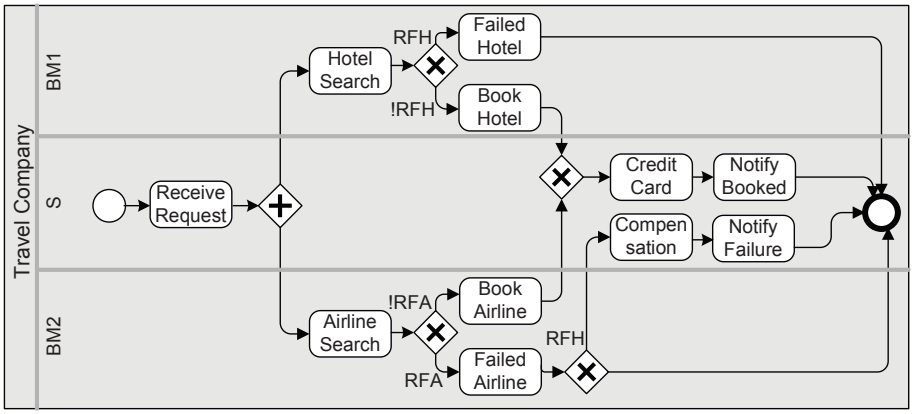


Figure 2.4: Example of BPMN model.

2.2 Planning & Scheduling

Planning (cf. Sect. 2.2.2) and scheduling (cf. Sect. 2.2.1) are two rather related areas, and hence many actual problems involve both of them (cf. Sect. 2.2.3). However, these areas also present some differences. Both the similarities and the main differences are discussed in the current section.

2.2.1 Scheduling

The area of scheduling (Brucker and Knust, 2006; Pinedo, 2008) includes problems in which it is necessary to determine an enactment plan for a set of known activities related by temporal constraints. Moreover, the execution of every activity requires the use of resources, hence they may compete for limited resources. In general, the goal in scheduling is to find a feasible plan which satisfies both temporal and resource constraints. Resource constraints lead to establish a specific ordering between activities which share the same resource, providing the problem with NP-hard complexity (Garey and Johnson, 1979). Several objective functions are usually considered to be optimized, in most cases related to temporal measures (e.g., minimization of completion time), or considering the optimal use of resources.

In scheduling, an activity refers to a task which needs to be executed during a specific amount of time units, usually without interruption (i.e., preemptive scheduling), and using certain specific resources.

A quite general scheduling problem is called Resource-Constrained Project Scheduling Problem (RCPSP, cf. (Brucker and Knust, 2006)). RCPSPs are specified by a set of activities which are related by precedence constraints[6]. Moreover, for the execution of each activity, several units of many resources may be required.

[6]"Activity a precedes activity b" means that activity b cannot start before a is finished

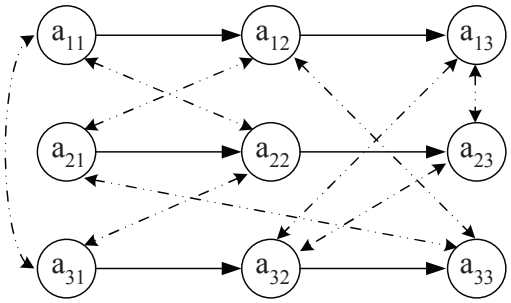


Figure 2.5: A disjunctive graph for a job shop problem.

An extension of RCPSPs is the Multi-mode Resource-Constrained Project Scheduling Problem (cf. (Drexl and Gruenewald, 1993)). This problem is a RCPSP in which the activities can be executed in more than one operating mode, each one potentially using different resources, and usually presenting different values for certain properties, e.g., duration or cost of the activity.

In many scheduling problems, the activities are organized in jobs, i.e., sequence of activities which establishes precedence relations between the activities so that an activity can start only when all its predecessors have been executed.

Many variants of scheduling problems exist. Some of them are listed as follows:

- Job Shop (cf. (Brucker and Knust, 2006; Pinedo, 2008)): Each activity can only be executed using a specific resource.

- Flow Shop (cf. (Brucker and Knust, 2006)): It is a special case of the job shop problem in which each job is composed by exactly the same number of activities (which is equal to the number of resources). In this way, each job contains exactly one activity to be executed using each resource, and hence each job uses each resource exactly once. Moreover, all jobs use the resources in the same ordering.

- Flexible Job Shop (cf. (Brandimarte, 1993): Many job centers exist, each one containing the same number of resources. In this way, an activity can be executed in any job center using the suitable resource).

- Cumulative Job Shop (cf. (Nuijten and Aarts, 1996): It is a generalization of the Job Shop in which the resources have a finite capacity and the activities may require several unities of several kinds of resources).

- Open Shop (cf. (Pinedo, 2008)): Unlike in job shop problems, in open shop problems the jobs do not have a predetermined fixed route.

Typically, the so-called disjunctive graph (Blazewic et al., 2000) is used to represent schedules for the job shop problem (cf. Fig. 2.5). A disjunctive graph $G = (V, C, D)$ is composed by:

- A set V of nodes, each one representing an activity.
- A set of edges which link the nodes. Two kinds of edges can be distinguished:
  - Precedence edges C (conjunctions), which correspond to the precedence constraints. They are directed arcs which link activities which are included in the same job.
  - Resource edges D (disjunctions), which correspond to the resource constraints. They are non-directed arcs which link the activities which are executed using the same resource.

In this way, a solution to the problem consists of establishing a direction for the undirected arcs, being feasible if there are no cycles. As an example, Fig. 2.5[7] shows the disjunctive graph representation for a simple job shop scheduling problem which includes 3 jobs, each one containing 3 activities.

There are many typical objective functions to be considered in scheduling problems. Some of them are listed as follows:

- Makespan: It refers to the time in which the execution of all activities have finished.
- Tardiness: It refers to the delay of all jobs or activities regarding a specific due date.
- Total Weighted Tardiness: It consists of a generalization of the tardiness, in which $\sum_{j \in Jobs} w_j \times T_j$ is minimized, where w_j usually refers to an importance factor related to job j, e.g., holding cost per unit time, and T_j refers to the delay of job j regarding a specific due date.
- Number of Tardy Jobs: It refers to the number of jobs which do not meet their due dates.
- Total Weighted Completion Time: It consists on minimizing $\sum_{j \in Jobs} w_j \times C_j$, where w_j usually refers to an importance factor related to job j, and C_j refers to the completion time of job j.
- Objectives related to the use of the resources by the activities, e.g., balanced use of resources.

[7] a_{ij} refers to the j-th activity of the job i.

2.2.2 Planning

In a wider perspective, in Artificial Intelligence (AI) planning (Ghallab et al., 2004), the activities to be executed are not established a priori, hence it is necessary to select them from a set of alternatives and to establish an ordering. In planning problems, usually the optimization of certain objective functions is considered.

Taking the goals to be achieved into account, different planning strategies can be used for representing and reasoning about planning scenarios, e.g., Classical Planning (Fikes and Nilsson, 1971; Lekavy and Návrat, 2007), Hierarchical Task Network (HTN) (Erol et al., 1994), Decision-Theoretic Planning (Joshi et al., 2011), Case-based Planning (Hammond, 1990) and Reactive Planning (Fernandes et al., 1983).

In order to reuse the same algorithms for solving different kinds of problems, and to solve the same problem using different algorithms, (1) domains for representing the problems, and (2) algorithms for solving the problems are specified in a separated way (domain-independent planning). For solving a specific problem, a domain-independent planner takes as input the problem specification and the domain information.

The first strategy which was proposed for representing and reasoning about planning scenarios was Classical Planning (Fikes and Nilsson, 1971; Lekavy and Návrat, 2007). The basic idea of Classical Planning consists of finding a sequence of actions which will modify the initial state of the world into a final state where the goal holds. The specification of Classical Planning problems is composed by:

- A set of literals from the propositional calculus which can be positive or negative and which represent the goal to be achieved.
- A set of literals from the propositional calculus which can be positive or negative and which represent the initial state, also known as initial conditions.
- A set of actions which are characterized by STRIPS operators. A STRIPS operator is a parameterized template used for stating a set of possible actions. Each action is composed by:
  - A set of preconditions: set of positive or negative literals which must be true for executing a specific action.
  - A set of effects: set of positive or negative literals which become true after the execution of the action.

As mentioned before, for executing an activity, all the literals included in the precondition of the activity need to be true. Therefore, at each stage, there is a set

of possible activities to be executed which depends on the literals which are true in that moment (state of the world, i.e., a set of atoms or literals that define how the objects of the model relate to each other and their properties). Each time a specific activity is executed, the set of literals which are true changes, and hence the set of possible activities to be executed also changes. In this way, the state of the world evolves.

A solution for a planning problem is determined by a sequence of activities which reaches the final state from the initial state.

Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) (Ghallab and et al., 1998) is a language for the definition of classical planning domains and problems which is used by the planning community. Specification in PDDL includes two separated items: a domain file for predicates and actions, and a problem file for objects, initial state and goal specification. Moreover, PDDL supports several characteristics, such as basic STRIPS-style actions, conditional effects, universal quantification over dynamic universes, specification of safety constraints, etc.

PDDL 2.2 (Hoffmann and Edelkamp, 2005), an extension of PDDL, includes more characteristics: it allows handling of numeric values, the actions can have a duration, and it is capable to deal with plan objective functions, derived predicates and timed initial literals.

In 2006, PDDL 3.0 (Gerevini and Long, 2006) was developed as an extension of PDDL, by allowing the user to express strong and soft constraints about the structure of the desired plans, as well as strong and soft problem goals.

The definition of the **domain** for a PDDL specification contains different items:

- Predicates: Outstanding properties of objects; can be true or false.
- Functions (Fluents): Allow handling of numeric values. They are used in actions preconditions or effects and their values are given in the problem file.
- (Durative) Actions/Operators: Ways of changing the state of the world.

The PDDL **problem** defines the next items:

- Objects: Outstanding things in the world.
- Initial state: Initial state of the world.
- Goal specification: Things that must be true.

- Objective function: Plan quality measures (metrics).

Once a problem is modelled through PDDL, a generic or specialized planner can be used to obtain the required solution, e.g., UCPOP (Weld, 1994), Graphplan (Blum and Furst, 1997), SHOP2 (Nau et al., 2003), VHPOP (Younes and Simmons, 2003) or SGPlan (Chen et al., 2006).

2.2.3 Integrating P&S

Planning and scheduling are rather related areas since both deal with the temporal planning of activities. The main difference between both areas is that in scheduling the activities to be planned are known and that it always involves the resource perspective, while in planning the activities which will be included in the plan need to be determined and resource constraints are not always considered.

Many works which combined P&S can be found, since several actual problems involve both of them. A problem involving P&S is characterized by: (1) there is a goal to be reached through the execution of a sequence of activities which are unknown a priori (planning), and (2) each of these activities has a specific duration and requires a specific resource to be executed, so that temporal constraints exist between the execution of activities, and certain (temporal) objective function needs to be optimized (scheduling).

Some of the extensions to scheduling that have been considered, such as alternative resources and process alternatives, lead to models that are closer to planning, as problems involving choice of actions are often regarded as planning problems (Smith et al., 2000).

Some planning techniques are not able to represent or reason with resources, metric quantities or continuous time. Moreover, planning techniques do not typically consider optimization. Therefore, there are many works which extend classical planning techniques in order to deal with resources (Drabble and Tate, 1994; Laborie and Ghallab, 1995), metric quantities (Koehler, 1998; Penberthy and Weld, 1994), and optimization criterions (Wolfman and Weld, 1999; Vossen et al., 1999). Furthermore, there exist works which extend planning techniques in order to allow working with continuous time and temporal constraints (Penberthy and Weld, 1994; Smith and Weld, 1999).

2.3 Constraint Programming

Constraint Programming (CP, cf. Fig. 2.6) is a software technology which is used for modelling and solving a wide scope of problems of varied nature which pursue

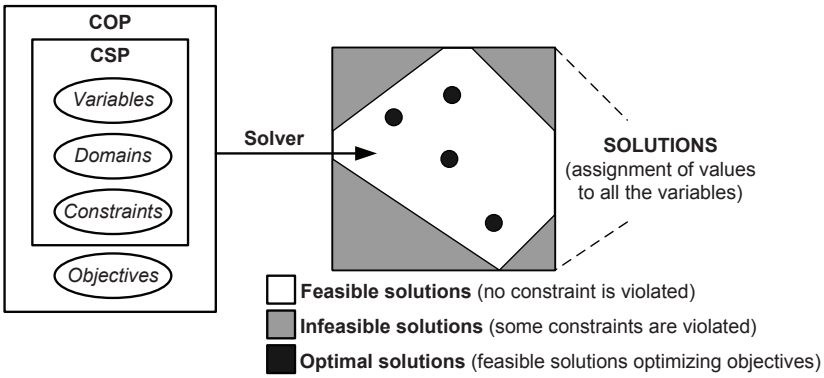


Figure 2.6: Constraint Programming.

different goals (Dechter, 2003; Rossi et al., 2006). CP can be used, among others, for planning and scheduling purposes (Salido, 2010).

In order to solve a problem through constraint programming, the process is divided into two steps:

1. Modelling the problem as a Constraint Satisfaction Problem, CSP (cf. Sect. 2.3.1), i.e., defining variables, domains for the variables, and constraints which relate the variables.

2. Solving the CSP (cf. Sect. 2.3.2). This step can be developed through many different techniques, e.g., search algorithms, consistency techniques, and hybrid techniques.

In constraint programming, dissociating the modelling of the problem from the solving technique is desirable, so that the problem is completely specified through a model which can be solved by using a wide scope of different techniques.

2.3.1 Constraint Satisfaction Problems

In order to solve a problem through constraint programming, it needs to be modelled as a constraint satisfaction problem (CSP) (cf. Def. 13).

Definition 13. *A **CSP** $P = (V, D, C_{CSP})$[8] is composed by a set of variables V, a domain of values D for each variable $var_i \in V$, and a set of constraints C_{CSP} between variables, so that each constraint represents a relation between a subset of variables and specifies the allowed combinations of values for these variables.*

[8] C_{CSP} and C_{BP} are used for relating a set of constraints of a CSP and of a constraint-based BP model, respectively.

In general, the same problem can be modelled in different ways. The selection of a specific model is essential since, in most cases, it greatly influences the strategy which must be followed in the search process as well as the execution time which is needed for finding the required solution.

A solution for a CSP (cf. Def. 14) consists of assigning values to all the CSP variables.

Definition 14. *A **solution** $S =< (var_1, val_1), (var_2, val_2), ...(var_n, val_n) >$ for a CSP $P = (V, D, R)$ is an assignment of a value val_i to each variable $var_i \in V$. A solution is **partial** if there exist one or more CSP variables which are not instantiated. A solution is **feasible** when the assignments variable-value satisfy all the constraints.*

In a similar way, a CSP is feasible if there exists at least one feasible solution for this CSP. From now on, S^{var} refers to the value assigned to variable *var* in the (partial) solution S.

Similar to CSPs, in constraint optimization problems (COPs, cf. Def. 15), a solution which meets the constraints and also optimize a specific objective function is pursued. The complexity of solving a satisfiability problem is, in general, NP-complete, while in the case of optimization problems is usually NP-hard.

Definition 15. *A **COP** $P_O = (V, D, R, o)$ is a CSP which also includes an objective function o to be optimized.*

A feasible solution S for a COP is **optimal** when no other feasible solution exists with a better value for the variable related to o (var_o). Many problems can involve multiple conflicting objectives that should be considered at the same time (multi-objective optimization problems).

Once a problem is modelled as a CSP, several goals can be pursued, e.g.:

- Finding any feasible solution for the CSP.
- Finding several feasible solutions for the CSP.
- Finding all the feasible solutions for the CSP.
- Finding the optimal (or optimized) solution for a COP.
- Finding a set of optimal (or optimized) solutions for a COP.

A classic problem which can be modelled as a CSP is the map coloring problem. This problem consists of coloring a map which is divided in a set of regions so that a color need to be assigned to each region, taking into account that regions sharing a boundary line do not have the same color and only specific colors can

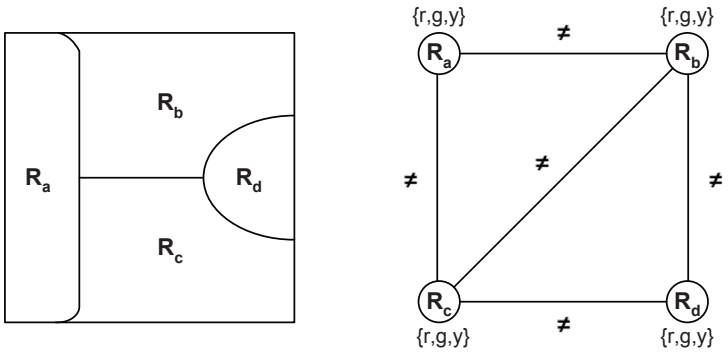


Figure 2.7: Map coloring problem.

be used. The modelling of this problem as a CSP is made so that each region is a CSP variable, the domain of each variable is composed by the set of allowed colors, and the constraints establish inequality relations between the variables which represent adjoining regions. Figure 2.7 shows an example for this problem in which there are 4 regions, R_a, R_b, R_c and R_d, and 3 allowed colors, red (r), green (g) and yellow (y).

2.3.2 Solving the CSP

As mentioned before, constraint programming allows to separate the models from the search algorithms, so that once a problem is modelled in a declarative way as a CSP, a generic or specialized constraint-based solver can be used to obtain the required solution. In this way, the same solving methods can be used for different problems. Several mechanisms are available for the solution of CSPs and COPs, which can be classified as search algorithms, consistency techniques and hybrid techniques.

Search Algorithms

Search algorithms are based on the exploration of the solution space, i.e., set of all the possible assignments of values to variables, until a solution is reached or that none exists is proved.

There are several ways to classify the search algorithms. One of the most used is classifying the search algorithms in complete and incomplete search.

Complete Search Complete search algorithms (which are also called systematic algorithms) guarantee that a solution will be found if one exists, and can be used to show that a CSP does not have a solution and to find a optimal solution in COPs.

The possible combinations of assignments of values to the CSP variables lead to a space state which can be represented by a tree o search graph. Each node

of the search tree represents a partial assignment of values to a set of variables. The root node of the search tree represents the case in which any variable is instantiated, while the leaf nodes represent the cases in which all the variables are instantiated.

There are many systematic search algorithms, most of them are based on chronological backtracking (Mouhoub et al., 2003).

Incomplete Search Incomplete search algorithms consist of exploring only certain regions of the state space so that, in general, the reach of a (optimal) solution can not be guaranteed. They are widely used due to the complete search usually requires a high cost. Local or stochastic search algorithms are examples of incomplete algorithms.

Local search algorithms typically start generating a first solution of a given problem instance (in a random or a heuristic way), which may be infeasible, suboptimal or incomplete. This initial solution is iteratively improved so that the value for the objective function is optimized in the case of COPs, or the number of inconsistencies are reduced in the case of CSPs. In most cases, these algorithms finish after certain tries or iterations have been completed, or when the required solution is found.

A wide scope of local search algorithms can be found in the literature, e.g., genetic algorithms (Mitchell, 1998), simulated annealing (Kirkpatrick et al., 1983), taboo search (Glover, 1989), and Greedy Randomized Adaptive Search Procedure (GRASP) (Feo and Resende, 1989, 1995). Different local search algorithms vary in the way in which improvements are achieved, and in particular, in the way in which situations are handled when no direct improvement is possible.

Moreover, some algorithms combine systematic and local search techniques, e.g, Large Neighborhood Search (LNS) (Pisinger and Ropke, 2010).

Consistency Techniques

They are also called constraint propagation techniques, and refer to any reasoning which consists in explicitly forbidding values or combinations of values for some variables of a problem because a given subset of its constraints cannot be satisfied otherwise (Rossi et al., 2006). In a related way, the inference process consists of evolving from a problem P to an equivalent problem P', i.e., exactly having the same set of feasible solutions, which presents a smaller solution space, and hence, which is easier to solve.

Concepts related to different grades of consistency (cf. Defs. 16, 17, 18, and 19) typically have a great relevancy in consistency techniques.

Definition 16. *A CSP* $= (V, D, C_{CSP})$ *presents* ***node consistency*** *iff every unary constraint* $\in C_{CSP}$ *on a variable is satisfied by all values in the domain of the variable, and vice versa.*

Definition 17. *A CSP* $= (V, D, C_{CSP})$ *presents* ***arc consistency*** *iff for all pairs of CSP variables* $(var_1, var_2)|var_1, var_2 \in V$*, for each value of* var_1 *in the domain of* var_1 *there is some value in the domain of* var_2 *that satisfy all the constraints stated in* C_{CSP} *between* var_1 *and* var_2*, and vice versa.*

Definition 18. *A CSP* $= (V, D, C_{CSP})$ *presents* ***path consistency*** *iff all pairs of CSP variables* $(var_1, var_2)|var_1, var_2 \in V$ *present path consistency with all the other variables* $\in V$*. A pair of variables* var_1 *and* var_2 *presents path consistency with a third variable* var_3 *iff for every pair of values* (val_1, val_2) *that satisfies the binary constraint between* var_1 *and* var_2*, there exists a value* val_3 *in the domain of* var_3 *such that* (val_1, val_3) *and* (val_2, val_3) *satisfy the constraint between* var_1 *and* var_3 *and between* var_2 *and* var_3*, respectively.*

Definition 19. *A CSP* $= (V, D, C_{CSP})$ *presents* ***i-consistency*** *iff all tuples of i-1 variables present i-consistency with all the other variables. A tuple of i-1 variables is i-consistent with another variable if every consistent evaluation of the i-1 variables can be extended with a value of the other variable while preserving consistency. In a related way, strong i-consistency is given when all* $j \leq i$ *present j-consistency.*

In this way, 2-consistency coincides with arc consistency if the problem is node-consistent, while 3-consistency coincides with path consistency only if all constraints are binary.

Many algorithms has been proposed in literature for establishing arc consistency in a CSP, e.g., the pioneers $AC1$, $AC2$ and $AC3$ (Mackworth, 1977) or the recent $AC3^{bit}$ (Lecoutre and Vion, 2008).

Consistency techniques can be complete or incomplete, as explained as follows.

Complete Inference The inference is called complete when after performing the inference process, the problem has a direct solution, e.g., adaptive consistency algorithm (Larrosa and Meseguer, 2003).

Incomplete Inference The inference is called incomplete when after performing the inference process, the problem does not have a direct solution, and hence a search for finding a solution needs to be performed, e.g., arc consistency.

To improve the modelling of the problems and to efficiently handle the constraints in the search for solutions, constraint-based proposals may include **global**

constraints implemented through **filtering rules** or propagators (i.e., responsible for removing values which do not belong to any solution). These global constraints facilitate the specification of the problem at the same time as the related filtering rules enable the efficiency in the search for solutions to increase since during search process these filtering rules remove inconsistent values from the domains of the variables.

Hybrid Techniques

Hybrid techniques are based on the search of solutions through the combination of search algorithms + consistency techniques, so that the best aspects from both techniques try to be combined in order to get a good solving mechanism.

Hybrid techniques can be classified in:

- Combination of systematic search and incomplete inference: In each node of the search subtree, the local consistency of the subproblem which is represented by that node is dealt with, so that the inconsistent partial tuples are detected. These inconsistent tuples are removed, and hence the state space is reduced and even inconsistency can be detected if the domain becomes empty. Examples of these techniques are conflict-directed backjumping (Prosser, 1993), learning (Frost and Dechter, 1994), and maintaining arc consistency (MAC) (Sabin and Freuder, 1994).

- Combination of systematic search and complete inference: In general, complete inference is very costly due to the high computational effort which it requires. However, taking into account the value of some parameters, there are certain situations in which the application of complete inference is suitable combined with a search algorithm. An example of this technique is called variable elimination search, VES (Larrosa, 2000).

2.3.3 Constraint Programming for Planning and Scheduling

Scheduling problems have been successfully addressed for a wide scope of applications using constraint-based techniques. Most of those problems can be modelled in a natural way, so that, since the actions are set, variables are chosen to correspond to the temporal unknowns (mainly start and end times) or to the ordering of tasks, and constraints gather precedence and resource constraints (Beck and Fox, 1998). Typical CSP modelings for the job shop problem states the start times of the activities as the variables of the CSP, and the constraints are divided in two groups:

- The precedence constraints are a set of inequalities involving the variables corresponding to the start times of the activities of the same job or related by precedence relations, and taking into account the durations of the activities.
- The resource constraints may be defined as disjunctions between the start time of the activities using the same resource. However, other approaches have been used, as representing the use of each resource by all the activities with global constraints, which may allow more efficient filtering algorithms.

Moreover, CP has been used in several recent AI planners (Nareyek et al., 2005; Vidal and Geffner, 2006; Tu et al., 2007; Gabriel and Grandcolas, 2009; Bao et al., 2011), since this paradigm is at the core for combining planning and scheduling techniques.

On the other hand, many constraint-based proposals for solving P&S problems exist in the literature, e.g., (Timpe, 2002; Liu and Jiang, 2006; Gomes et al., 2006; Garrido et al., 2008; Moura et al., 2008; Garrido et al., 2009). Furthermore, several filtering algorithms for specialized scheduling constraints have been developed. Specifically, (Beck and Fox, 2000) and (Barták and Cepek, 2010) model scheduling problems which include alternative and optional tasks respectively, together with their filtering rules. Moreover, the work (Barták and Cepek, 2008) proposes filtering rules for both precedence and dependency constraints in order to solve log-based reconciliation (P&S) problems in databases. In those studies, the precedence constraints signify the same as in P&S problems, while the dependency constraints are given between optional activities which can potentially be included in the final schedule. The work (Laborie et al., 2009) introduces new types of variables (time-interval, sequence, cumulative, and state-function variables) for modelling and reasoning with optional scheduling activities. In addition, (Lombardi and Milano, 2010) presents a set of filtering rules for cumulative constraint propagation when solving an extension of the resource-constrained project scheduling problem which includes time lags and uncertain, bounded activity durations. Furthermore, (Monette et al., 2009) includes a constraint-based approach for the Just-In-Time Job Shop Scheduling, i.e., each activity has an earliness and a tardiness cost with respect to a due date. This approach includes a filtering algorithm which uses a machine relaxation to produce a lower bound, and dedicated heuristics. This work also includes pruning rules which update the variable bounds and detect precedence constraints.

2.4 AI Planning and Scheduling for BPM

In recent years, interest has grown in the integration of P&S techniques with BPMSs, e.g., (Ha et al., 2006; R-Moreno et al., 2007; González-Ferrer et al., 2009;

Rhee et al., 2010; Hoffmann et al., 2010; Tsai et al., 2010; Barba and Del Valle, 2010; PLANET, 2003; Berry and Drabble, 2000; Goldmann et al., 2000; Jarvis and et al., 2000), since there are several points where P&S tools can be effectively applied to BPMSs. However, from our point of view, several connections between these two disciplines remain to be explored.

P&S techniques can be used to enhance different stages of the traditional BPM Life Cycle (Weske, 2007) (cf. Fig. 2.1), as outlined below:

- Process Design & Analysis: As stated, traditionally, this phase is mostly a human activity. However, this phase can be extended so that P&S techniques can be leveraged to automate the generation of optimized BP models by taking into account some information, such as dependencies between activities, activity durations, resource availabilities and objective functions.

- Process Enactment: In this phase, P&S techniques can be used to automatically adopt the execution of BP models to changing circumstances, e.g., resource unavailability, differences between an actual activity duration and expected duration, failures, etc.

- Evaluation: In this phase, P&S techniques can be used to improve BP models by considering the information stored in the execution logs.

Table 2.1 (R-Moreno et al., 2007) describes at a high level the concepts which AI P&S share with the BPM community (for a more detailed description cf. Workflow Management PLANET TCU (PLANET, 2003) roadmap).

Table 2.1: Concepts mapping between AI P&S and BPM.

P&S	BPM
Modelling + Planning and Scheduling	Modelling and Scheduling
Execution	Enactment
Re-planning	Exceptions
Monitoring	Monitoring
Operators	Activities, tasks, ...
Initial State	Organization, resources
Goals	Business goals, service provision, ...

Most related work integrates P&S with BPMS in the enactment phase in order to make dispatching decisions as to which activity should be executed using a resource when it becomes free (dynamic scheduling) (cf. Sect. 2.4.2), while very few integrations are carried out during the modelling phase (cf. Sect. 2.4.1).

2.4.1 P&S for the Process Design & Analysis Phase

In (González-Ferrer et al., 2009), the BP information is provided through an execution/interchange language, XPDL. The XPDL file is analyzed to obtain a workflow pattern decomposition, which is translated into the HTN-PDDL language. This is used as the input of a planner, and the resulting plans could be interpreted as workflow instances. In a similar way, (Alves et al., 2008) presents a proposal for the automatic generation of BP models in workflow systems, based on genetic techniques for the optimized planning and scheduling of BP activities. In (Alves et al., 2008), the BP information is provided through the XPDL language, which is translated into PDDL, and the generated BP model is also specified in XPDL. The XPDL language lacks some power for the correct representation of complex workflow patterns (van der Aalst, 2003). Specifically, only serial, parallel split-join, and parallel exclusive-OR templates are considered in (González-Ferrer et al., 2009).

In a related way, in (R-Moreno et al., 2007), planning tools are used for the semi-automatic generation of BP models, by considering the knowledge introduced through BP Reengineering languages. This knowledge is translated into predicate logic terms in order to be handled by a planner, and an updated BP model is obtained. The process information is provided through an object-oriented structure modelling tool that follows a rule-based approach.

Additionally, (Hoffmann et al., 2010) proposes a planning formalism for the modelling of BPs through an SAP specification (Status and Action Management, SAM), which is a variant of PDDL.

Furthermore, (Ferreira and Ferreira, 2006) proposes to refine BP models by combining learning and planning techniques, starting from processes which are not fully described. (Ferreira and Ferreira, 2006) needs past process executions and examples provided by the user to apply learning techniques. Moreover, (Ferreira and Ferreira, 2006) does not considered the optimization of any objective function in the generation of the plans.

2.4.2 P&S for the Process Enactment Phase

Most of the related works integrate scheduling tools in BPM systems for the enactment phase, in order to take dispatching decisions as to which activity should be executed using a resource when it becomes free (dynamic scheduling). As follows, a representative set of them are briefly summarized.

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 work (Son and 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.

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.

Furthermore, the work (Rhee et al., 2010) presents a Theory of Constraints (TOC)-based method for the improvement of the efficiency of BPs.

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.

Related to decision support systems, (Özbayrak and Bell, 2003) develops a knowledge-based decision support system (KBDSS) for short-term scheduling in flexible manufacturing systems (FMS). This work considers the optimization of the efficient use of the machining cells by using a knowledge-based expert system in order to support the decision making process. Moreover, (Chaturvedi et al., 1993) manages multiple objectives in a hierarchical way. In a related way, (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.

Chapter 3

From Constraint-based Specifications to Optimized BP Enactment Plans

3.1 Introduction

3.1.1 Motivation

Typically, business processes are specified in an imperative way, i.e., an activity sequence that will result in obtaining the related corporate goal is defined. However, declarative BP models are increasingly used and their usage allows the user to specify what has to be done instead of having to specify how it has to be done. Declarative BP model specifications facilitate the human work involved, avoid failures, and obtain a better optimization, since the tacit nature of human knowledge is often an obstacle to eliciting accurate process models (Ferreira and Ferreira, 2006).

The advantages of using declarative languages for BP modelling instead of imperative languages, i.e., support for partial workflows (Wainer et al., 2004), absence of over-specification (Pesic et al., 2007), and provision of more maneuvering room for end users (Pesic et al., 2007), are discussed in several studies, e.g., (Rychkova et al., 2008a; Fahland et al., 2009, 2010; Pichler et al., 2011).

3.1.2 Contribution

Due to their flexible nature, frequently several ways to execute declarative process models exist. Therefore, there are several enactment plans related to a specific declarative model, each one presenting specific values for different objective values,

e.g., overall completion time, i.e., time needed to complete all process instances which were planned for a certain period.

In this chapter, generating optimized BP enactment plans from declarative specifications is proposed in order to optimize the performance of a process, according to objective functions like minimizing the overall completion time. The generated optimized BP enactment plans can leverage the BPM life cycle (Weske, 2007), since they can be used for simulation (Rozinat et al., 2009), time prediction (van der Aalst et al., 2011), recommendations (Schonenberg et al., 2008; Haisjackl and Weber, 2010; Barba et al., 2011), and generation of optimized BP models (R-Moreno et al., 2007; Alves et al., 2008; González-Ferrer et al., 2009; Barba and Del Valle, 2011b), which are innovative and interesting topics nowadays. Specifically, in this Thesis Dissertation, the optimized plans are used for:

1. Giving users of flexible BPMSs assistance during the process execution (cf. Chapter 4).

2. Generating optimized BPMN models (cf. Chapter 5).

For the generation of these optimized enactment plans, activities to be executed have to be selected and ordered (planning problem (Ghallab et al., 2004)) considering both control-flow and resource constraints (scheduling problem (Brucker and Knust, 2006)) imposed by the declarative specification.

For planning and scheduling (P&S) the activities such that the process goal is optimized, a constraint-based approach is proposed since Constraint Programming (CP) (Rossi et al., 2006) supplies a suitable framework for modelling and solving problems involving P&S aspects (Salido, 2010). For this, the declarative model is complemented with information related to estimates regarding the number of instances, activity durations, and resource availabilities.

For the declarative specification of BP models, ConDec (Pesic and van der Aalst, 2006; van der Aalst and Pesic, 2006b) (cf. Sect. 2.1.2) is considered to be a suitable base language, since it allows the specification of BP activities together with the constraints which must be satisfied for correct BP enactment and for the goal to be achieved. In this work, an extension of ConDec which includes reasoning with resources and parallel executions, named ConDec-R (Barba and Del Valle, 2011a) (cf. Sect. 3.2), is considered.

The main contributions of this chapter can be summarized as follows:

- The definition of a language for the constraint-based specification of BPs which extends ConDec (Pesic and van der Aalst, 2006; Pesic et al., 2007), named ConDec-R (cf. Sect. 3.2, Step 1 in Fig. 3.1), to enable the reasoning about resources.

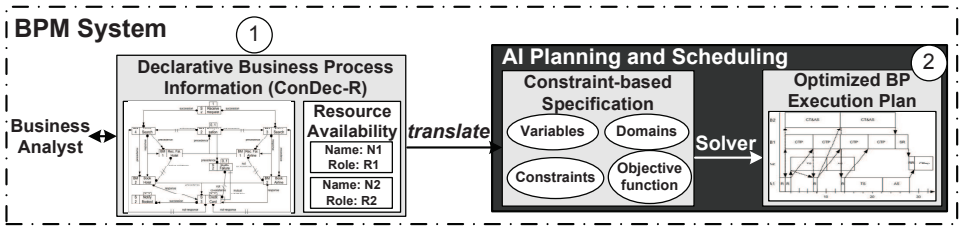


Figure 3.1: AI P&S techniques for the generation of optimized BP enactment plans.

- Automatic planning and scheduling of the BP activities for the generation of optimized BP enactment plans from the ConDec-R specifications, through a constraint-based approach (cf. Sect. 3.3, Step 2 in Fig. 3.1) (Barba and Del Valle, 2011a). The proposed constraint-based approach includes new global constraints for the definition of the high-level relations between activities which can be executed several times, i.e., repeated activities, and the corresponding filtering rules (cf. Appendix B). The developed global constraints facilitate the specification of the problem at the same time as the related filtering rules enable the efficiency in the search for solutions to increase. The developed filtering rules deal with a combination of several aspects, and, in most cases, result in new complex filtering rules.

- Validation of the proposed approach through the analysis of different performance measures related to a range of test models of varying complexity (cf. Sect. 3.4).

It should be emphasized that the proposed constraint-based approach (cf. Sect. 3.3) can be used to help the modelling and the solving of further planning and scheduling problems which include similar relations between repeated activities, and which are unrelated to business process environments.

This chapter is organized as follows: Section 3.2 introduces the declarative BP language which is proposed, i.e., ConDec-R, Section 3.3 details the generation of optimized BP enactment plans from ConDec-R specifications, Section 3.4 deals with the evaluation of the proposed approach, and finally, Section 3.5 summarizes related work.

3.2 ConDec-R: Constraint-based Specification of Business Processes

In order to plan and schedule the BP activities, ConDec is used as a basis. As stated in Sect. 2.1.2, ConDec allows the specification of BP activities together

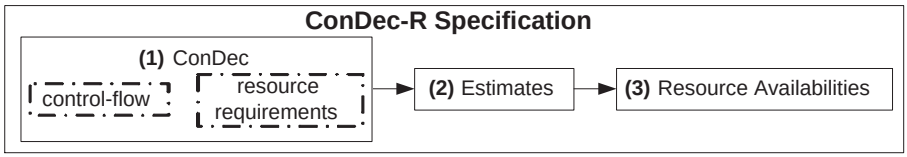


Figure 3.2: ConDec-R process model specification.

with the constraints which must be satisfied for correct BP enactment and for the goal to be achieved. ConDec is based on constraint-based BP models (cf. Def. 5 on page 14), and resource constraints can be specified for each BP activity by associating a role with that activity. Constraints can be added to a ConDec model to specify forbidden behavior, restricting the desired behavior (for a description of the complete set of constraints, cf. (van der Aalst and Pesic, 2006a)). In this way, ConDec basic templates can be divided into existence, relation and negation constraints.

The basic ConDec-R templates, extending the ConDec templates (van der Aalst and Pesic, 2006a), together with its formal specification through constraints and some examples of valid and invalid traces are listed in **Appendix A**.

To make ConDec usable for our concrete application, an extension of ConDec (named ConDec-R) is defined in this chapter, as detailed in Sect. 3.2.1 and Sect. 3.2.2. ConDec-R supports all constraints described in (van der Aalst and Pesic, 2006a) and additionally provides extended support for branched templates, as described in (Montali, 2009).

3.2.1 Extending ConDec with Estimates and Resource Availabilities

To support the direct reasoning of resources (which is not possible in ConDec) ConDec is extended with estimates of activity durations and the specification of resource availabilities. In short, a ConDec-R process model specification (cf. Def. 20 on page 39, Fig. 3.2) extends a ConDec model (cf. Fig. 3.2(1)), i.e., specifying control-flow and resource requirements, by including:

- The **estimated duration** of each activity (cf. Fig. 3.2(2)). Estimates can be obtained by interviewing business experts or by analyzing past process executions (e.g., by calculating the average values of the parameters to be estimated from event logs). Moreover, both approaches can be combined to get more reliable estimates.

- The **available resources** of each role (cf. Fig. 3.2(3)). This information is independent of the ConDec-R activities, and hence it can be changed without affecting the specification of the activities, and vice versa.

Notice that the resource availabilities can be unknown until starting the generation of the optimized plans. This can be tackled by the current proposal since the most static information, i.e., the control-flow and resource requirements (cf. Fig. 3.2(1)), is complemented with more changing information, i.e., the estimates (cf. Fig. 3.2(2)), and finally the most dynamic data, i.e., information on resource availabilities (cf. Fig. 3.2(3)), is included. In this way, the same BP constraint-based specification can be easily adapted to changing conditions, e.g., different resource availabilities or activity durations.

In this way, a ConDec-R model (cf. Def. 20) extends a ConDec model (cf. Def. 5 on page 14) by including resource availabilities and extended BP activities, i.e., BP activities including resource requirements plus estimated duration.

Definition 20. *A **ConDec-R process model** $CR = (Acts, C_{BP}, Res)$ related to a constraint-based process model $S = (A, C_{BP})$ is composed by a set of extended BP activities Acts, which contains tuples $(a, role, dur)$ which includes for each BP activity $a \in A$ the role of the required resource (i.e., role) and the estimated duration (i.e., dur); a set of ConDec constraints C_{BP}; a set of available resources Res, and composed by tuples $(role, \#role)$ which includes for each role (i.e., role) the number #role of available resources.*

An example of a ConDec-R process model is depicted in Fig. 3.3(1), where $Acts = \{(A, R_1, 5), (B, R_2, 3), (C, R_1, 4)\}$, $C_{BP} = \{Exactly(2, A), Precedence(A, B), Precedence(A, C), NotCoexistence(B, C)\}$, and $Res = \{(R_1, 2), (R_2, 1)\}$.

3.2.2 Extending ConDec with Parallel Execution of Activities

In ConDec no parallelism is considered in the execution of activities which are related by ordering constraints since ConDec activities are atomic. In this work, non atomic activities, i.e., durative activities, are considered, and hence the ConDec templates are extended so that the relations which are stated in Allen's interval algebra (Allen, 1983) are allowed in order to deal with temporal reasoning. In ConDec-R, the relation *activity b must be executed after a* can imply four different meanings:

- $st(b) \geq et(a)$ (default option)
- $st(b) \geq st(a)$
- $et(b) \geq et(a)$

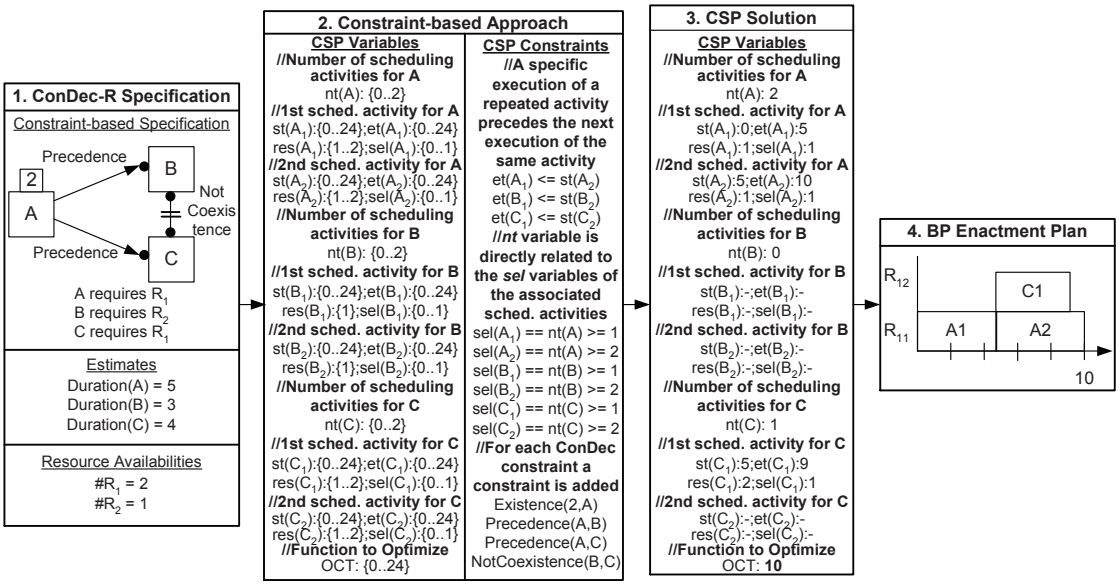


Figure 3.3: From ConDec-R specification to BP enactment plan.

- $et(b) \geq st(a)$

In this way, in ConDec-R some of the ConDec templates (Pesic and van der Aalst, 2006; Pesic et al., 2007) are adapted and extended by considering the possible parallelism in the execution of those activities that are related by ordering constraints. This leads to four variants for the same temporal relation between two activities *a* and *b*, which is represented by an additional label at the end of the template name. This label represents: first, the time related to *a* which is constrained (start, S, or end, E), and the time related to *b* which is constrained (start, S, or end, E) through the inclusion of the template[1]. Therefore, the four variants for the same template are: SS, ES, SE, EE. As an example, ResponseSS(A,B) means that after starting the last execution of A, at least one execution of B must be started. A case of study which include some examples of this kind of extension is detailed in Sect. 5.3.

Notice that the ConDec-R language allows those typical constraints which are considered in literature to be specified, i.e. (Sadiq et al., 2005; van der Aalst and Pesic, 2006a):

- Constraints which restrict the selection of activities.
- Constraints which restrict the ordering of activities.

[1] In a similar way, ConDec++ (Montali, 2009) and Saturn (Demeyer et al., 2010) also consider constraints imposed on the start and the completion times of non-atomic activities.

- Constraints which restrict the requirements of resources.

3.3 From ConDec-R to Optimized Enactment Plans

In this section, the complete process which is proposed to generate BP enactment plans from a ConDec-R specification through constraint programming is detailed (cf. Fig. 3.3). As stated, BP activities and constraints are specified in a ConDec-R model (cf. Step 1 in Fig. 3.3, Sect. 3.2) so that frequently several feasible ways to execute this model exist. Each specific feasible execution of a ConDec-R model leads to a specific value for the function to optimize. In general, there can be more than one optimal execution, i.e., feasible solution leading to a minimal completion time[2]. In order to generate optimal (or optimized) execution plans for a specific ConDec-R model[3], a constraint-based approach for P&S the BP activities (cf. Step 2 in Fig. 3.3) is proposed. The obtained plans, i.e., solutions to the COP (cf. Step 3 in Fig. 3.3), optimize the specified objective function (cf. Def. 12 on page 17) and satisfy all the constraints which are stated in the specification of the problem, i.e., reaches the specified goal (cf. Def. 11 on page 17). Lastly, the generated plans are visualized as Gantt chart (Gantt, 1913) (cf. Step 4 in Fig. 3.3) which illustrates the start and the end times of the activities together with the assigned resource.

3.3.1 Translating the ConDec-R Model as a CSP Model

As first step of the constraint-based approach, the Condec-R model needs to be translated to a CSP. Regarding the CSP model of the proposed constraint-based approach, BP activities (repeated activities, cf. Def. 21), which can be executed arbitrarily often if not restricted by any constraints, are modelled as a sequence of optional scheduling activities (cf. Def. 22). This is required since each execution of a BP activity is considered as one single activity which needs to be allocated to a specific resource and temporarily placed in the enactment plan, i.e., stating values for its start and end times.

In this way, there are two main types in the proposed CSP model (cf. Fig. 3.4):

1. A type representing the BP activities, named *RepeatedActivity*.

[2]In this chapter the overall completion time as objective function is considered. However, note that the proposal can be easily extended to support further objective functions like cost.

[3]Notice that although the generation of optimal solutions is desirable, optimized solutions are acceptable for problems which present NP-complexity, such as the considered problems, since finding the optimal solution cannot been ensured in all cases.

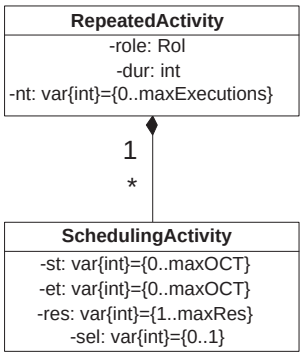


Figure 3.4: RepeatedActivity and SchedulingActivity types

2. A type representing each execution of the BP activity, named *Scheduling-Activity*.

Definition 21. *A **repeated activity** $ra = (role, dur, nt)$ is a BP activity which can be executed several times. The properties of a repeated activity are:*

- *role: it represents the role of the required resource for the activity execution*[4].
- *dur: it is related to the estimated duration of the BP activity*[5].
- *nt: it is a CSP variable which represents the number of times the BP activity can be executed*[6].

Given a ConDec-R process model $CR = (Acts, C_{BP}, Res)$ (cf. Def. 20), the set of related repeated activities is composed by $\{(role, dur, nt), (a, role, dur) \in Acts\}$. For example, activities *A*, *B* and *C* of the constraint-based model of Fig. 3.3(1) are repeated activities. Activity *A*, for example, requires resource of role R_1 and has an estimated duration of 5. Moreover, the maximum number of activity executions for A is between 0 and 2.

Since a repeated activity can be executed several times, a scheduling activity (cf. Def. 22) refers to a specific execution of a repeated activity. In this way, for each repeated activity, *nt* scheduling activities exist, which are added to the CSP problem specification, apart from including a variable *nt* (cf. Fig. 3.3(2)). For example, for repeated activity *A* in Fig. 3.3 two scheduling activities exist (referred to as A_1 and A_2).

[4]For sake of simplicity, the same required resource is considered for all the executions of a BP activity. Note that the proposed approach can also deal with BP activities which require several resources of various kinds of roles.

[5]The same estimated duration is considered for all the executions of a BP activity.

[6]Lower and upper bounds are related to the domain of each integer CSP variable *var*, representing minimum and maximum value which can be given to *var* in a feasible solution, respectively. Thereby, $LB(var)$ and $UB(var)$ refer to the lower and upper bounds of the domain of *var*.

Definition 22. *A **scheduling activity** $sa = (st, et, res, sel)$ represents a specific execution of a repeated activity, where:*

- *st: it is a CSP variable indicating the start time of the activity execution (each execution of a BP activity needs to be temporarily placed in the enactment plan).*
- *et: it is a CSP variable indicating the end time of the activity execution (each execution of a BP activity needs to be temporarily placed in the enactment plan).*
- *res: it is a CSP variable representing the resource used for the execution (identified by a number between 1 and #role(role(res))).*
- *sel: it is a CSP variable indicating whether or not the scheduling activity is selected to be executed (cf. Fig. 3.3(2)).*

Moreover, an additional CSP variable representing the objective function to be optimized, i.e., the overall completion time in the context of the current proposal, named OCT, is also included in the CSP model: $OCT = max_{a \in Acts}(et(a_{nt(a)}))$[7].

In order to ensure the consistency between the CSP variables, several constraints have to be added to the CSP (cf. Fig. 3.3(2)):

- $\forall i : 1 \leq i < UB(nt(a)) : et(a_i) \leq st(a_{i+1})$, i.e., a specific execution of a repeated activity precedes the next execution of the same activity.
- $\forall i : 1 \leq i \leq UB(nt(a)) : sel(a_i) = nt(a) >= i$, i.e., the nt variable of the repeated activity is directly related to the sel variables of the associated scheduling activities, e.g., if $nt(a) = 2$, then $sel(a_1) = 1$, $sel(a_2) = 1$ and $\forall i : nt(a) < i \leq UB(nt(a)), sel(a_i) = 0$.

Furthermore, for each ConDec template, a global constraint is added to the CSP model, i.e., $Existence(2,A)$, $Precedence(A,B)$, $Precedence(A,C)$, and $Not-Coexistence(B,C)$ for the constraint-based model depicted in Fig. 3.3.

Definition 23. *A **CSP-ConDec problem** related to a ConDec-R process model $CR = (Acts, C_{BP}, Res)$ (cf. Def. 20 on page 39) is a COP $P_o = (V, D, C_{CSP}, o)$ (cf. Def. 15 on page 26) where:*

- *The set of variables V is composed by all the CSP variables included in the presented CSP model plus the CSP variable related to the overall completion time (OCT), i.e., $V = \{nt(a), a \in Acts\} \cup \{st(a_i), et(a_i), res(a_i), sel(a_i), a \in Acts, i \in [1..UB(nt(a))]\} \cup OCT$.*

[7] a_i refers to the scheduling activity related to the i-th execution of the repeated activity a.

- *The set of domains D is composed by the domain of each CSP variable v, $D(v)$, i.e.:*

$$
\begin{aligned}
D = \{&D(nt(a)) = \{0..MC\},\ a \in Acts\} \cup \\
&\{D(st(a_i)) = \{0..MC \times \sum_{a \in Acts} dur(a)\}, a \in Acts, i \in [1..UB(nt(a))]\} \cup \\
&\{D(et(a_i)) = \{0..MC \times \sum_{a \in Acts} dur(a)\}, a \in Acts, i \in [1..UB(nt(a))]\} \cup \\
&\{D(res(a_i)) = \{1..\#role(role(a))\}, a \in Acts, i \in [1..UB(nt(a))]\} \cup \\
&\{D(sel(a_i)) = \{0..1\}, a \in Acts, i \in [1 \ldots UB(nt(a))]\}
\end{aligned}
$$

  where MC is the maximum cardinality for the BP activities (established by existence relations in the constraint-based model).

- *The set of constraints C_{CSP} is composed by the global constraints (implemented by the filtering rules, cf. Appendix B) related to the ConDec-R constraints included in C_{BP} together with the constraints from the proposed CSP model, i.e.:*

  – $\forall i : 1 \leq i < nt(a) : et(a_i) \leq st(a_{i+1})$
  – $\forall i : 1 \leq i \leq UB(nt(a)) : sel(a_i) = nt(a) \geq i$

  for each repeated activity $a \in Acts$.

- *The objective function o is minimizing the OCT variable.*

Figure 3.3 includes the translation from a ConDec-R specification into a CSP so that the CSP variables and constraints are stated as explained in Def. 23 (cf. Step 2). In general, for each repeated activity *a*, a CSP variable *nt* is added to the CSP model. Thereby, the value for $LB(nt(a))$ is initially set to 0 (it will be automatically updated during the solving process if an existence constraint is added through the corresponding filtering rule, cf. Appendix B), and for $UB(nt(a))$ a rough initial estimate is made by considering the maximum obligatory cardinality of all repeated activities which is stated by existence constraints. For the constraint-based model depicted in Fig. 3.3, for example, the upper bound for all repeated activities is initially set to 2 (due to the existence constraint related to activity *A*). This value states the minimum *nt* for all BP activities ensuring a feasible solution (the optimal solution, however, in general includes lower values of *nt* for several activities).

Moreover, $LB(OCT)$ is initially set to 0[8], and $UB(OCT)$ is estimated as the maximum cardinality times the sum of the duration of all the BP activities, i.e.,

[8] $LB(OCT)$ can be also initialized to other different values, e.g., maximum duration of all the mandatory activities.

$2 \times (5+3+4)$ in the example of Fig. 3.3. This is since a trivial solution which can be obtained results in a plan which includes the execution of each BP activity the maximum number of times when all activities are sequentially executed.

For similar reasons, for each scheduling activity a_i, lower and upper bounds for *st* and *et* are set to the lower and the upper bounds of OCT. Furthermore, in general, $D(res(a_i)) = \{1..\#role(role(a))\}$, i.e., $D(res(A_i)) = \{1..2\}$ and $D(res(C_i)) = \{1..2\}$ for any i since $\#R_1 = 2$, and $D(res(B_i)) = \{1\}$ for any i since $\#R_2 = 1$ for the constraint-based model depicted in Fig. 3.3. In addition, for each scheduling activity a_i, $D(sel(a_i)) = \{0..1\}$, since *sel* is a binary variable indicating whether or not the scheduling activity is selected to be executed.

3.3.2 Global Constraints and Filtering Rules

To improve the modelling of the problems and to efficiently handle the constraints in the search for solutions, the proposed constraint-based approach includes for each ConDec template a related global constraint which is implemented through a filtering rule (responsible for removing values which do not belong to any solution) for the definition of the high-level relations between the BP activities. In this way, the constraints stated in the ConDec-R specification (cf. Def. 20 on page 39) are included in the CSP model through the related global constraints. These global constraints facilitate the specification of the problem at the same time as the related filtering rules enable the efficiency in the search for solutions to increase since during search process these filtering rules remove inconsistent values from the domains of the variables. Notice that in the CSP model specification, initial estimates are made for upper and lower bounds of variable domains (cf. Sect. 3.3.1), and these values are refined during the search process.

As examples, three filtering rules of varied complexity are presented as follows (for a detailed description of all the developed filtering rules cf. **Appendix B**).

Existence(A, N)

$Existence(A,N)$ means that A must be executed more than or equal to N times, $nt(A) \geq N$.

```
Existence(A,N) is added ->
    If N > LB(nt(A)) then
        LB(nt(A)) <- N
```

Figure 3.5: Filtering Rule for the Existence Template

The *Existence* rule (Fig. B.1) is invoked when the template is added to the constraint model, hence its trigger ”Existence(A,N) is added”.

Proposition 1. *If implemented properly, the time complexity of the rule Existence, which includes all possible recursive calls, is* $\Theta(1)$.

Proof. The *Existence* rule is fired only when the constraint is added, and the time complexity of its execution is constant, hence the complexity of this rule is $\Theta(1)$. □

Precedence(A, B)

$Precedence(A,B)$ means that before the execution of B, A must have been executed, $nt(B) > 0 \Rightarrow (nt(A) > 0) \wedge (et(A_1) \leq st(B_1))$. As can be seen in Fig. A.1(a) (cf. Appendix A), this relation implies that A_1 must precede B_1 in the case that $nt(B) > 0$.

```
Precedence(A,B) is added OR bounds of
nt(A) changed OR bounds of nt(B) changed OR bounds of st(B1)
changed OR bounds of et(A1) changed ->
    If LB(nt(B)) > 0 then
        nt(A) <- nt(A) - {0}
    If UB(nt(A)) == 0 then
        VAL(nt(B)) <- 0
    If LB(et(A1)) > LB(st(B1))then
        LB(st(B1))) <- LB(et(A1))
    If UB(et(A1)) > UB(st(B1)) then
        UB(et(A1)) <- UB(st(B1))
```

Figure 3.6: Filtering Rule for the Precedence Template

The *Precedence* rule (Fig. 10) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 2. *If implemented properly, the worst-case time complexity of the rule Precedence, which includes all possible recursive calls, is* $O(n)$*, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *Precedence* rule can be fired, at most, $3 \times n$ times. This is due to the fact that only a change in the first execution (*st* and *et* variables of Act_1) or in the *nt* variable of a repeated activity can fire this rule. Moreover, the time complexity of the *Precedence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

Alternate Precedence(A, B)

$AlternatePrecedence(A,B)$ means that before the execution of B, A must have been executed, and between each two executions of B, A must be executed. It implies that:

```
Alternate Precedence (A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
st(Ai) for any i changed OR bounds of et(Ai) for any i changed OR
bounds of st(Bi) for any i changed OR bounds of et(Bi) for any i
changed ->
   if (LB(nt(A)) < LB(nt(B))) {LB(nt(A)) <- LB(nt(B))}
   if (UB(nt(B)) > UB(nt(A))) {UB(nt(B)) <- UB(nt(A))}
   for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi;
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi
   }
   for (int i = 1; i < LB(nt(B)); i++){
      int dif = UB(nt(A)) - LB(nt(B));
      SchedulingActivity a = Ai+dif+1;
      SchedulingActivity b = Bi;
      if (LB(et(b)) > LB(st(a))) {LB(st(a)) <- LB(et(b))}// Bi -> Ai+dif+1
      if (UB(st(a)) < UB(et(b))) {UB(et(b)) <- UB(st(a))}// Bi -> Ai+dif+1
   }
   for (int i = 2; i <= UB(nt(B)); i++){  // force exists A between Bi-1 and Bi
      int dif = UB(nt(A)) - max(i,LB(nt(B)));
      SchedulingActivity b1 = Bi-1;
      SchedulingActivity b2 = Bi;
      int j = i; // Candidate As between Bi-1 and Bi
      SchedulingActivity aFor;
      int possible = 0;
      while (j <= (i + dif) && possible < 2) {
         SchedulingActivity aPos = Aj;
         //If Bi-1->aPos->Bi possible
        if (UB(st(aPos))>=LB(et(b1)) && LB(et(aPos))<=UB(st(b2))){
            possible++;
            aFor = aPos;
        }
        j++;
      } // end while j
      if (possible == 1){ // force Bi-1 -> aFor -> Bi
        // Bi-1 -> aFor
        if (LB(et(b1))>LB(st(aFor))){LB(st(aFor)) <- LB(et(b1))}
        if (UB(st(aFor))<UB(et(b1)) {UB(et(b1)) <- UB(st(aFor))}
        // aFor -> Bi
        if (LB(et(aFor))>LB(st(b2))){LB(st(b2) <- LB(et(aFor))}
        if (UB(st(b2))<UB(et(aFor))){UB(et(aFor)) <- UB(st(b2))}
      } // end if
      if(possible == 0)
        return Failure;
   }//end for i
```

Figure 3.7: Filtering Rule for the Alternate Precedence Template

1. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) >= nt(B)$.

2. Between each two executions of B, A must be executed at least once. Specifically, between the $(i-1)$-th and the i-th execution of B, the earliest execution of A that can exist is i, and hence A_{i-1} must precede B_{i-1} (as can be seen in Fig. A.1(b), cf. Appendix A). In a similar way, between the $(i-1)$-th and the i-th execution of B, the latest execution of A that can exist

is $i + nt(A) - nt(B)$, and hence B_i must precede $A_{i+nt(A)-nt(A)+1}$. This can also be seen in Fig. A.1(b), where the possible activities to be executed between the $(i-1)$-th and the i-th execution of B are framed within the dotted rectangle. $\forall i : 2 \leq i \leq nt(B) : \exists j : i \leq j \leq i + nt(A) - nt(B) : st(A_j) \geq et(B_{i-1}) \wedge et(A_j)) \leq st(B_i)$.

3. Before B, A must be executed: $st(B_1) \geq et(A_1)$.

Proposition 3. *If implemented properly, the worst-case time complexity of the rule AlternatePrecedence, which includes all possible recursive calls, is $O(n \times nt^3)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *AlternatePrecedence* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *AlternatePrecedence* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

3.3.3 Solving the COP

Once the problem is modelled, several constraint-based mechanisms can be used to obtain the solution for the COP, i.e., optimized enactment plans (cf. Def. 24). Since the generation of optimized plans presents NP-complexity (Garey and Johnson, 1979), it is not possible to ensure the optimality of the generated plans for all the cases. The developed constraint-based approach, however, allows solving the considered problems in an efficient way, as demonstrated in Sect. 3.4.

Definition 24. *A **BP enactment plan** is composed by:*

1. *The number of times each BP activity is executed.*
2. *The start and the completion times for each activity execution.*
3. *The resource which is used for each activity execution.*

In general, when optimizing a CSP variable, if a feasible solution which is known exists, the value of the variable to optimize in the known solution can be used for discarding large subsets of fruitless candidates by using upper and lower estimated bounds of the quantity being optimized during the search process. Thus, if a known feasible solution S for the problem to solve exists, the objective value for this solution (S^{OCT}) is a valuable information which can be added to the

constraint model through the constraint $OCT < S^{OCT}$. Thus, some non optimal candidates, i.e., candidates whose OCT value cannot be less than S^{OCT} in any case, are discarded during the search, increasing the efficiency in the search for solutions.

Moreover, in the proposed approach, during the search process, some of the values which only lead to non-feasible solutions, i.e., inconsistent values, are removed from the domains of the CSP variables through the developed filtering rules (cf. Appendix B) in order to reduce the search space by maintaining arc consistency (cf. Def. 17 on page 29).

In the proposed approach, the developed filtering rules and CSP modelling (cf. Sect. 3.3.1) are implemented such that they maintain the arc consistency for all pairs of CSP variables during all the search process.

There are a wide scope of search algorithms of varied nature which can be used for finding optimal/optimized solutions to the COPs. In general, the suitability and efficacy of each search algorithm highly depend on the specific problem to solve. For this reason, different search strategies are tested in the empirical evaluation of each chapter related to the generation of optimized enactment plans from ConDec-R specifications (cf. Sect. 3.4, Sect. 4.4, and Sect. 5.4).

A proof-of-concept prototype has been implemented through a web-based application, which allows for the generation of optimized enactment plans from ConDec-R specifications, and it can be accessed at:

http://regula.lsi.us.es/OptBPPlanner [9]

3.4 Empirical Evaluation

In order to evaluate the effectiveness of the proposal, a controlled experiment is conducted. Section 3.4.1 describes the design underlying the experiment, and Section 3.4.2 shows the experimental results and the data analysis.

3.4.1 Experimental Design

Purpose: The purpose of the experimental evaluation is to determine the efficacy of the proposed constraint-based approach for generating optimized plans. As far as the developed filtering rules greatly influence and improve the efficiency of the search, it has been considered suitable to compare the results obtained by: first, the proposal with propagators (i.e., filtering rules) and the related high-level global

[9] Only non-branched templates and the default option in the ordering constraints, i.e., ES, are considered in this prototype.

constraints; and secondly, with neither propagators nor global constraints (in this case, the equivalent local constraints are included)[10].

Experimental Design: The behavior of five representative propagators are tested: Exactly, Existence, Precedence, Alternate and Chain Precedence. The results obtained with two proposals, which use the same variables, are compared to study the efficiency:

1. With propagators: the user-defined constraints are used for the establishment of the high-level relations between the ConDec-R activities. In this case, the proposed filtering algorithms are responsible for removing inconsistent values from the domain of the variables.

2. Without propagators: the relations between the activities are established directly through basic constraints, as shown in Appendix A.

Objects: Due to the use of a specific and new language, it has been impossible to find a set of public benchmarks for ConDec-R problems.

ConDec-R problems can be modelled as an extension of scheduling problems. In this way, a set of well-known job shop scheduling (cf. Sect. 2.2.1) benchmarks (FT06 (6x6), FT10 (10x10), ABZ06 (10x10), LA21 (15x10), LA36 (15x15)) have been extended for use in the empirical evaluation in the following way:

- In a job shop scheduling problem, the activities are executed only once. For this evaluation, the cardinality of a random percentage p% of activities has been set to a value c greater than one, in order to test the behavior of the proposal when some repeated activities are executed several times. In Table 3.1, the column $Card = c(p\%)$ refers to extensions to job shop scheduling problems in a way that the cardinality of a random percentage p% of activities has been set to a value c. In order to minimize the influence of the random component in the performance measures which are considered, 50 random instances are generated for each problem so that average values are reported.

- In a job shop scheduling problem, all the temporal relations between activities are precedence relations, i.e., ”A precedes B” means that A must finish before B starts. This relation can be modelled through several ConDec-R templates, e.g., Precedence, Response, or Succession. For modelling the temporal relations of job shop scheduling problems in ConDec, the Precedence template has been selected although other options can also be used. Moreover, in order to consider more complex relations between the repeated

[10] Due to the fact that the analyzed relations are introduced in this work for the first time, there has not been possible to find a previously developed solver which deals with these relations.

activities, the alternate and chain precedence relations[11] are also considered in the following way: in the original problem P, all the precedence relations between the activities of the same job are changed to alternate precedence relations (this is represented by *PAlt*. in Table 3.1) and chain precedence relations (this is represented by *PChain* in Table 3.1).

Setting the cardinality of p% activities to a value $v > 1$, can result in a solution where there are p'% activities ($p' > p$) with cardinality v. For example, each activity a with $nt(a) = v$ implies that for all b where $AlternatePrecedence(b,a)$ or $ChainPrecedence(b,a)$ hold, then $nt(b) \geq nt(a)$ must be satisfied (the same logic applies for all the activities c such that $AlternatePrecedence(c,b)$ or $ChainPrece-dence(c,b)$, etc).

Notice that the scheduling benchmarks with the lowest complexity are those that only include precedence relations and have the lowest cardinality for the activities.

Independent Variables: For the empirical evaluation, the random percentage of activities set to a cardinality higher than 1, p%, and the value for this cardinality, c are taken as independent variables.

Response Variables: Some performance measures are reported (Table 3.1):

- B_P^M: Number of instances for which the makespan found by the proposal with propagators is shorter than the makespan found without propagators.
- B^M: Number of instances for which the makespan found by the proposal without propagators is shorter than the makespan found with propagators.
- B_P^T: Number of instances for which the solutions found by both proposals obtain the same makespan value, but the proposal with propagators is faster. The values related to this variable appear between brackets in Table 3.1, and it is only included in the case that it is greater than one.
- B^T: Number of instances for which the solutions found by both proposals obtain the same makespan value, but the proposal without propagators is faster. The values related to this variable appear between brackets in Table 3.1, and it is only included in the case that it is greater than one.

Experimental Execution: For both proposals, a complete search approach has been applied: first, the variables related to the number of times that each activity is executed are instantiated. The search procedure then determines the

[11] Alternate Precedence(A, B) means that before the execution of *B*, *A* must have been executed, and between each two executions of *B*, *A* must be executed. In a similar way, Chain Precedence(A, B) means that immediately before *B*, *A* must be executed. For more details, cf. Appendix A.

order of execution of the activities within each resource at each step, such that the next resource to be ranked is selected depending on its slack. Within each resource, the activities are ranked in a non-deterministic way.

For the experiments, each algorithm is run until it finds the optimal solution or until a 5-minute CPU time limit has been reached. The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running Windows XP. In order to solve the constraint-based problems (cf. Sect. 3.3), the developed algorithms have been integrated with the system COMET (Dynadec, 2011), which is able to generate high-quality solutions for highly constrained problems in an efficient way.

3.4.2 Experimental Results and Data Analysis

As stated before, several experimental results are shown in Table 3.1. After analyzing these values, some conclusions can be obtained:

- For problems with only precedence relations, the solutions for the two proposals are very similar. In this case, the problem is simply a Job Shop problem, and COMET includes efficient mechanisms for solving scheduling problems.
- For problems with alternate and chain precedence relations, the proposal without propagators obtains better solutions for problems with lower complexity (or cardinality). When the percentage of activities with greater cardinality increases, the proposal with propagators proves better than the other proposal in almost all instances.

In short, the proposal without propagators obtains better solutions for problems of lower complexity (only precedence relations, low cardinality), while the proposal with propagators is much better for more complex problems, and hence it seems to be better for general cases[12].

[12]Notice that a limited empirical evaluation is carried out, and hence the conclusions which are obtained are dependent on the considered problems.

Table 3.1: Results on a set of ConDec-R problems from JSS instances

	Card = 2 (5%)		Card = 2 (10%)		Card = 3 (5%)		Card = 3 (10%)	
Problem	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)	B_P^M (B_P^T)	B^M (B^T)
FT06 Prec.	0	0 (1)	0	0 (2)	0	0	0	0 (2)
FT06 Alt.	6 (1)	20 (22)	7 (2)	36 (4)	37	0 (3)	50	0
FT06 Chain	7 (1)	23 (19)	6 (2)	37 (4)	37	0 (3)	50	0
FT10 Prec.	0	0	0	0	0	0	0	0
FT10 Alt.	11	36	5	44	50	0	50	0
FT10 Chain	13	35	3	46	50	0	50	0
ABZ06 Prec.	0	0	0	0	0	0	0	0
ABZ06 Alt.	13	30	6	42	50	0	50	0
ABZ06 Chain	10	36	3	44	50	0	50	0
LA21 Prec.	0	0	0	0	0	0	0	0
LA21 Alt.	11	39	2	48	50	0	50	0
LA21 Chain	3	47	2	48	50	0	50	0
LA36 Prec.	0	0	0	0	0	0	0	0
LA36 Alt.	3	47	1	49	50	0	50	0
LA36 Chain	0	50	0	50	50	0	50	0

3.5 Related Work

In recent years, several filtering algorithms for specialized scheduling constraints have been developed. Specifically, (Beck and Fox, 2000) and (Barták and Cepek, 2010) model scheduling problems which include alternative and optional tasks respectively, together with their propagators. Furthermore, the work (Barták and Cepek, 2008) proposes propagators for both precedence and dependency constraints in order to solve log-based reconciliation (P&S) problems in databases. In those studies, the precedence constraints mean the same as in P&S problems, while the dependency constraints are given between optional activities which can potentially be included in the final schedule. The work (Laborie et al., 2009) introduces new types of variables (time-interval, sequence, cumulative, and state-function variables) for modelling and reasoning with optional scheduling activities. In the current chapter, the proposed model and propagation for the optional activities are very similar to the proposal presented in (Laborie et al., 2009).

Regarding the works (Beck and Fox, 2000), (Barták and Cepek, 2008), (Barták and Cepek, 2010) and (Laborie et al., 2009), the main contribution from the developed propagators is the complex reasoning about several combined innovative aspects, such as the alternating executions of repeated activities together with the variable number of times which these activities are executed (i.e. alternate and chain relations). Furthermore, the areas of application of those studies are unrelated to those of the presented proposal.

Additionally, there exist some proposals which could be used to generate optimized enactment plans for BPs from constraint-based process specifications. Specifically, (Pesic, 2008) proposes the generation of a non-deterministic finite state automaton from constraint-based specifications based on Linear Temporal Logic (LTL) which represents exactly all traces that satisfy the LTL formulas. When extending this approach by including estimates, the overall completion time of all the traces could then be calculated (e.g., (van der Aalst et al., 2011)). However, the big disadvantage following such an approach would be that the process of generating the automaton from the declarative specifications is NP complete, and, unlike the proposed approach, no heuristic can be used. In a similar way, CLIMB (Montali, 2009) could be used to generate quality traces from declarative specifications, and calculate its completion time. Then, the best traces could be selected. Unlike the proposed approach, (Montali, 2009) does neither consider optimality nor resource availabilities. Therefore, this would only cover the planning part of the current proposal, but not the scheduling aspects addressed by the proposed approach.

Chapter 4

User Recommendations for the Optimized Execution of BPs

4.1 Introduction

4.1.1 Motivation

In the current dynamic business world the economic success of an enterprize increasingly depends on its ability to react to changes in its enterprize in a quick and flexible way. Therefore, flexible BPMSs (cf. Def. 4 on page 10) are required to allow companies to rapidly adjust their BP (cf. Def. 1 on page 9) to changes in the environment (van der Aalst and Jablonski, 2000). The specification of process properties in a declarative way is an important step towards the flexible management of BPMSs (van der Aalst et al., 2009).

As mentioned, due to their flexible nature, frequently several ways to execute declarative process models exist. Typically, given a certain partial trace (reflecting the current state of the process instances), users can choose from several enabled activities which activity to execute next. This selection, however, can be quite challenging since performance goals of the process (e.g., minimization of overall completion time) should be considered, and users often do not have an understanding of the overall process. Moreover, optimization of performance goals requires that resource capacities are considered. Therefore, recommendation support is needed during BP execution, especially for inexperienced users (van Dongen and van der Aalst, 2005).

4.1.2 Contribution

In order to support users of flexible BPMSs during process execution in optimizing performance goals like minimizing the overall completion time (i.e., time

needed to complete all process instances which were planned for a certain period), in this chapter the use of optimized enactment plans which are generated from declarative BP specifications (cf. Chapter 3) is proposed for giving recommendations.

Recommendations on possible next steps are then generated taking the partial trace and the optimized plans into account. In the proposed approach, replanning is supported if actual traces deviate from the optimized enactment plans (e.g., because estimates turned out to be inaccurate).

In order to evaluate the effectiveness of the proposed recommendation system, different constraint-based algorithms are applied to a range of test models of varying complexity. The suitability of the proposed approach is tested regarding both (1) build-time, i.e., for the generation of complete optimized plans before starting the BP enactment; and (2) run-time, i.e., for the generation of partial, optimized plans by considering the actual partial trace of the process as the execution of the process proceeds (replanning). The results indicate that the proposed approach produces a satisfactory number of suitable solutions, i.e., solutions which are optimal in most cases and quite good in other cases[1].

The main contributions of this chapter can be summarized as follows:

- Method for generating recommendations to users of flexible BPMSs during run time (cf. Sect. 4.2 (Barba et al., 2011)). For this, optimized enactment plans which are generated from constraint-based specifications (cf. Chapter 3) are used.

- Validation of the proposed approach through the analysis of different performance measures related to a range of test models of varying complexity (cf. Sect. 4.4).

This chapter is organized as follows: Sect. 4.2 includes an overview of the proposed approach, Sect. 4.3 shows the application of the proposed approach to a running example, Sect. 4.4 shows some experimental results, Sect. 4.5 presents a critical discussion of the proposed approach, and finally, Sect. 4.6 summarizes related work.

4.2 Method for Generating Recommendations

As stated, constraint-based processes offer much flexibility. Typically, given a constraint-based process model and a certain partial trace, users can choose from

[1]Notice that a limited empirical evaluation is carried out, and hence the conclusions which are obtained are dependent on the considered problems.

several enabled activities which activity to execute next, which is a challenging selection in most cases. In order to address this challenge, an approach to assist users during process execution in optimizing performance goals like minimizing the overall completion time is proposed. Specifically, users of flexible BPMSs are supported during process execution by a recommendation service which provides recommendations on how to proceed best with the execution. Hereby, a recommendation (cf. Def. 25) is composed by one or more enabled activities (cf. Def. 10 on page 17) to be executed next, together with their resource allocations since both control-flow and resource perspectives are considered.

Definition 25. *A **recommendation** is composed by a set of pairs* (a_i, R_{jk}) *suggesting to start the i-th execution of activity a using resource* R_{jk}[2].

For example, the recommendation $< (A_1, R_{01}), (B_2, R_{12}) >$ suggests to start the first execution of activity A using resource R_{01} and the second execution of activity B using resource R_{12}.

The recommendation service is based on optimized enactment plans which are already generated during build-time by P&S all BP activities (cf. Chapter 3) and further optimized during run-time (cf. Fig. 4.1). Recommendations are suggested during the BP execution at time T when the partial trace of the BP is part of one of the considered optimized enactment plans which contains one or more activities which start right at time T. In this way, at specific times of the process execution, the recommendation system generates the recommendations by considering: (1) the optimized enactment plans, (2) the partial traces (cf. Def. 8 on page 16) of the process instances to be optimized, and (3) the resource availabilities (cf. Sect. 4.2.1). Thereby, the recommendation service ensures that not only single process instances get optimized, but the whole set of instances which is planned to be executed within a certain timeframe, hence allowing for a global optimization.

4.2.1 Generating Recommendations on Possible Next Execution Steps

This section describes how the generated optimized plans (cf. Chapter 3) are then used for assisting users during process execution. At run-time, process instances (cf. Def. 9 on page 16) are executed by authorized users (a in Fig. 4.1). At any point during the execution of a process instance, the user can select from the set of enabled activities (cf. Def. 10 on page 17) what to do next. However, to guide the user to optimize the overall process goals, recommendations (cf. Def. 25) are

[2] R_{jk} refers to the k-th resource with role j.

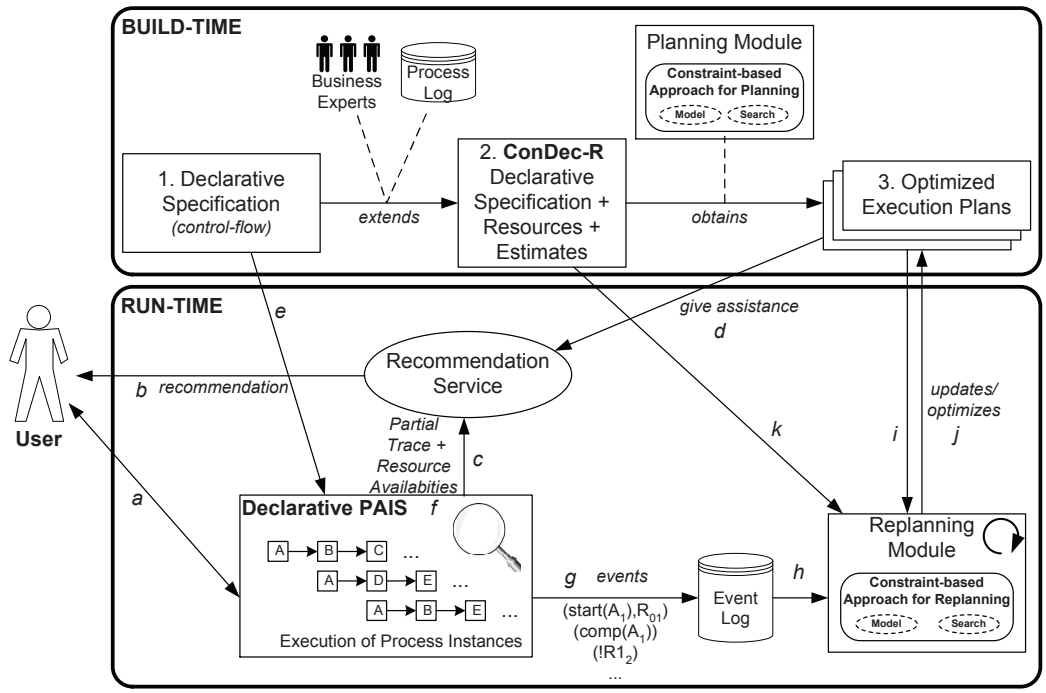


Figure 4.1: Generating Recommendations on Possible Next Execution Steps.

provided by the recommendation service (b in Fig. 4.1), i.e., proposing the most suitable activity to execute next[3].

Algorithm 1 shows how the recommendations are generated. As input data some information is required: (i) the ConDec-R specification of the problem (cf. Def. 20 on page 39) and (ii) the initial optimized enactment plans (cf. Def. 24 on page 48) generated during the build-time phase. As stated, for a particular timeframe a BP enactment plan for a set of instances (cf. Def. 9 on page 16) is generated. Algorithm 1 starts at the beginning of such a timeframe and lasts until all the planned instances have completed (line 15 in Alg. 1).

Algorithm 1 continuously generates recommendations (line 11) on how to proceed with process execution considering (1) the best available enactment plan (d in Fig. 4.1) meeting the constraints imposed by the constraint-based specification (e in Fig. 4.1), and (2) all events that occurred during process execution (i.e., *allEvents*). This includes (1) the current partial traces of the process instances (c in Fig. 4.1), and (2) the current information about resource availabilities (c in Fig. 4.1), e.g., $(\neg R_{jk}, T)$ means that k-th resource with role j becomes unavailable at time T (cf. Fig. 4.3). In the case that a recommendation is suggested (line 12), the recommendation system sends it to the user (line 13).

[3]For the current work, the durations of the recommendation generation is considered negligible compared to the duration of the process activities.

Algorithm 1: Provide Recommendations

input : ConDec-R Specification *cr*
set<EnactmentPlan> *plans*

```
1  Recommendation rec;
2  Set<Event> allEvents ← ∅;
3  Set<Event> newEvents;
4  int T ← currentTime();
5  repeat
6      if event(newEvents, T) then
7          allEvents ← allEvents ∪ newEvents;
8          plan ← update(cr, plans, allEvents);
9      if optimizerPlan(cr, plans, allEvents)! = null then
10         plan ← optimizerPlan(cr, plans, allEvents);
11     rec ← generateRecommendation(plans, allEvents);
12     if rec! = null then
13         send(rec);
14     T ← currentTime();
15 until ¬CompleteTrace(cr, allEvents) ;
```

As execution proceeds, the BP enactment and the resource availabilities are monitored (f in Fig. 4.1). If there are new events at time T (line 6 in Alg. 1), i.e., activities get started/completed or resources become available/unavailable (g in Fig. 4.1), then the set of events *allEvents*, which includes both the partial trace and the resource availability events, is updated (line 7 in Alg. 1).

Whenever events are updated the Replanning Module (h in Fig. 4.1) analyzes the optimized plans (i in Fig. 4.1) as well as the events. In particular, it checks if the current execution traces match up with any of the optimized enactment plans (and if a recommendation can be suggested) or whether updates of the execution plans are needed (j in Fig. 4.1). In general, updates of the execution plan can become necessary due to deviations (line 8 in Alg. 1), i.e., (i) the execution trace is not part of one of the optimized enactment plans (e.g., the user is not always following the recommendations), (ii) estimates are incorrect (e.g., when activity executions take longer/shorter than estimated, or more or less instances than expected get executed), or (iii) resource availabilities change (i.e., resources become unavailable). Note that not every deviation requires replanning (some examples are given in Sect. 4.3).

Moreover, plan updates are conducted whenever the replanning module finds a solution which is better than the current optimized plans (lines 9 and 10 in Alg.

1). The Replanning Module is continuously searching for a better plan by considering the event log during BP execution, provided that the current plan is not optimal. If plan updates are required, the Replanning Module needs to access the extended constraint-based specification of the process (k in Fig. 4.1) to generate new optimized plans considering both the estimates and the constraint-based specification. If necessary, the replanning, i.e., the generation of new optimized enactment plans, is carried out by applying a constraint-based approach for P&S the BP activities (cf. Chapter 3).

Despite the NP-complexity of the considered problems, in general replanning is less time consuming than initial planning, since most of the information about previous generated plans can usually be reused, and CSP variable values become known as execution proceeds (cf. Sect. 4.4).

A running example illustrating the complete process is detailed in Sect. 4.3.

4.3 A Running Example

In this section, the proposed approach is used for managing recommendations during a hypothetical execution of a running example which represents a travel agency. This agency manages holiday bookings by offering clients the following three services: transport, accommodation, and guided excursions. After the client request is carried out, the agency must write a report which contains the information in answer to the request, which will then be sent to the client. Besides managing the client requests, people who work in the agency must perform further activities related to management, and accounting tasks. The number of client requests (P), management tasks (M) and accounting tasks (A) which must be dealt with during a working day is known at the beginning of the day, hence it is necessary to organize the work considering the estimated work load. The objective to be considered by the agency is to minimize the overall completion time of the daily processes. However, this example can easily be extended in order to consider the optimization of further objective functions, such as cost. In the travel agency, optimized BP enactment plans must be created every day to generate recommendations about the activities to be executed and the correct ordering. The activities which must be executed to deal with the client requests, management and accounting tasks are detailed in Table 4.1. For activities which are executed more than once, each execution must finish before the next execution can start. Moreover, in order to simplify the analyzed problem, all the executions of the same activity require a resource of the same role with the same duration is assumed[4].

[4]Note that the proposed approach can deal with BP activities which require several resources of various kinds of roles, since the considered problems are modeled as scheduling problems with optional activities, where the resources have a discrete capacity.

Table 4.1: BP activities.

ID	Description	Role	Duration
G	The client request is received	R_0	2
AS	A suitable accommodation is searched	R_0	6
GE	Guided excursions are organized	R_1	5
TS	A suitable transportation is searched	R_0	7
WR	A report with the answers to the requests is written	R_1	5
MT	Management task is carried out	R_0	4
AT	Accounting task is carried out	R_1	5

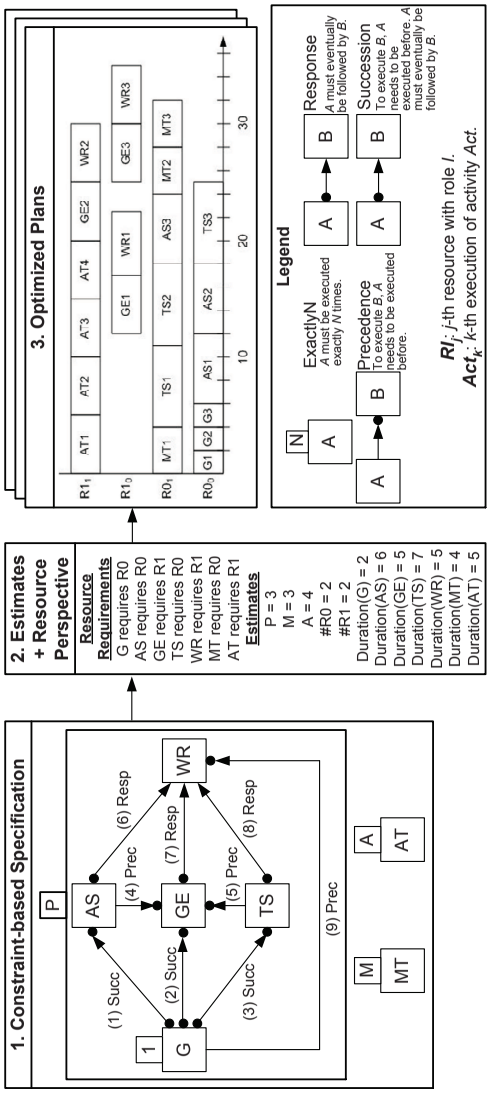


Figure 4.2: Build-time for the Running Example.

4.3.1 Build-time Phase

To solve this problem through the proposed approach, the first step is the creation of the related ConDec specification. The constraint-based specification includes seven activities, G, AS, GE, TS, WR, MT and AT, and several relations (ConDec templates (van der Aalst and Pesic, 2006a)) between the activities (cf. Fig. 4.2(1)): (i) after a client request (G), transport and accommodation search (TS and AS), and guided excursions elaboration (GE), must be processed, and before the execution of these services (AS, TS and GE), the client request (GR) must be received (Relations (1), (2) and (3) in Fig. 4.2(1)); (ii) before preparing the guided excursion (GE), accommodation and transport must be known (Relation (4) and (5) in Fig. 4.2(1)); (iii) any execution of activities related to client requests, i.e., AS, TS and GE, must be reported (Relation (6), (7) and (8) in Fig. 4.2(1)); and (iv) the report cannot be written before a client request (Relation (9) in Fig. 4.2(1)).

In a next step, the constraint-based specification is extended with resource requirements, estimates for the number of instances to be executed, resource availabilities, and the duration of the activities (cf. Fig. 4.2(2)). Lastly, the constraint-based approach is applied to generate optimized enactment plans for the specified problem (cf. Fig. 4.2(3)).

4.3.2 Run-time Phase

Figure 4.3 shows the behavior of the proposed recommendation service when hypothetical process instances with given traces are executed for the constraint-based specification, for three client requests (P = 3), three management tasks (M = 3) and four accounting tasks (A = 4).

At the beginning of the execution, plan P_1 (which has already been generated during build-time considering the estimates for P, M and A) is considered for the generation of the recommendations. Initially, the partial trace for all instances is empty, which can be seen in column Partial Trace, where completed events for activity executions are depicted. Furthermore, for all three client requests (i.e., I_1, I_2 and I_3), G is enabled (reflected by the white bars in columns G_1, G_2, G_3), whereas AS, GE, TS, and WR of instances I_1, I_2 and I_3 are not yet enabled (reflected by the black bars in columns $AS_i, GE_i, TS_i, WR_i, \forall i \in \{1,2,3\}$). Moreover, MT and AT are always enabled since there are not any constraints restricting their execution. Activities AS, GE and TS are not enabled since G must be executed before executing AS, GE and TS due to the succession constraints. In a similar way, activity WR is not enabled since the execution of WR requires a previous execution of G. At time 0, starting execution of G_1 using R_{00}, MT_1 using R_{01}, and AT_1 using R_{11} is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_1 is not enabled anymore. At time 2, G_1 is completed, hence AS_1, TS_1 and WR_1

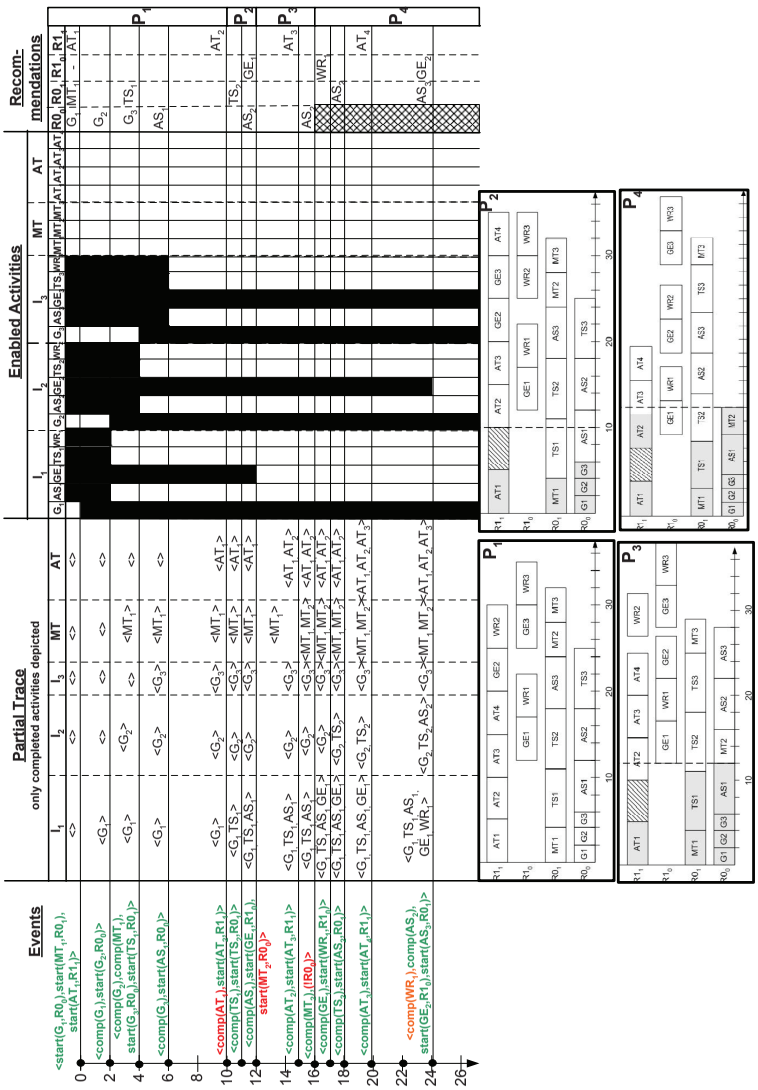


Figure 4.3: Run-time for the Running Example.

becomes enabled, and the partial trace of I_1 contains G_1. Activity GE_1 is not yet enabled since the execution of GE_1 requires a previous execution of both AS_1 and TS_1. Furthermore, at time 2, starting execution of G_2 using R_{00} is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_2 is not enabled anymore. At time 4, G_2 and MT_1 are completed, hence AS_2, TS_2 and WR_2 become enabled. Furthermore, at time 4, starting execution of G_3 using R_{00} and TS_1 using R_{01} is suggested. The user follows the recommendation. Due to `Exactly_1(G)`, G_3 is not enabled anymore. At time 6, G_3 is completed, hence AS_3, TS_3 and WR_3 becomes enabled. Furthermore, at time 6, starting execution of AS_3 using R_{00} is suggested. The user follows the recommendation. At time 10, AT_1 is completed five time units later than expected. Since there was no slack time between AT_1 and

AT_2, P_1 becomes outdated, and the replanning module generates P_2 considering the new conditions.

At time 10, based on P_2 starting execution of AT_2 using R_{11} is suggested. The user follows the recommendation. At time 11, TS_1 is completed. Furthermore, at time 11, starting execution of TS_2 using R_{01} is suggested. The user follows the recommendation. At time 12, AS_1 is completed, hence GE_1 becomes enabled. Furthermore, at time 12, starting execution of AS_2 using R_{00} and GE_1 using R_{10} is suggested. The user decides to partially follow the recommendation, so that, instead of executing AS_2 she decides to start MT_2. After this unexpected decision, P_2 becomes invalid, and the replanning module generates P_3 considering the new conditions.

At time 15, AT_2 is completed. Furthermore, at time 15, based on P_3, starting execution of AT_3 using R_{11} is suggested. The user follows the recommendation. At time 16, MT_2 is completed. Furthermore, at time 16, an unexpected event occurs (i.e., resource R_{00} became unavailable), hence P_3 is no longer valid, and the replanning module generates P_4 considering the new conditions.

At time 17, GE_1 is completed. Furthermore, at time 17, based on P_4, starting execution of WR_1 using R_{10} is suggested. The user follows the recommendation. At time 18, TS_2 is completed. Furthermore, at time 18, starting execution of AS_2 using R_{01} is suggested. The user follows the recommendation. At time 20, AT_3 is completed. Furthermore, at time 20, starting execution of AT_4 using R_{11} is suggested. The user follows the recommendation. At time 24, WR_1 is completed two time units later than expected. Even with the occurrence of this unexpected event, P_4 is still valid due to the slack time between WR_1 and GE_2. Moreover, at time 24, AS_2 is completed, hence GE_2 becomes enabled. Furthermore, at time 24, starting execution of AS_3 using R_{01} and GE_2 using R_{10} is suggested. The user follows the recommendation. The remaining activities are executed as expected by considering P_4.

In this way, the recommendation service supports users of flexible BPMSs during process execution to optimize the overall process performance goals, by considering optimized enactment plans which are updated when necessary.

4.4 Empirical Evaluation

In order to evaluate the effectiveness of the proposed approach, a controlled experiment has been conducted. Section 4.4.1 describes efficient search algorithms for solving the CSP used for the empirical evaluation of the current chapter, Section 4.4.2 describes the design underlying the experiment, and Sect. 4.4.3 shows the experimental results and the data analysis.

4.4.1 Search Algorithms

Once a CSP is modelled, several constraint-based mechanisms can be used to obtain the required solution. In this section, some search algorithms which efficiently deal with CSP-ConDec problems are introduced.

In the current chapter, existing methods are adapted and applied, specifically complete search (Rossi et al., 2006), incomplete search (Rossi et al., 2006), and GRASP (Feo and Resende, 1989, 1995) (cf. Sect. 2.2), for solving the specific considered problems and their suitability for the generation of recommendations is also evaluated.

In general, when optimizing a CSP variable, if a feasible solution which is known exists, the value of the variable to optimize in the known solution can be used for discarding large subsets of fruitless candidates by using upper and lower estimated bounds of the quantity being optimized during the search process (cf. Sect. 3.3.3). Thus, if a known feasible solution S for the problem to solve exists, the objective value for this solution (S^{OCT}) is a valuable information which can be added to the constraint model through the constraint $OCT < S^{OCT}$. Thus, some non optimal candidates, i.e., candidates whose OCT value cannot be less than S^{OCT} in any case, are discarded during the search, and hence increases the efficiency in the search for solutions.

Moreover, in the proposed approach, during the search process, some of the values which only lead to non-feasible solutions, i.e., inconsistent values, are removed from the domains of the CSP variables in order to reduce the search space through maintaining arc consistency (cf. Def. 17 on page 29).

In the proposed approach, the developed filtering rules (cf. Appendix B) and CSP modelling (cf. Sect. 3.3.1) are implemented such that they maintain the arc consistency for all pairs of CSP variables during all the search process.

Proposition 4. *Let S be the best complete solution, i.e., with fewest overall completion time, which can be obtained for a CSP-ConDec problem P by considering certain fixed values for all nt variables ($nt_{Act_1}, nt_{Act_2}, ..., nt_{Act_{\#Act}}$), i.e., $S^{nt(Act_i)} = nt_{Act_i}, \forall i \in \{1 \ldots \#Act\}$. Let S' be the best complete solution which can be obtained for P by considering certain fixed values for all nt variables ($nt_{Act_1}, nt_{Act_2}, \ldots, nt'_{Act_i}, \ldots, nt_{Act_{\#Act}}$, $nt'_{Act_i} = nt_{Act_i} + 1$). Then: $S'^{OCT} < S^{OCT}$ is not possible.*

Proof. Let $P_O = (V, D, R, O)$ be a CSP-ConDec problem (cf. Def. 23 on page 43) related to a ConDec-R process model $CR = (Acts, C, Res)$ (cf. Def. 20 on page 39). Increasing the number of times a repeated activity is executed has different effects depending on the kind of high-level constraints stated in C:

- Case 1: $\nexists c \in C$ of type *Alternate* or *Chain*, i.e., including disjunctions related to existential forms since these relations imply that between each

two executions of a specific BP activity, at least one execution of another specific BP activity must exist (cf. Appendix A). In this case, the CSP which is obtained after instantiating all the *nt* variables can be represented by a precedence graph, i.e., an acyclic directed graph where nodes correspond to activities and there is an arc from A to B if A must precede B. In this way, the fact of increasing the value of any *nt* variable results in including one additional scheduling activity in the previous precedence graph. Therefore, the new CSP, resulted from increasing one of the *nt* variables (i.e., adding a new scheduling activity), can be represented by a precedence graph which extends the previous graph by including the precedence constraints in which the new scheduling activity is involved. Taking into account that $OCT = max(et(Act_{nt(Act)})), \forall Act \in A$, increasing the number of times a repeated activity is executed does not make improving the optimal solution possible.

- Case 2: $\exists c \in C$ of type *Alternate* or *Chain*, i.e., some disjunctions related to the existential forms of alternate and chain templates (cf. Appendix A) exist. These disjunctions can result in having a set of possible alternative precedence graphs *PGs*, so that one of the graphs included in the set *PGs* leads to the optimal solution of the problem. The fact of increasing one of the *nt* variables results in adding a new scheduling activity to the precedence graph (together with the precedence constraints in which this activity is involved in). Moreover, the existential relations can be modified due to adding this new activity. The fact of adding all these new relations between activities implies that each graph which belongs to the new set of precedence graphs PGs' corresponds to a reinforcement of some of the original graphs which belonged to *PGs*, and hence any graph belonging to PGs' can lead to a solution with less OCT value.

□

As stated, the arc consistency for all pairs of CSP variables is maintained during entire the search process. In the proposed approach, after posting all ConDec relations between the BP activities, i.e., adding the arc consistent filtering rules, all the *nt* variables are instantiated to $LB(nt)$. If the resulting CSP is feasible, then the optimal solution for this CSP is also an optimal solution for the original CSP-ConDec problem (Prop. 4). Otherwise, when the resulting CSP is unfeasible, the values of the *nt* variables are increased step by step. In this way, the optimal solution is searched by considering the CSP which is obtained as a result of instantiating *nt* variables in the first feasible solution which is found, i.e., for the fewest feasible values of *nt*. This optimal solution is also the optimum for the original CSP-ConDec problem (Prop. 4). Usually, the instantiation of all *nt* variables to

$LB(nt)$ is feasible due to the arc consistency which is maintained. However, this may not be true for some combinations of *Alternate* or *Chain* relations, and hence, greater values for *nt* variables need to be considered.

After instantiating all the *nt* variables to a fixed value, the search for the optimal solution only entails the consideration of the remaining CSP variables.

Complete Search

As stated, complete search consists of exploring a search tree for the CSP problem which is based on all possible combinations of assignments of values to the CSP variables. In general, both the ordering of instantiation of the variables and the ordering of selection of values for each variable have a great influence on the efficiency of the search process and also on the quality of the solutions which are obtained.

Once the *nt* variables of the repeated activities are instantiated, the considered problem becomes an extension of the Cumulative Job Shop Scheduling Problem (Baptiste et al., 1999), CJSSP. While CJSSP considers sequences of activities related by precedence constraints which require some shared discrete resources and which must be scheduled in order to minimize some objectives, the proposed extension includes further kinds of relations, e.g., alternate or chain (van der Aalst and Pesic, 2006a) which are not pure precedence relations considered in typical scheduling problems. In the presented proposal for performing a complete search, after generating a first feasible solution by using Alg. 5 (detailed later), an efficient method for solving the CJSSPs, named *setTimes* (Le Pape et al., 1994), based on (van Hentenryck, 1999) (cf. (Dynadec, 2011)) is used.

Iterative Bounded Greedy

Due to the NP-complexity of the considered problem, in addition to the complete search, an incomplete search approach, named iterative bounded greedy (IBG), is implemented including randomized components to achieve diversified results (cf. Alg. 2). In Alg. 2, a greedy randomized algorithm (Alg. 5) is used (line 3) for iteratively improving the best solution found (line 4), until a time limit is reached. The best solution over all iterations is returned as the result (line 5).

With the proposed incomplete search, all the solutions can be reached and the search procedure efficiently explores a wide range of solutions from diversified areas of the search space.

Algorithm 2: Iterative Bounded Greedy

input : Set<RepeatedActivities> *repAct*
Set <Template> *templates*
output: Solution *bestSol*

```
1 Solution sol;
2 while Iterative Bounded Greedy stopping criterion not satisfied do
3     sol ← ConstructGreedyRandomizedSolution(repAct, templates);
4     UpdateSolution(sol, bestSol);
5 return bestSol;
```

GRASP-LNS

In addition to complete and incomplete search, a hybrid approach is considered. GRASP (Feo and Resende, 1989, 1995) (cf. Alg. 3) consists on an iterative process in which each iteration includes two phases: (i) a construction phase (line 3), in which a feasible solution is built through Alg. 5, and (ii) a local search phase, i.e., incomplete search (line 4), in which the neighborhood of the generated solution is explored to find a local optimum. The best solution over all GRASP iterations (line 5) is returned as the result (line 6).

Algorithm 3: GRASP-LNS

input : Set<RepeatedActivities> *repAct*
Set <Template> *templates*
output: Solution *bestSol*

```
1 Solution sol;
2 while GRASP-LNS stopping criterion not satisfied do
3     sol ← ConstructGreedyRandomizedSolution(repAct, templates);
4     LocalSearch(sol);
5     UpdateSolution(sol, bestSol);
6 return bestSol;
```

In the current approach, LNS (Pisinger and Ropke, 2010) is used for exploring the neighborhood of current solutions in the GRASP algorithm (line 4 of Alg. 3), resulting in an efficient hybrid technique, named GRASP-LNS. In a LNS algorithm (cf. Alg. 4), in each iteration, a neighborhood is explored with CP trying to improve the current best solution (*bestSol* variable) in the following way: first, part of the current solution is relaxed (line 5 of Alg. 4) so that the domain of some variables is restored to its initial range, while fixing the remaining variables to

their current value; secondly, the restricted problem is re-optimized by using CP with a limit on the number of failures (line 6 of Alg. 4). The best solution over all iterations (line 7 of Alg. 4) is returned as the result (line 8 of Alg. 4). In Alg. 4 the reason for setting a failure limit is to avoid exploring a neighborhood for too long, allowing the search to explore a variety of neighborhoods.

Algorithm 4: LocalSearch

input : Solution *sol*
output: Solution *bestSol*

```
1 PartialSolution pSol;
2 Solution tempSol;
3 bestSol ← sol;
4 while bestSol can be improved AND a failure limit does not occur do
5 |   pSol ← Relax(bestSol);
6 |   tempSol ← Re − optimizeCP(pSol);
7 |   UpdateSolution(tempSol, bestSol);
8 return bestSol;
```

Greedy Generation of a Feasible Solution

As follows, a greedy randomized algorithm which is used for the generation of a feasible solutions is presented. This algorithm is invoked by the different proposed searches, i.e., complete search, iterative bounded greedy, and GRASP-LNS, as explained before. After instantiating all *nt* variables for all the repeated activities, a feasible solution can be quickly generated by Alg. 5.

The main idea of Alg. 5 consists of instantiating the value of certain allowed variables related to a specific P&S activity in each step, i.e., those variables which can be instantiated by taking the P&S activities which have been already instantiated and the set of relations (templates) into account. In line 1 and 2, both the set of activities allowed to be instantiated in the next step and the set of activities previously instantiated are created and initialized. Furthermore, a map which relates each repeated activity *A* to the last execution of *A* which has already been instantiated is created and initialized to 0 (line 3). In each step, the set *allowedActs* is filled with the P&S activities which are allowed to be instantiated next (lines 5-10), by considering that only one execution of each repeated activity is analyzed to be included since the P&S activity related to the i-th execution of *A* can only be instantiated after instantiating the P&S activity related to (i-1)-th execution of *A*. In this way, for each repeated activity *A* (line 6), the index of the P&S activity related to the execution of *A* to be instantiated next is stored in the variable

Algorithm 5: ConstructGreedyRandomizedSolution

```
   input  : Set<RepeatedActivities> repAct
            Set<Template> templates
   output: Solution sol
 1 Set<SchedAct> allowedActs ← ∅;
 2 Set<SchedAct> actAlreadyInstantiated ← ∅;
 3 Map<RepeatedActivities,Integer> instantiatedN(repAct) ← {0,...,0};
 4 repeat
 5     allowedActs ← ∅;
 6     foreach A in repAct do
 7         int nextA ← instantiatedN(A) + 1;
 8         if nextA ≤ nt(A) then
 9             if Allow(A, nextA, templates, actAlreadyInstantiated) then
10                 allowedActs ← allowedActs ∪ A_nextA;
11     if allowedActs ≠ ∅ then
12         SchedAct actToInst ← Select(allowedActs);
13         Instantiate(actToInst, sol);
14         actAlreadyInstantiated ← actAlreadyInstantiated ∪ actToInst;
15         instantiatedN(actToInst) = instantiatedN(actToInst) + 1;
16 until allowedActs == ∅ ;
17 return sol;
```

nextA (line 7). If there is any execution of A which remains to be executed (line 8), then the method *Allow* checks if this activity instance can be instantiated next (line 9), i.e., is enabled (cf. Def. 10 on page 17). In affirmative case, the related P&S activity is included in the set *allowedActs* (line 10). If there is any activity allowed to be instantiated next (line 11), one of these activities is selected (line 12). After the selection of the P&S activity to be instantiated next, the start and the end variables of this selected activity are instantiated to the minimum value of its domains (line 13). These instantiations, in general, result in updating the domain of some CSP variables (the developed filtering rules are in charge of carrying these updates out). Moreover, the instantiated activity is included in the set *actAlreadyInstantiated* (line 14), and its related information is updated (line 15). The lines 5-15 are repeated until a solution is completely constructed (line 16). The generated solution is returned as the result (line 17).

A feasible good solution can be swiftly generated through Alg. 5 by considering different heuristics for the implementation of the *Select* method (line 11 of Alg. 5). In this proposal, for the IBG (cf. Alg. 2) and GRASP-LNS (cf. Alg. 3)

searches, in order to get diversified results each time Alg. 5 is invoked, the *Select* method is implemented (heuristic) so that the activities are selected by considering the probability $\pi(A)$ of selecting an activity A as:

$\pi(A) = \frac{1/r(A)}{\sum_{B \in Acts} 1/r(B)}$

where $r(A)$ denote the rank of A when all the candidates are ranked according to their earliest start time.

On the other hand, for the complete search, the heuristic which is considered selects the activity with the lowest starting time, i.e., the most promising one, since for the considered complete search Alg. 5 is only used for generating an initial solution. In most cases, this initial solution will be improved as the complete search proceed.

Notice that for a feasible combination of *nt* values, Alg. 5 always generates a feasible solution, since the arc consistency is maintained during all the process by the developed filtering rules.

4.4.2 Experimental Design

In this section, the design underlying the experiment is detailed.

Purpose: The purpose of the empirical evaluation is to analyze the behavior of the proposed approach in the generation of optimal enactment plans from extended ConDec specifications (including estimates), in order to test the suitability of the proposed approach for giving recommendations, in terms of performance and quality of recommendations. In particular, to investigate in how far different search algorithms are suitable for solving the considered problems is aimed. Specifically, three search algorithms are tested for solving the generated models (cf. Sect. 4.4.1), i.e., complete search (CP), iterative bounded greedy (IBG), and hybrid search (GRASP-LNS). Since the underlying strategies of the considered search techniques are completely different, to investigate under which circumstances each one is the most suitable for obtaining the solution of certain problems is aimed. Moreover, to find out whether these techniques are complementary is analyzed, and hence whether they can be used in a combined way for obtaining a good solution. The suitability of the proposed approach is tested regarding both (1) build-time, i.e., generation of complete optimized plans before starting the BP enactment; and (2) run-time, i.e., generation of (partial) optimized plans by considering the actual partial trace of the process as the execution of the process proceeds.

Objects: The empirical evaluation considers different ConDec models taking some important characteristics into account in the generation of the models: (i) correctness, i.e., the ConDec models must represent feasible problems without conflicts (i.e., there are some traces that satisfy the model) and must not include

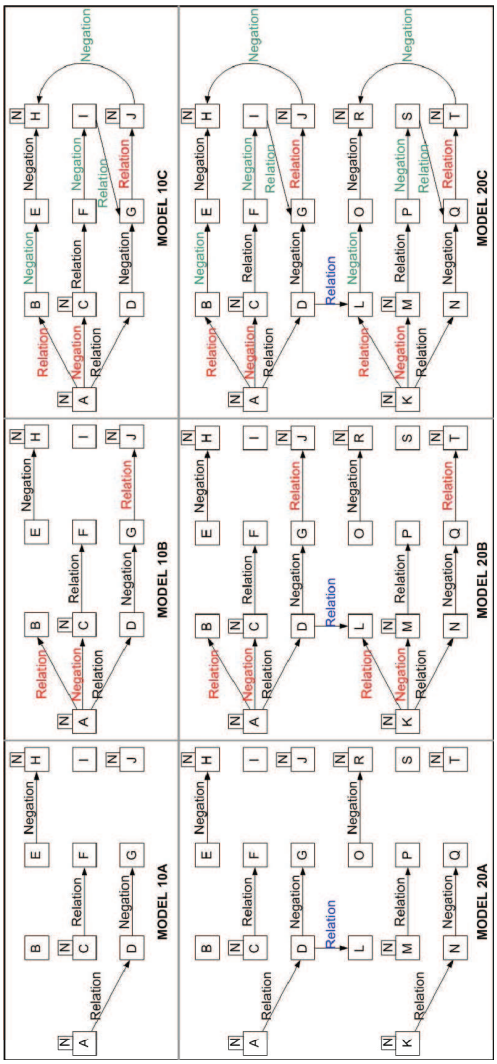


Figure 4.4: Generic ConDec Models.

any dead activities (i.e., none of the traces that satisfies the model contains this activity), (ii) representativeness, i.e., the ConDec models must represent problems which are similar to actual BPs. Consequently, the test models to be of medium-size (i.e., including 10-20 activities) and comprise all three types of ConDec templates, i.e., existence, relation, and negation (cf. Sect. 2.1.2) are required. Overall, 6 generic test models are considered with 10 and 20 activities respectively and a varying number of constraints (cf. Table 4.2). Figure 4.4 shows the ConDec representation of the generic models 10*A*, 10*B*, 10*C*, 20*A*, 20*B*, and 20*C*. During the experiment, the generic relations are then instantiated with concrete constraints leading to different ConDec problems. Since the filtering rules (cf. Appendix B) for the different ConDec constraints significantly vary in their computational

Table 4.2: Generic constraint-based BP models.

Model	#Acts	Description
M10A	10	Includes 10 activities and 4 constraints
M10B	10	Extends M10A by including 3 constraints more
M10C	10	Extends M10B by including 4 constraints more
M20A	20	Includes 20 activities and 9 relations
M20B	20	Extends M20A by including 6 constraints more, similar to M10B
M20C	20	Extends M20B by including 8 constraints more, similar to M10C

complexity (cf. Table 4.3 for the different complexity groups), the considered ConDec problems cover all complexity groups is ensured[5].

In this way, the generic ConDec models presented in Fig. 4.4 are specified by replacing the labels *Relation* and *Negation* with a concrete template of each group. In order to generate all the possible combinations of type and complexity, several types of problems are considered by including templates of the following groups (cf. Table 4.3): {E1, R2, N4}, {E1, R2, N5}, {E1, R2, N6}, {E1, R3, N4}, {E1, R3, N5}, {E1, R3, N6}. Specifically, for each group of templates, a representative item (in bold in Table 4.3) is selected for the tests. Moreover, in the case of *Existence* templates, a value for label N must be established ($N \in \{10, 20, 30, 40, 50\}$ is considered). In addition, different durations and required resources for each BP activity are considered (game, G), since these aspects have a great influence on the complexity of the search of optimal solutions due to the considered problems are an extension of typical scheduling problems. Specifically, 30 instances are randomly generated for each specific ConDec model by varying activity durations between 1 and 10 and role of required resources between $R1$ and $R2$.

Regarding the number of available resources, in turn, for all the generated test models, two available resources of two kinds of roles are considered.

Independent Variables: Considering the generated test models and search algorithms (cf. Sect. 4.4.1), Table 4.4 depicts the independent variables which are considered for the empirical evaluation.

Response Variables: As stated, the suitability of the proposed approach is tested regarding both build-time and run-time phases, analyzing different response variables.

1. Build-time phase. For the build-time evaluation, a 5-minutes time limit is established, since, in general, the initial optimized plans are generated prior

[5]Notice that in this chapter only the basic ConDec templates introduced in (van der Aalst and Pesic, 2006a) are considered for the empirical evaluation.

Table 4.3: Type and Complexity of ConDec-R Filtering Rules.

Template	Type	Complexity	Group
Existence(N,A)	**Existence**	$\Theta(1)$	**E1**
Absence(N,A)	Existence	$\Theta(1)$	E1
Exactly(N,A)	Existence	$\Theta(1)$	E1
Responded Existence(A,B)	Relation	$O(n)$	R2
CoExistence(A,B)	Relation	$O(n)$	R2
Precedence(A,B)	Relation	$O(n)$	R2
Response(A,B)	**Relation**	$O(n)$	**R2**
Succession(A,B)	Relation	$O(n)$	R2
Alternate Precedence(A,B)	Relation	$O(n \times nt^3)$	R3
Alternate Response(A,B)	**Relation**	$O(n \times nt^3)$	**R3**
Alternate Succession(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Precedence(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Response(A,B)	Relation	$O(n \times nt^3)$	R3
Chain Succession(A,B)	Relation	$O(n \times nt^3)$	R3
Responded Absence(A,B)	Negation	$O(n)$	N4
Negation Response(A,B)	**Negation**	$O(n)$	**N4**
Negation Alternate Precedence(A,B)	Negation	$O(n \times nt^2)$	N5
Negation Alternate Response(A,B)	**Negation**	$O(n \times nt^2)$	**N5**
Negation Alternate Succession(A,B)	Negation	$O(n \times nt^2)$	N5
Negation Chain Succession(A,B)	**Negation**	$O(n \times nt^3)$	**N6**

to BP enactment, i.e., the search algorithms can be run for long time. The optimality of the initially generated plans is considered as a relevant aspect for giving good recommendations. Furthermore, the time spent for obtaining the optimal plans is a rather relevant response variable to be analyzed. The performance measures depicted in Table 4.5 are studied.

2. Run-time phase. For the run-time evaluation, a 5-seconds time limit is established since the search algorithms for generating optimized plans during run-time swiftly need to find a suitable solution due to the user is probably expecting for a recommendation. Therefore, the quality of the generated plans during run-time is an important aspect to be analyzed in order to check the suitability of the proposed approach. The performance measures depicted in Table 4.6 are studied.

   The **quality** of a solution S' regarding S is stated by: S^{OCT}/S'^{OCT}.

For both build-time and run-time evaluation, the results which are obtained by each search technique are compared in order to obtain information about which technique is better in general, or to solve some specific kinds of problems.

Table 4.4: Independent variables.

Name	Description	Values
Model	Generic ConDec model (cf. Fig. 4.4)	{*M10A*, *M10B*, *M10C*, *M20A*, *M20B*, *M20C*}
Relation	Value for the label *Relation* in the specific ConDec model	{*Response*, *AlternateResponse*}
Negation	Value for the label *Negation* in the specific ConDec model	{*NegationResponse*, *NegationAlternateResponse*, *NegationChainSuccession*}
N	Value for the label *N* in the specific ConDec model	$\{10, 20, 30, 40, 50\}$
G	Game containing duration and required resource for each BP activity	$\{1, 2, \ldots, 30\}$
Search	Search algorithms used for solving the generated models	{*CP*, *IBG*, *GRASP-LNS*}

Experimental Design: For each of the 6 generic models, 150 problem instances are generated considering different values for variable N (5 values) and G (30 values) using each of the search algorithms. The response variables are then calculated by considering average values for all 150 problem instances[6].

Experimental Execution: The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running on Windows XP. In order to solve the constraint-based problems, the system COMET (Dynadec, 2011) is used, which

[6]However, notice that problems stated by a simple model with a high value for *N* can entail a higher complexity than complex models with lower value for *N*.

Table 4.5: Response variables in build-time.

Name	Description
%*Opt*	Average percentage of optimal solutions which are found by each technique (cf. Fig. 4.5(a))
$\%Opt_{ANY}$	Average percentage of optimal solutions which are found by any technique (Table 4.8)
T_{Opt}	Average time for getting optimal solutions for each technique, considering the cases in which the optimal solution is found (cf. Fig. 4.5(b))
%*Best*	Average percentage of cases in which a specific technique gets a better solution (i.e., less overall completion time) than the other ones (cf. Fig. 4.5(c))

Table 4.6: Response variables in run-time.

Name	Description
$\%Q_X, X \in \{0,25,50,75\}$	Average quality of solutions which are found by each technique regarding the best solution which is known for each problem after X% of the optimized plan is executed (cf. Fig. 4.6)
$\%Q_{ANY}$	Average quality of solutions which are found by any technique regarding the best solution which is known for each problem after 0%, 25%, 50% and 75% of the optimized plan is executed (cf. Fig. 4.7)

Table 4.7: ID for the considered Relation-Negation.

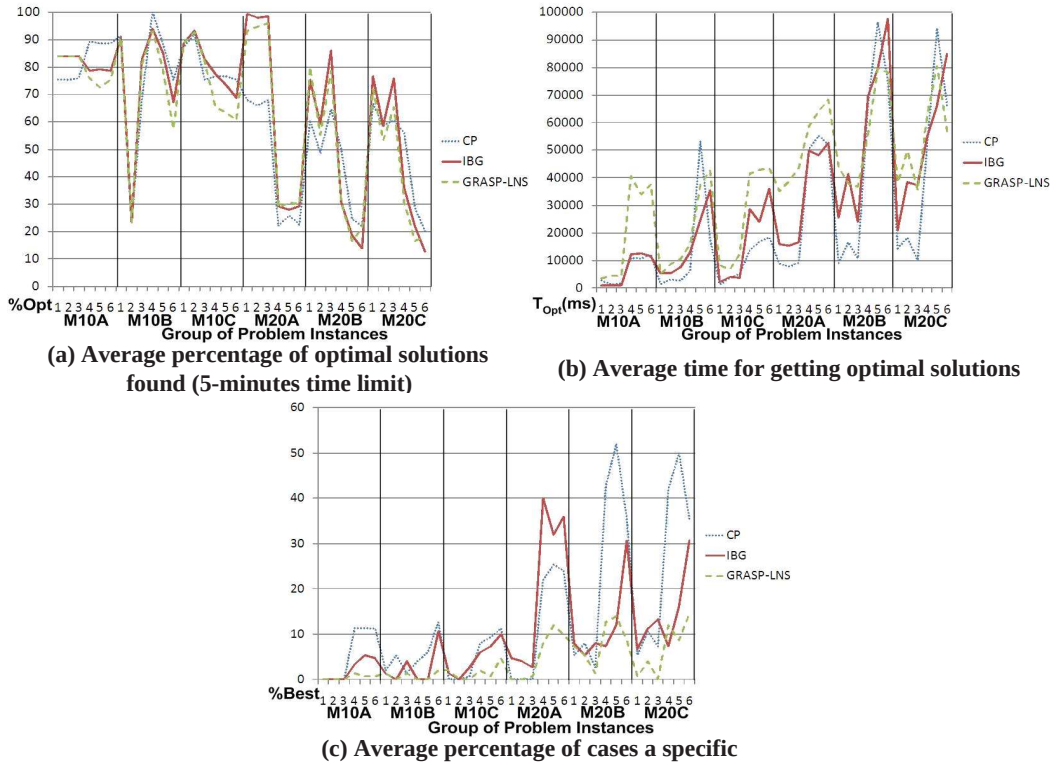
ID	Relation-Negation
1	Response - Negation Response
2	Response - Negation Alternate Response
3	Response - Negation Chain Succession
4	Alternate Response - Negation Response
5	Alternate Response - Negation Alternate Response
6	Alternate Response - Negation Chain Succession

is able to generate high-quality solutions for highly constrained problems in an efficient way.

4.4.3 Experimental Results and Data Analysis

Build-time. For the experiments of the build-time phase, the constraint-based search algorithm is run until a 5-minutes CPU time limit is reached.

Figure 4.5 shows for each search algorithm: (a) the average percentage of optimal solutions which are found, (b) the average time for getting optimal solutions, considering the cases in which the optimal solution is found, and (c) the average percentage of cases in which a specific technique gets a better solution (i.e., less overall completion time) than the other ones, (response variables $\%Opt$, T_{Opt}, and $\%Best$ respectively) versus the problem to be solved. For all the tables and figures presented in this section the considered problems are grouped according to the generic model, i.e., $M10A$, $M10B$, $M10C$, $M20A$, $M20B$, $M20C$, and the response variable is represented versus specific problems which are obtained after instantiating the relations and negations of the generic model as stated in Table 4.7, i.e., 6 specific problems ($\{1,\ldots,6\}$) for each generic model are considered.



(a) Average percentage of optimal solutions found (5-minutes time limit)

(b) Average time for getting optimal solutions

(c) Average percentage of cases a specific technique gets a better solution than the other ones

Figure 4.5: Experimental results regarding build-time phase.

Figure 4.5(a) shows that CP search reaches the optimum solution for a greater number of problems than IBG and GRASP-LNS when the complexity of the problems is low, i.e., for models M10A, M10B and M10C. As the complexity of the problem increases, IBG and GRASP-LNS reach the optimal solutions for a greater number of problems than CP, i.e., M20A, M20B and M20C, presenting the highest difference for model M20A. These results can be explained by the fact that CP is a good strategy for getting the optimal solutions for problems which present a low complexity, since for small problems almost entire the search space can be explored. However, problems with high complexity, an exhaustive search is not a good strategy for searching the optimal solution, since the search area explored by CP is not diversified enough and hence only a part of the possible solutions is analyzed in a limited time. Thus, incomplete techniques are better in general. Furthermore, the evaluation shows that in most cases, IBG outperforms GRASP-LNS. This result can be explained by the fact that the considered IBG is implemented in an efficiency way due to the considered heuristic, and GRASP techniques require greater parameter tuning (Feo and Resende, 1989, 1995), and hence more tests need to be done to get the best setting for this technique.

Table 4.8: Average percentage of optimal solutions found in 5-minutes time limit when considering all techniques.

Relation-Negation	M10A	M10B	M10C	M20A	M20B	M20C
1 Resp.-Neg. Resp.	84	95.33	88.66	99.33	87.33	78.66
2 Resp.-Neg. Alt. Resp.	84	23.33	93.33	98	62	67.33
3 Resp.-Neg. Chain Succ.	84	86	83.33	99.33	93.33	87.33
4 Alt. Resp.-Neg. Resp.	89.33	100	80	41.33	50.66	58.66
5 Alt. Resp.-Neg. Alt. Resp.	89.33	92.66	77.33	43.33	30	32
6 Alt. Resp.-Neg. Chain Succ.	88.66	76.66	80.66	44	30	26.66

On the other hand, regarding the results which can be obtained by combining all the search algorithms, Table 4.8 shows the average number of optimal solutions which are found by any of the techniques (response variable $\%Opt_{ANY}$) versus the problem to be solved. It can be seen that for models $M10A$ and $M10C$, the percentage of optimal solution which are found is rather high (more than 77% in all cases), regardless of the relations which are given between the activities. However, for models $M20A$, $M20B$, and $M20C$, this measure highly depends on the relations which are given between the BP activities, finding the fewest values when Alternate Response (relations 4-6) is given. On the other hand, for relations 1 and 3, the percentage of optimal solutions which are found is rather high (more than 78% in all cases), regardless of the specific model. However, for relations 4, 5 and 6, this measure highly depends on the model, finding the fewest values for models of 20 activities. Taking these results into account, in general, the specific *Relation* and the number of activities of the BP model seems to be much more influential than the other considered aspects. Furthermore, the average value for all cases is 72.94%, minimum value is 23.33%, and maximum value is 100%, which can be considered rather good results.

Figure 4.5(b) shows that for all search algorithms, for the same model the response variable greatly increases as the complexity of the filtering rules increases. Furthermore, for the same filtering rules the response variable greatly increases as the complexity of the models increases. Moreover, in most cases, *CP* reaches the optimum swifter than the other techniques. In turn, *GRASP*, seems to be the slowest of the three techniques. For *CP* search, most of these optimums are obtained for the simplest models (cf. Fig. 4.5(a)), and hence, less time is required. In general, it can be seen that the average time for getting optimums is less than 100 seconds, which can be considered a rather good result.

Figure 4.5(c) shows that for all search algorithms, for the same model the response variable increases as the complexity of the filtering rules increases, i.e., all techniques present a similar behavior regarding getting optimums for low-

complex filtering rules and this behavior highly differs as the complexity of the filtering rules increases. Furthermore, for the same filtering rules the response variable greatly increases as the complexity of the models increases, i.e., all techniques present a similar behavior regarding getting optimums for low-complex models and this behavior highly differs as the complexity of the models increases. Therefore, the use of the considered techniques in a coordinated way is highly recommendable for complex problems, since their behavior is highly different and their result can be combined for obtaining solutions of high quality.

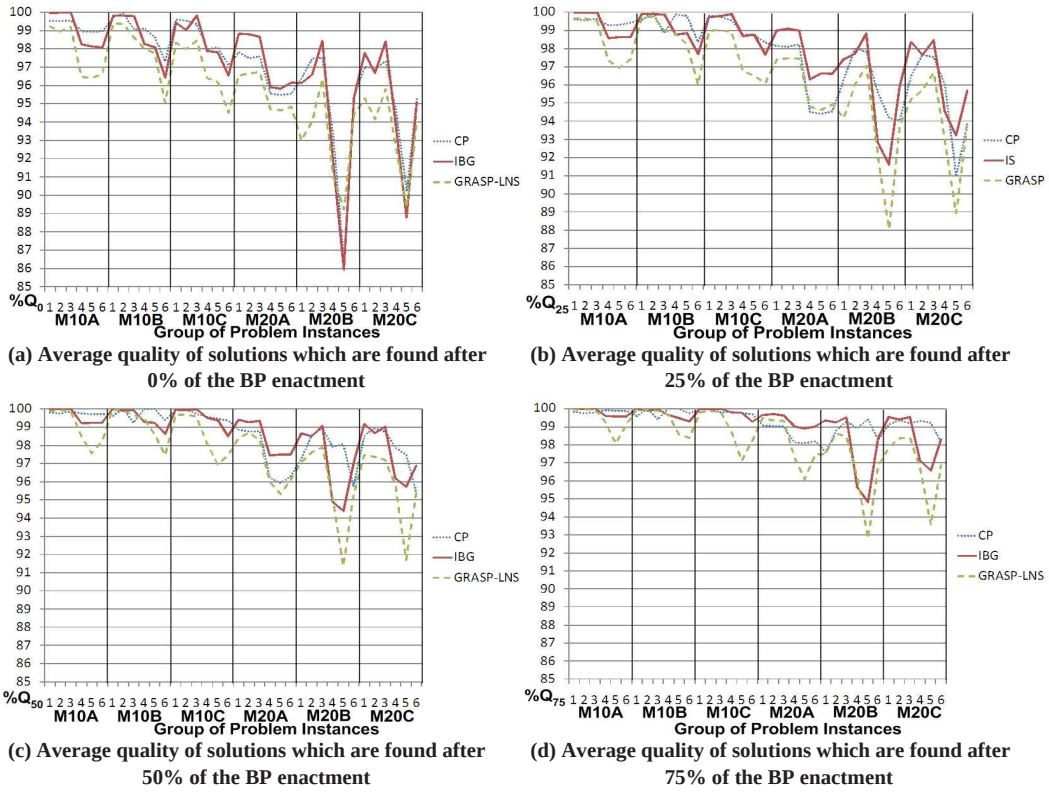
In brief, various conclusions can be drawn after analyzing the results:

- CP search obtains better results than IBG and GRASP-LNS for simple problems (cf. Fig. 4.5(a)).
- IBG and GRASP-LNS searches obtain better results than CP for complex problems (cf. Fig. 4.5(a)).
- The percentage of optimum solutions is very high for almost all cases when considering all techniques (Table 4.8).
- The average time for getting optimums is quite low (cf. Fig. 4.5(b)).
- The use a combination of all techniques in a coordinated way highly increases the quality of the solutions, specially for complex problems.

Run-time. For the experiments of the run-time phase, the constraint-based search algorithm is run until a 5-seconds CPU time limit is reached.

Figures 4.6 and 4.7 show the average quality of solutions which is found when compared with the best known solution for each problem, after 0%, 25%, 50%, and 75% of the BP optimized enactment plan is executed (response variables $\%Q_0$, $\%Q_{25}$, $\%Q_{50}$, $\%Q_{75}$, $\%Q_{ANY}$) versus the problem to be solved. When comparing the different search techniques, similar results compared to the build-time evaluation are obtained due to the previously explained reasons. Moreover, for all search algorithms, it is possible to see that for the same model the quality decreases as the complexity of the filtering rules increases, obtaining the lowest values for relations 5 and 6 in most cases. Furthermore, for the same filtering rules the quality decreases as the complexity of the models increases, obtaining the lowest values for models M20B and M20C. It should be emphasized that, as stated, in general replanning is less time consuming than initial planning, since CSP variable values become known as execution proceeds. Therefore, the quality of the solutions increases as BP execution proceeds (cf. Fig. 4.7).

In general, the quality of the solutions which is obtained applying each technique individually is quite good (greater than 86% for all cases). However, the use



(a) Average quality of solutions which are found after 0% of the BP enactment

(b) Average quality of solutions which are found after 25% of the BP enactment

(c) Average quality of solutions which are found after 50% of the BP enactment

(d) Average quality of solutions which are found after 75% of the BP enactment

Figure 4.6: Experimental results regarding run-time phase.

of all techniques in a coordinated way increases the quality of the solutions even further (greater than 92% for all cases), specially for complex problems (cf. Fig. 4.7).

4.5 Discussion and Limitations

One advantage of the presented proposal is that the recommendation service is based on optimized enactment plans which are generated by P&S all BP activities, hence it allows for a global optimization of the performance goal. In addition, this approach allows modelling the considered problems in an easy way, since the considered declarative specifications are based on high-level constraints. Furthermore, BPs are specified in a declarative way, which is an important step towards the flexible management of BPMSs. Moreover, the proposed approach, as extension of other similar works (Schonenberg et al., 2008; Haisjackl and Weber, 2010), considers the resource perspective besides the control-flow perspective, hence greater optimization can be obtained. Additionally, in order to consider de-

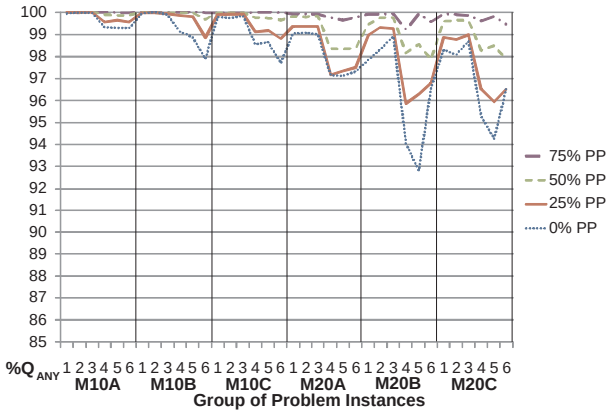


Figure 4.7: Average quality of solutions which are found by any technique after 0%, 25%, 50% and 75% of the BP enactment.

viations in the estimates, the optimized plans can be updated if necessary, allowing to react to changes in a quick and flexible way.

On the other hand, the proposed approach presents some drawbacks. First, the business analysts must deal with a not standard language for the declarative specification of BPs, therefore a period of training is required to let the business analysts become familiar with ConDec specifications. Secondly, the optimized plans are generated by considering estimated values for activity durations and resource availabilities, hence the presented proposal is only appropriate for processes for which the duration of the activities and resource availabilities can be estimated. However, as stated earlier, in the enactment phase, the replanning module can update the current plan by considering the current values of the estimates. Moreover, the considered constraint-based specifications deal with both control-flow and resource perspectives, but do not consider the non-temporal data perspective. It is intended to consider this aspect in our future work. Furthermore, in this work, only the basic ConDec templates introduced in (van der Aalst and Pesic, 2006a) are considered for the empirical evaluation. In Chapter 5 some extensions to the basic ConDec templates are considered in the empirical evaluation, e.g., branched templates (i.e., high-level relations given between more than two activities).

4.6 Related Work

Other authors use AI for exception handling during run-time (Russell et al., 2006; Friedrich et al., 2010); our proposal, in turn, use a constraint-based modelling approach and make suggestions to users. Furthermore, unlike the proposed approach, the related proposals rely on imperative specifications. Notice that inte-

grating AI and BPM is not new. However, the use of AI P&S techniques for giving user recommendations is a new application area.

Related to decision support systems, (Özbayrak and Bell, 2003) develops a knowledge-based decision support system (KBDSS) for short-term scheduling in flexible manufacturing systems (FMS). Unlike the presented proposal, (Özbayrak and Bell, 2003) considers the optimization of the efficient use of the machining cells by using a knowledge-based expert system in order to support the decision making process. Moreover, (Chaturvedi et al., 1993) manages multiple objectives in a hierarchical way. While both approaches (Özbayrak and Bell, 2003; Chaturvedi et al., 1993) are based on the knowledge which has been learnt in prior executions, the proposed approach is based on a constraint-based approach for the generation of optimized plans by considering estimated values. Furthermore, (Özbayrak and Bell, 2003; Chaturvedi et al., 1993) do not consider declarative process specifications. In a related way, (Thompson and Goodale, 2006) addresses the scheduling of a group of employees which present different productivity considering the stochastic nature of customer arrivals and, unlike the presented proposal, replans during run-time when estimates are incorrect.

Related to user recommendations, (Vanderfeesten et al., 2008) generates recommendations to select the next step to optimize some objective functions. While (Vanderfeesten et al., 2008) is based on a product data model for generating the recommendations, the proposed approach is based on optimized enactment plans. In addition, (Schonenberg et al., 2008; Haisjackl and Weber, 2010) support users during the execution of declarative process models by selecting among enabled activities. Unlike the presented proposal, the recommendations are based on similar past process executions, instead of optimized plans which are generated through a constraint-based approach. Moreover, while the proposed approach allows for a global optimization of the performance goal, the recommendations described in (Schonenberg et al., 2008; Haisjackl and Weber, 2010) bear the risk of creating local optimums. Furthermore, (Vanderfeesten et al., 2008; Schonenberg et al., 2008; Haisjackl and Weber, 2010) only consider the control-flow perspective, while the proposed approach also deals with the resource perspective. All of the previously mentioned approaches optimize the execution of single process instances. The proposed approach, in turn, provides recommendations for optimizing the execution of several instances.

Chapter 5

From Optimized BP Enactment Plans to Optimized BP Models

5.1 Introduction

5.1.1 Motivation

The Process Design & Analysis phase plays an important role in the BPM life cycle for any improvement initiative, since it greatly influences the remaining phases of this cycle. In addition, also run-time aspects are important for BP improvement, e.g., resource allocation and scheduling may significantly impact business process performance.

Traditionally, two steps are considered in the BP Design & Analysis phase to create a BP model (Aguilar-Savén, 2004). The first step consists of analyzing the business process, e.g., by interviewing stakeholders (people involved in the process), in order to draw an initial BP model (as-is model). Secondly, in order to improve this initial model, different techniques can be employed like simulation (Barjis and Verbraeck, 2010) or BP redesign (Reijers, 2003), resulting in the generation of a to-be model. Typically, different quality dimensions like time, cost, flexibility and quality can be differentiated (Reijers, 2003) between which trade-off decisions have to be made when creating a BP design. Once a certain process design has been chosen and implemented, business processes are executed according to this design[1]. During process execution, scheduling decisions are then typically made by the BPM systems (BPMSs), by automatically assigning activities to resources (Russell et al., 2005).

In most cases, the overall process of creating a BP model is carried out manually by business analysts, who specify the BP information through an imperative

[1] In this chapter, the assumption that there is a BPMS executing the BPs is made.

language by choosing between several different alternative designs the one which best meets the performance goals of the organization. Therefore, analysts must deal with several aspects in order to generate a suitable BP model, such as: (1) the activity properties, e.g., activity duration, resource or role which is required for activity execution, (2) the relations between the activities, i.e., control-flow of the BP, and (3) the optimization of several objectives, e.g., minimization of completion time. The manual specification of imperative BP models can therefore form a very complex problem, i.e., it can consume a great quantity of time and human resources, may cause certain failures, and may lead to non-optimized models since the tacit nature of human knowledge is often an obstacle to eliciting accurate process models (Ferreira and Ferreira, 2006).

Moreover, the result of process modelling is typically a static plan of action, which is difficult to adapt to changing procedures or to different business goals (Ferreira and Ferreira, 2006).

Not only the process design, but also the allocation of resources during process execution has a great influence on process performance. However, scheduling is only considered to a limited degree in existing BPMSs, and is typically done during run-time by assigning activities to resources.

5.1.2 Contribution

To support process analysts in the definition of optimized BP models, a method for automatically generating imperative BP models using AI planning techniques from constraint-based specifications (which describe the activities to be executed as well as constraints to be considered) is suggested. Unlike imperative models, the specification of process properties in a declarative way, e.g., using a constraint-based specification, only requires process designers to state what has to be done instead of having to specify how it has to be done. The proposed AI-based approach, in turn, is in charge of determining how it is to be done in order to satisfy the constraints imposed by de constraint-based problem specifications, and to attain an optimization of certain objective functions (e.g., minimization of completion time). For this optimization, scheduling is done on a short-term basis by considering the optimization of a set of instances, which is typically not considered in most BPMSs.

Figure 5.1 provides an overview of the proposed approach. Considering the constraint-based specifications as a starting point (cf. Fig. 5.1(1)), enactment plans can automatically be generated (cf. Fig. 5.1(2)), as detailed in Chapter 3. The generated enactment plans are then automatically translated into a BPMN model (BPMN, 2011) (cf. Fig. 5.1(3)), which can be then further improved by a business analyst, where necessary. In most cases, BPMN models can be translated into an execution language (Ouyang et al., 2006), such as BPEL (BPEL, 2007),

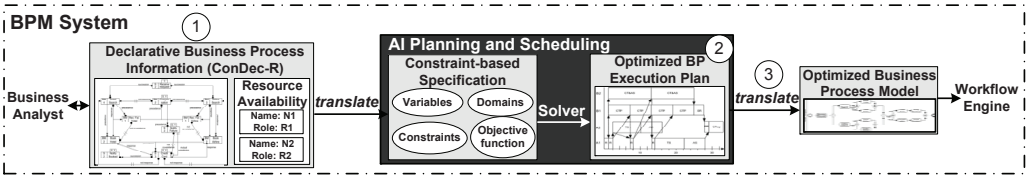


Figure 5.1: AI P&S techniques for the generation of optimized BP models.

which enables BP designs to be deployed into BPM systems and let their instances be executed by a BPM engine.

The main contributions of this chapter can be summarized as follows:

- Automatic generation of optimized business process models in BPMN from optimized BP enactment plans (cf. Sect. 5.2, Step 3 in Fig. 5.1).

- Validation of the proposed approach through the analysis of different performance measures related to a range of test models of varying complexity (cf. Sect. 5.4). The proposed empirical evaluation deals with some extensions to the basic ConDec templates, e.g., branched templates (i.e., high-level relations given between more than two activities, cf. Sect. 3.2).

In this way, the automatic generation of BP models simplifies the BP design phase by facilitating the human work in most cases, preventing failures in the developed BP models, and enabling better optimization to be attained in the enactment phase. Furthermore, the proposed approach is suitable for adapting BP models to changing procedures or to different business goals. This is since imperative BP models can dynamically be generated from static constraint-based specifications just before starting the BP enactment, once some values for the enactment parameters, e.g., resource availabilities, are known. Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way (as will be demonstrated in Sect. 5.4). Therefore, a wide study of several aspects can be carried out, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by generating several kinds of alternative specifications.

The remainder of this chapter is organized as follows: Section 5.2 details the approach for generating optimized BPMN models from optimized BP enactment plans, Section 5.3 explains a running example, Section 5.4 deals with the evaluation of the proposed approach, Section 5.5 presents a critical discussion of the advantages and limitations of the proposed approach, and finally, Section 5.6 summarizes related work.

5.2 From Optimized Enactment Plans to Optimized Business Process Models

Section 3.3 has described how optimized BP enactment plans can be generated from ConDec-R specifications. This section describes how a BPMN model can be generated from an optimized BP enactment plan. Notice that the imperative BP model can be generated just before starting the BP enactment by considering the actual values of the resource availabilities.

The generated BPMN model includes the same activities to be executed in the same ordering and also using the same resources than the enactment plan. For each role in the BP enactment plan, a BPMN pool (cf. Def. 26) is created, which contains as many lanes as number of available resources for that role.

Definition 26. *A **BPMN pool** $BPMNPool = (role, \#role)$ is a pool of a BPMN model, which is composed of $\#role$ lanes.*

Moreover, for each scheduling activity in the BP enactment plan a BPMN activity (cf. Def. 27) is created. Additionally, one start and one end activities are included in the BPMN model.

Definition 27. *A **BPMN activity** $BPMNAct = (pool, lane, dur, st)$ is an activity of a BPMN model placed in the lane named $lane$ of the pool named $pool$, with duration dur and start time st.*

One of the most important aspects to be considered for the generation of optimized BPMN models are the precedence relations between the BPMN activities (scheduling activities, cf. Def. 22 on page 43). For establishing these precedence relations the values for the start and the end times of the scheduling activities in the enactment plan are considered. These precedence relations are then used as a basis for generating BPMN models (cf. Def. 31) from BP enactment plans. Some related definitions are given below:

Definition 28. *In a BP enactment plan regarding a CSP solution S, a scheduling activity a_i is a **predecessor** of another scheduling activity b_j, $a_i \in predecessors(b_j)$, if the relation $S^{et(a_i)} \leq S^{st(b_j)}$ holds due to resource or template relations.*

Definition 29. *In a BP enactment plan, a scheduling activity a_i is a **direct predecessor** of another scheduling activity b_j, $a_i \in DP(b_j)$, if $a_i \in predecessors(b_j) \wedge \nexists c_k \in predecessors(b_j) \mid a_i \in predecessors(c_k)$.*

Definition 30. *In a BP enactment plan, a scheduling activity a_i is an **indirect predecessor** of another scheduling activity b_j, $a_i \in IP(b_j)$, if $a_i \in predecessors(b_j) \wedge \exists c_k \in predecessors(b_j) \mid a_i \in predecessors(c_k)$.*

Definition 31. *A **BPMN model** $BPMN = (Pools, Activities, Predecessors)$ related to a ConDec-R process model $CR = (Acts, C_{BP}, Res)$ (cf. Def. 20 on page 39) and to a solution S of the related CSP-ConDec problem (cf. Def. 23 on page 43) is a BP model specified through the BPMN language, where:*

- *The set of pools is composed by:*
  $Pools = \{BPMNPool(role, \#role), (role, \#role) \in Res\}$.
- *The set of activities is composed by:*

$$\begin{aligned} Activities = & \\ & \{BPMNAct(role(a), S^{res(a_i)}, dur(a), S^{st(a_i)}), a \in Acts, i \in [1..nt(a)]\} \\ & \cup \{start = BPMNAct(P_0, L_0, 0, 0)\} \\ & \cup \{end = BPMNAct(P_0, L_0, 0, max_{a \in Acts, i \in [1..nt(a)]} et(a_i))\}. \end{aligned}$$

- *The set of predecessors is composed by:*

$$\begin{aligned} Predecessors = & \\ & \{(start, a_i), a \in Acts, i \in [1..nt(a)], S^{st(a_i)} = 0\} \cup \{(a_{nt(a)}, end), \\ & a \in Acts, \nexists b_i, i \in [1..nt(b)], b \in Acts, \mid a_{nt(a)} \in predecessors(b_i)\} \cup \\ & \{(b_i, c_j), i \in [1..nt(b)], b \in Acts, j \in [1..nt(c)], c \in Acts, \mid b_i \in DP(c_j)\}. \end{aligned}$$

In this way, the precedence relations between activities are stated so that:

- The start activity is predecessor of all scheduling activities whose *st* value is equal to 0.
- The activities which are not predecessors of any other activity, are predecessor of the end activity.
- In general, one activity b_i is predecessor of another activity c_j iff b_i is direct predecessor of c_j.

The set *Predecessors* is represented in the BPMN model by BPMN connections between a source activity a_i and a sink activity b_j, in the case that a_i is the only predecessor of b_j, or by a parallel merging gateway between a set of source activities *Sources* and a sink activity b_j in the case that b_j has more than one predecessor.

The pseudocode and complexity analysis of the algorithms which were developed for generating BPMN models from optimized BP enactment plans are included in **Appendix C**.

5.3 A Running Example

In this section, a running example, the travel agency problem, is developed in order to clarify the overall proposed approach. First, the proposed problem is detailed (cf. Sect. 5.3.1), together with its ConDec-R specification (cf. Sect. 5.3.2). The generated optimized enactment plan and the corresponding BP model representation are then shown and explained (cf. Sect. 5.3.3). The running example deals with a set of representative templates in order to illustrate various kinds of relations which can be given between activities of business processes.

5.3.1 The Travel Agency Problem

The analyzed running example represents an agency which manages holiday bookings by offering clients the following three services: transport, accommodation, and guided excursions. For some of the services the travel agency can ask a travel company to help in managing some client requests. After all the client requests are carried out, the agency must write a report which contains the information in answer to the requests, which will then be sent to the clients. For efficiency reasons, the agency creates only one report a day, hence it must be written after all the requests are carried out.

The activities which can be executed in order to deal with the client requests are detailed in Table 5.1.

Assume that the travel agency wants to minimize the response time for the clients (end time of the client-report activity). For this the agency not only considers the number of client requests (#P), which is known at the beginning of each working day, but also the number of resources available in the agency (role A, #A) and the number of available resources in the company (role B, #B).

5.3.2 ConDec-R Specification for the Travel Agency Problem

In order to solve this problem through the proposed approach, the first step is the creation of the related ConDec-R specification (cf. Fig. 5.2). Since a ConDec-R specification contains two parts which are independent, these parts can be specified separately fostering their reuse:

- Information about the BP activities (required resources, durations, as well as unary templates) and the high-level relations which are given between the BP activities[2] (cf. Fig. 5.2A; Table 5.2).
- Information about the roles and available resources (cf. Fig. 5.2B).

[2] ConDec-R relations are detailed in Section 3.2.

Table 5.1: Activities of the travel agency problem.

Id	Description	Constraints	Role	Dur.
G	The client request is received by the agency	The following client request cannot be received until the current request has **started** to be processed (both trip plan and transport search have started)	A	1
TS	The agency searches for a suitable transportation	Can only be executed after *G* has completed, and if the agency and not the company deals with the search of transport for this request	A	8
AS	The agency searches for suitable accommodation	Must be executed after each TS	A	6
TP	The agency organizes a trip plan	Can only be executed after *G* has completed, if the agency and not the company deals with the creation of the trip plan for this request	A	5
CT&AS	The company searches for transport and accommodation	Can only be executed after *G* has completed, if the company and not the agency deals with the search of transport for this request	B	12
CTP	The company creates a trip plan	Can only be executed after *G* has completed, if the company and not the agency deals with the creation of the trip plan for this request	B	6
SReport	The company sends a report to the agency which includes information about all the trip requests	Must be executed after all the activities related to the client request which are carried out by the company have finished, when there is at least one client request	B	3
RReport	The agency receives the report with all the trip requests	Must be executed after each *SReport*	A	1
CReports	The agency writes a report which includes information about all handled requests	Must be executed after having completed all activities. It is executed only once	A	4

5.3.3 Optimized Enactment Plan and Optimized BP Model for the Travel Agency Problem

Using the ConDec-R specification of Fig. 5.2, the related constraint problem is generated and solved through the constraint-based proposal which is described in Sect. 3.3, resulting in an optimized enactment plan for the travel agency problem. This plan is used for the generation of an optimized BP model.

As commented, in order to define an instance (#P, #A, #B) for the travel agency problem, the following parameters must be stated: number of client requests (#P), number of resources available in the agency (#A), and number of resources available in the company (#B).

In this section, two instances are studied as illustrations, specifically Problem 1 defined by (#P = 4, #A = 1, #B = 1), and Problem 2 defined by (#P = 4, #A = 2, #B = 2). For Problem 1, Fig. 5.3 shows both the optimized Gantt chart (overall completion time = 47) and the BP model which are obtained through the proposed approach. It can be seen that regarding the first client request, depicted by G1 in the Gantt diagram), the trip plan is created by the agency (TP1 activity), while the transport and accommodation search are carried out by the company (CT&AS1).

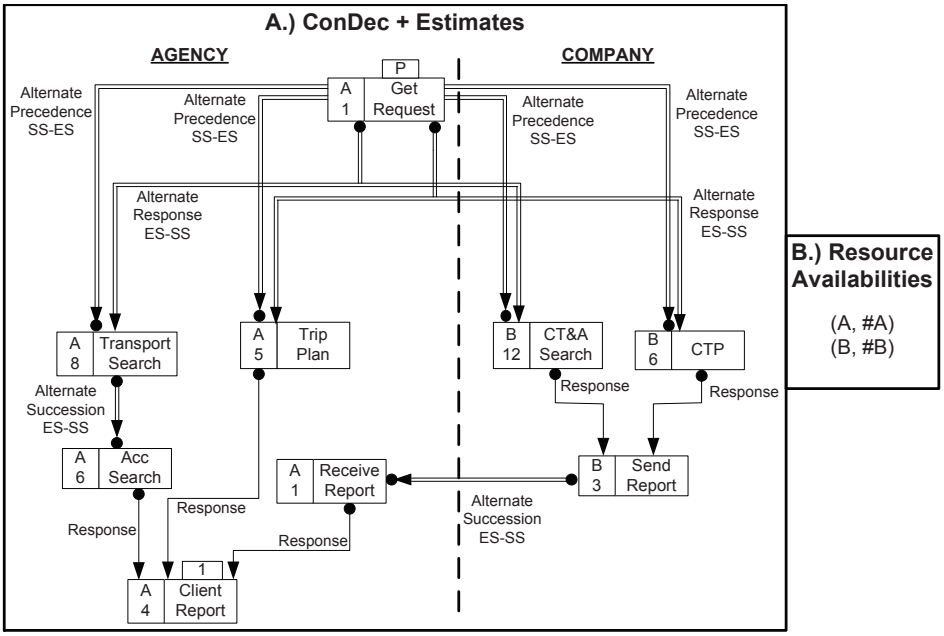


Figure 5.2: ConDec-R Specification for the Travel Agency Problem.

Once both activities TP1 and CT&AS1 **start**, the second client request can be received (G2). The fact that an activity B can only start after another activity A has **started** is stated by considering the predecessors of A as predecessors of B. In this case, the predecessor of TP1 and CT&AS1 (i.e., G1) must (directly o indirectly) precede G2. Regarding the second Get Request activity (G2), the trip plan is organized by the company (CTP2 activity), while the transport and accommodation search are carried out by the agency (TS2 and AS2 activities). In this case, activity AS2, which is related to the second request, is postponed until after the end of the execution of other activities related to the third request (G3, TP3), for efficiency reasons (notice that there is no constraint between the repeated activities TP and AS). Once both activities CTP2 and TS2 start, the third client request can be received (G3). Regarding the third Get request (G3), the trip plan is created by the agency (TP3 activity), while the transport and accommodation search are carried out by the company (CT&AS3 activity). Once both activities TP3 and CT&AS3 start, the fourth Get request can be received (G4). Regarding the fourth request (G4), the trip plan is created by the company (CTP4 activity), while the transport and accommodation search are carried out by the agency (TS4 and AS4 activities). After all the client requests which are carried out by the company are finished, the Send Report (SR) activity can be executed. After this, the Receive Report (RR) activity can be executed. Finally, after all activity executions, the Client Reports (CR) activity is executed.

Table 5.2: Templates of the travel agency problem.

Relation	Description
$Exactly(\#P,G)$	G must be executed each time a client request is received.
$Exactly(1,CReports)$	$CReports$ must be executed exactly once.
$AltPrecSS-ES(G,TS)$	Before the first execution of TS, G must be executed, and between two executions of TS (TS_{i-1} and TS_i), G must also be executed. Formally: $st(G_j) \geq st(TS_{i-1})$, i.e., the next client request can be received at the same moment the previous request **starts** to be processed. $st(TS_i) \geq et(G_j)$, i.e., a transport search cannot start until the client request is completely received (**finishes**).
$AltPrecSS-ES(G,CT\&AS)$	Exactly the same than $AltPrecSS-ES(G,TS)$ by replacing TS by $CT\&AS$.
$AltPrecSS-ES(G,TP)$	Exactly the same than $AltPrecSS-ES(G,TS)$ by replacing TS by TP.
$AltPrecSS-ES(G,CTP)$	Exactly the same than $AltPrecSS-ES(G,TS)$ by replacing TS by CTP.
$AltRespES-SS(G,\{TS,CT\&AS\})$	After the last execution of G, at least one of TS or $CT\&AS$ must be executed, and between two executions of G, at least one of TS or $CT\&AS$ must also be executed.
$AltRespES-SS(G,\{TP,CTP\})$	Exactly the same than $AltRespES-SS(G,\{TS,CT\&AS\})$ by replacing $\{TS,CT\&AS\}$ by $\{TP,CTP\}$.
$AltSucES-SS(TS,AS)$	Before the first execution of AS, TS must be executed, and between two executions of TS (TS_{i-1} and TS_i), AS must also be executed. Furthermore, after the last execution of TS, AS must be executed, and between two executions of AS (AS_{i-1} and AS_i), TS must be executed. Formally: $st(TS_j) \geq st(AS_{i-1})$, i.e., the next transport search can be received at the same moment the previous one **starts** to be processed. $st(AS_i) \geq et(TS_j)$, i.e., the accommodation search cannot start until the transport search is **completed**.
$Resp(CT\&AS,SReport)$	After the last execution of $CT\&AS$, $SReport$ must be executed.
$Resp(CTP,SReport)$	After the last execution of CTP, $SReport$ must be executed.
$AltSucES-SS(SReport,RReport)$	Exactly the same than $AltSucES-SS(TS,AS)$ by replacing TS by $SReport$, and AS by $RReport$.
$Resp(RReport,CReports)$	After the last execution of $RReport$, $CReports$ must be executed.
$Resp(AS,CReports)$	After the last execution of AS, $CReports$ must be executed.
$Resp(TS,CReports)$	After the last execution of TS, $CReports$ must be executed.

In a similar way, for Problem 2, Fig. 5.4 shows the optimized Gantt chart (overall completion time = 33) and the BP model which are obtained with our proposal. It can be seen that regarding the first reception of a client request (depicted by G1 in the Gantt diagram), the trip plan, and the transport and accommodation search are carried out by the company (CTP1 and CT&AS1 activities). Once both activities CTP1 and CT&AS1 start, the second client request can be received (G2). Regarding the second request (G2), the trip plan is created by the company (CTP2 activity), while the transport and accommodation search are carried out by the agency (TS2 and AS2 activities). Once both activities CTP2 and TS2 start, the third client request can be received (G3). Regarding the third Get Request (G3), the trip plan, and the transport and accommodation search are carried out by the company (CTP3 and CT&AS3 activities). Once both activities CTP3 and CT&AS3 start, the fourth Get Request can be received (G4). Regarding the fourth

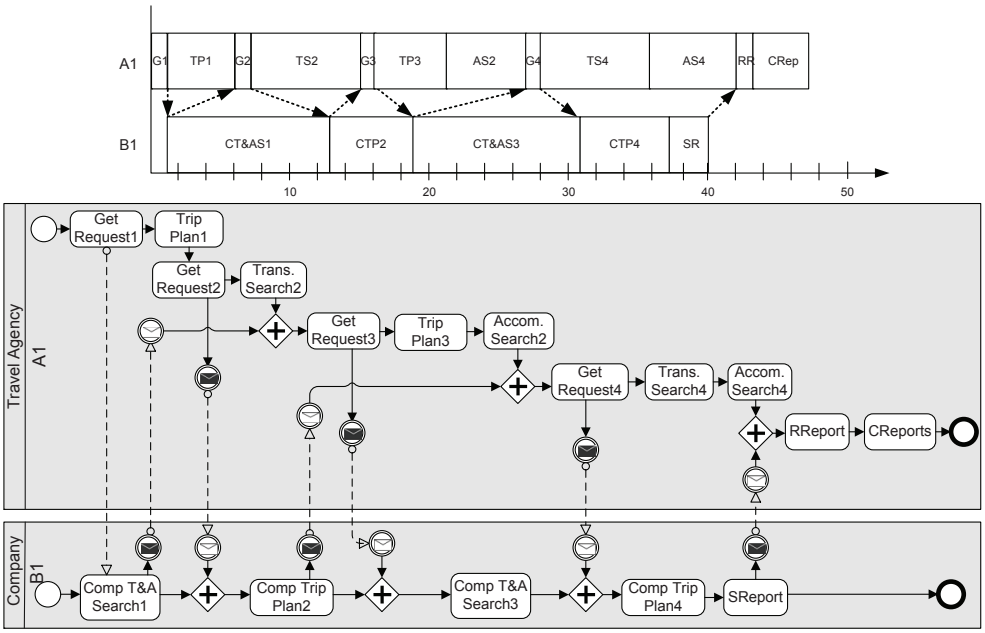


Figure 5.3: Optimized Gantt chart and BPMN for the Travel Agency Problem for #P = 4, #A = 1 and #B = 1.

request (G4), the trip plan is created by the company (CTP4 activity), while the transport and accommodation search are carried out by the agency (TS4 and AS4 activities). After all the client requests which are carried out by the company are finished, the Send Report (SR) activity can be executed. After this, the Receive Report (RR) activity can be executed. Finally, after all the activity executions, the Client Reports (CR) activity is executed.

5.3.4 Dynamic Programming for Combining Solutions of the Travel Agency Problem

A feasible solution to a model of a number of instances can be easily obtained by concatenating known solutions for the same model with a smaller number of instances. The optimal way to perform this concatenation can be achieved by **Dynamic Programming** (DP) (Bellman, 1957). For the current chapter, DP is used to swiftly obtain a feasible solution, usually of good quality, which helps to the CP search. This initial solution is built as result of the optimal concatenation of the available (optimal or good) solutions to smaller problems. In general, the way in which solutions to problems of a given size can be combined to provide a solution to a larger problem depends on the type of problem considered.

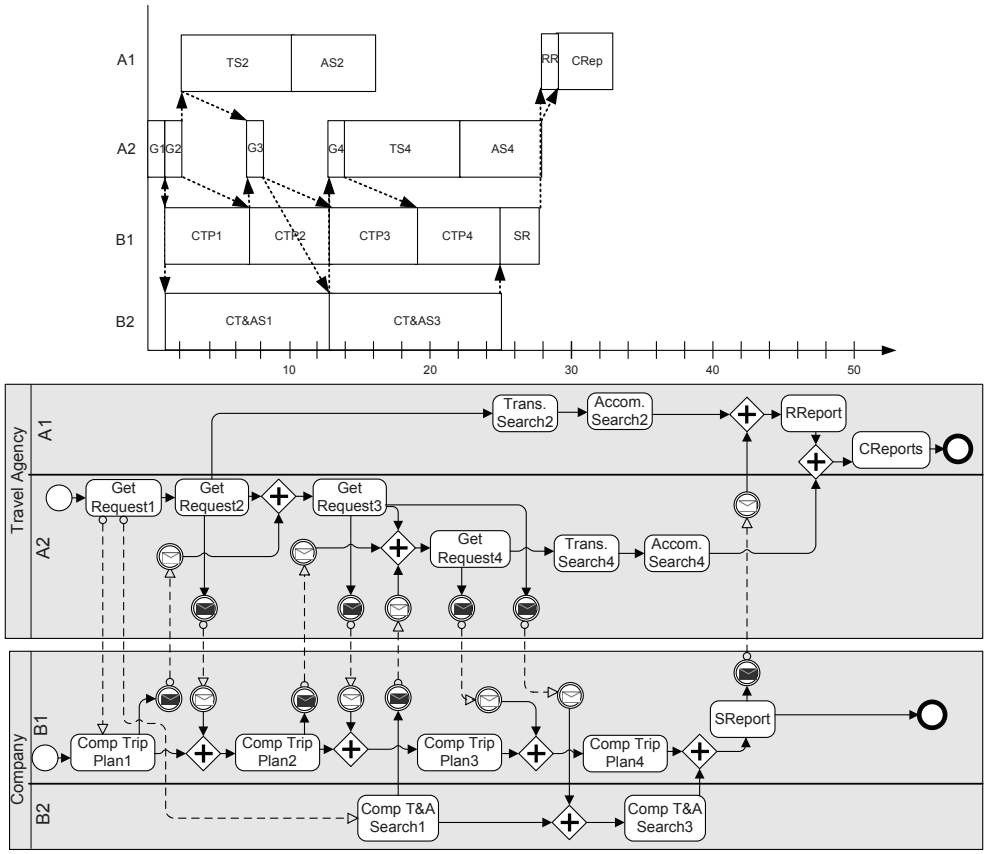


Figure 5.4: Optimized Gantt chart and BPMN for the Travel Agency Problem for #P = 4, #A = 2 and #B = 2.

For the travel agency problem, DP can be applied for obtaining good solutions by joining optimal solutions to smaller problems. Let $OCT_{a,b}(p)$ be the best overall completion time which is known for an instance `(p, a, b)` of the travel agency problem. DP can be applied to the travel agency problem so that the best overall completion time for `(p, a, b)` obtained through DP, $OCT\_DP_{a,b}(p)$, can be defined by:

$$OCT\_DP_{a,b}(p) = \min_{1 \leq i \leq p/2}(OCT_{a,b}(i) + OCT_{a,b}(p-i) - dur(CReports))$$[3]

i.e., the best combination of two optimal/optimized solutions to smaller problems is chosen.

[3] `CReports` activity must be executed only once, and must be allocated after the execution of all other activities.

Table 5.3: Response variables

Id	Description
CP/DP($s1$+$s2$)	The way in which the best solution is found, which can be by means of the proposed constraint-based approach, CP, or by dynamic programming through combining $s1$ and $s2$, DP($s1$+$s2$)
OCT	Overall completion time for the generated optimized enactment plan
%BusyA	Average percentage of use of resources of role A, regarding the overall completion time
%BusyB	Average percentage of use of resources of role B, regarding the overall completion time

5.4 Empirical Evaluation

In order to evaluate the effectiveness of the proposal, a controlled experiment is conducted. Section 5.4.1 describes the design underlying the experiment, and Section 5.4.2 shows the experimental results and the data analysis.

5.4.1 Experimental Design

Purpose: The purpose of the empirical evaluation is to analyze the proposed approach in the generation of optimal enactment plans from ConDec-R specifications, specifically, the goals are: (1) the comparison of the proposed constraint-based proposal with Dynamic Programming (DP) (cf. Sect. 5.4.2), and (2) the demonstration of its use for simulation purposes (cf. Sect. 5.4.2).

Objects: The travel agency problem is used as example for the current evaluation, since it includes various and representative relations of several types and complexity from the set of all the ConDec-R templates[4].

Independent Variables: For the empirical evaluation, the number of client requests, #P, the number of resources of role A, #A, and the number of resources of role B, #B, are taken as independent variables.

Response Variables: Some performance measures (cf. Table 5.3) related to the best generated plan are reported for the generated problems (Figs. 5.5; Tables 5.4, 5.5, and 5.6).

Experimental Design: Based on the travel agency problem, a wide set of problem instances by varying the different independent variables are generated: $\#P$, $\#A$ and $\#B$. For variable $\#P$, the values 1..100 are considered, for $\#A$, the values 1..5 are considered, and for $\#B$, the values 1..5 are considered.

[4]A tool for generating optimized BP models for the travel agency problem can be found at http://regula.lsi.us.es/AgenciesOptimizedModels/, where some tests can be carried out.

Table 5.4: %OCT of the best solution found and method (CP or DP) that reaches it

P	(#A, #B)	OCT	CP/DP($s1$+$s2$)	P	(#A, #B)	OCT	CP/DP($s1$+$s2$)
1	(1, 1)	20	CP	11	(1, 1)	119	DP(5+6)
1	(2, 2)	19	CP	11	(2, 2)	72	CP
1	(3, 3)	19	CP	11	(3, 3)	68	CP
1	(4, 4)	19	CP	11	(4, 4)	68	CP
1	(5, 5)	19	CP	11	(5, 5)	67	CP
2	(1, 1)	27	CP	12	(1, 1)	128	DP(6+6)
2	(2, 2)	21	CP	12	(2, 2)	80	DP(5+7)
2	(3, 3)	21	CP	12	(3, 3)	71	CP
2	(4, 4)	21	CP	12	(4, 4)	70	CP
2	(5, 5)	21	CP	12	(5, 5)	70	CP
3	(1, 1)	38	CP	13	(1, 1)	139	DP(6+7)
3	(2, 2)	28	CP	13	(2, 2)	85	CP
3	(3, 3)	28	CP	13	(3, 3)	78	CP
3	(4, 4)	28	CP	13	(4, 4)	75	CP
3	(5, 5)	28	CP	13	(5, 5)	75	CP
4	(1, 1)	47	CP	14	(1, 1)	149	DP(6+8)
4	(2, 2)	33	CP	14	(2, 2)	92	DP(7+7)
4	(3, 3)	33	CP	14	(3, 3)	84	CP
4	(4, 4)	33	CP	14	(4, 4)	84	CP
4	(5, 5)	33	CP	14	(5, 5)	84	CP
5	(1, 1)	57	CP	15	(1, 1)	160	DP(7+8)
5	(2, 2)	36	CP	15	(2, 2)	98	DP(7+8)
5	(3, 3)	35	CP	15	(3, 3)	91	DP(5+10)
5	(4, 4)	35	CP	15	(4, 4)	88	CP
5	(5, 5)	35	CP	15	(5, 5)	88	CP
6	(1, 1)	66	CP	16	(1, 1)	170	DP(8+8)
6	(2, 2)	44	CP	16	(2, 2)	103	CP
6	(3, 3)	43	CP	16	(3, 3)	97	CP
6	(4, 4)	43	CP	16	(4, 4)	93	CP
6	(5, 5)	43	CP	16	(5, 5)	93	CP
7	(1, 1)	77	CP	17	(1, 1)	181	DP(8+9)
7	(2, 2)	48	CP	17	(2, 2)	110	DP(8+9)
7	(3, 3)	45	CP	17	(3, 3)	101	DP(7+10)
7	(4, 4)	45	CP	17	(4, 4)	99	CP
7	(5, 5)	45	CP	17	(5, 5)	99	CP
8	(1, 1)	87	CP	18	(1, 1)	190	DP(6+12)
8	(2, 2)	54	CP	18	(2, 2)	116	DP(9+9)
8	(3, 3)	52	CP	18	(3, 3)	104	CP
8	(4, 4)	52	CP	18	(4, 4)	104	CP
8	(5, 5)	52	CP	18	(5, 5)	104	CP
9	(1, 1)	98	CP	19	(1, 1)	201	DP(7+12)
9	(2, 2)	60	CP	19	(2, 2)	122	DP(8+11)
9	(3, 3)	57	CP	19	(3, 3)	112	DP(7+12)
9	(4, 4)	57	CP	19	(4, 4)	111	DP(7+12)
9	(5, 5)	57	CP	19	(5, 5)	111	DP(7+12)
10	(1, 1)	109	DP(4+6)	20	(1, 1)	211	DP(8+12)
10	(2, 2)	67	CP	20	(2, 2)	128	DP(9+11)
10	(3, 3)	60	CP	20	(3, 3)	116	DP(10+10)
10	(4, 4)	60	CP	20	(4, 4)	116	DP(10+10)
10	(5, 5)	60	CP	20	(5, 5)	116	DP(10+10)

Experimental Execution: By taking into account the NP-complexity of the considered problems, an **incomplete search** is adapted and applied for solving the specific considered problems and also its suitability for the generation of BPMN

models from constraint-based specifications is evaluated. This incomplete search includes **randomized components** in order to diversify the search. By means of this approach, a first feasible solution is quickly found by a randomized greedy algorithm. The same greedy algorithm is used for iteratively improving the best solution found until a time limit is reached. Through this incomplete search, all the solutions can be reached and the search procedure efficiently explores a wide range of solutions from diversified areas of the search space.

For the experiments, the constraint-based search algorithm is run until a 10-minute CPU time limit is reached. The machine for all experiments is an Intel Core2, 2.13 GHz, 1.97 GB memory, running on Windows XP. In order to solve the constraint-based problems (cf. Sect. 3.3, the developed algorithms have been integrated with the system COMET (Dynadec, 2011), which is able to generate high-quality solutions for highly constrained problems in an efficient way.

5.4.2 Experimental Results and Data Analysis

As commented, the purpose of the empirical evaluation is twofold, i.e., analyzing the suitability of the proposed approach through a comparison with DP, and through the use for simulation.

Comparison with DP

Dynamic programming (Bellman, 1957), DP, is a widely used technique in solving optimization problems, leading to solutions of high quality in most cases. Specifically, for the travel agency problem, DP can be applied (cf. Sect. 5.3.4). We would like to evaluate whether the constraint-based proposal usually improves the solution of good quality which can be obtained by DP, i.e., works efficiently. In this way, DP is used for generating a first feasible solution of good quality. As discussed, CP tries to improve known solutions by taking advantages of the information about their objective value.

Table 5.4 shows the overall completion time for the best solutions which are found for some representative instances, together with the method (either dynamic programming (DP) or constraint programming (CP)) which finds the best solution (column CP/DP($s1$+$s2$) in Table 5.4). Thereby DP($s1$+$s2$) means that the best solution is found by dynamic programming through combining $s1$ and $s2$. It can be observed that for $1 \leq \#P \leq 14$, the constraint-based approach obtains better solutions than DP for almost all of the instances. Moreover, for $15 \leq \#P \leq 18$, in some cases DP obtains the best solution, and in other cases CP obtains the best solution. Moreover, for $19 \leq \#P \leq 20$, DP obtains solutions that are better than or equal to those obtained through CP for all the instances. Furthermore, it seems that the solutions for $\#P = \{6, 7, 8, 9, 10 \text{ and } 12\}$ are largely optimized since

they widely appear in the DP solutions. The results (cf. Table 5.4) show that the proposed constraint-based approach, i.e., CP, is able to improve the solution obtained by DP in most of the cases. This shows that the proposed constraint-based approach works efficiently in the generation of optimized enactment plans, and hence, for the automatic generation of optimized BPMN models.

In short, the presented data indicates that for the generation of optimized BP models, CP enables complex problems to be solved in a more efficient way than they would be through other alternative methods, such as DP or the manual specification of BP models.

Additionally, when #P increases, the complexity of the problem rises sharply, and hence the manual treatment of the problem would become almost inextricable. In contrast, when using the presented approach an optimized solution for large problems, such as for $\#P = 100$ can be obtained in only 10 minutes.

Threats to validity: There are several factors which may threaten the validity of the presented experiments for the attainment of generalizable conclusions:

- The specific characteristics of the running example, i.e., the empirical evaluation only considers a concrete problem with a specific number of BP activities and specific relations between the BP activities.
- The way in which the optimal/optimized solutions for problems of a certain size are combined in order to obtain solutions for larger problems is specific for the considered running example. In most cases, feasible solutions for larger problems can be generated by concatenating solutions to smaller problems through dynamic programming. However, the way in which solutions to problems of a given size can be combined to provide a solution to a larger problem depends on the type of problem considered.

Use for Simulation

The proposed approach can be used for simulation purposes in the BP Design & Analysis phase in order to study the relevance of several parameters in the quality of the generated plans, e.g., resource availability. As an example, the relevance of the number of available resources for the travel agency problem is analyzed as follows.

Figure 5.5 shows the completion time of the best BP enactment plan (overall completion time) which is generated through the proposed approach. In both graphics of Fig. 5.5, the overall completion time is shown depending on the number of resources of roles A and B. First, the considered resources are grouped according to #A (Fig. 5.5.(a)). Secondly, the considered resources are grouped according to #B (Fig. 5.5.(b)). It can be seen, in most cases, that the overall completion time greatly decreases as #A increases. Additionally, in most cases, the

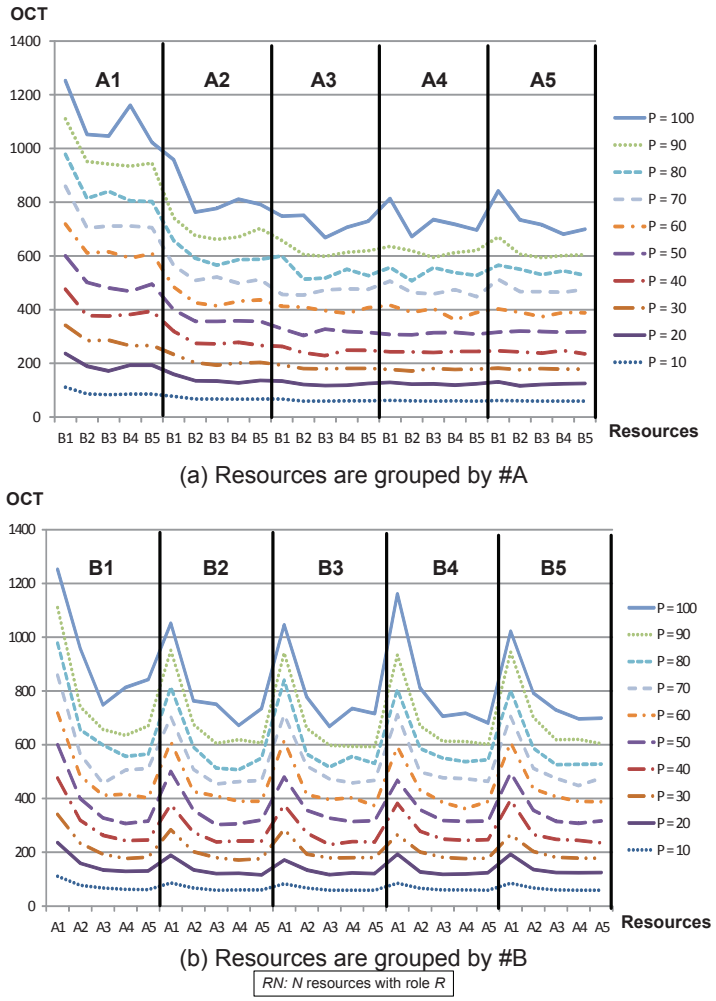


Figure 5.5: Overall completion time depending on #A #B.

overall completion time remains almost the same when #B increases. Therefore, #A seems to be much more influential than #B for the overall completion time, i.e., A is a more critical resource for the considered travel agency problem.

In Tables 5.5 and 5.6, the average percentages of use of the resources of role A and role B, respectively, regarding the overall completion time, are shown. In all cases, for the same value of #P, these percentages decrease as the number of resources of the associated role increases. Moreover, as expected, the average percentage of use of resources of role A is greater than the average percentage of use of resources of role B for the same value of #P.

Table 5.5: %Busy A versus #P

	#P									
#A	10	20	30	40	50	60	70	80	90	100
1	95.2	95.1	95.0	95.0	95.0	95.0	95.0	95.0	95.0	95.0
2	79.8	82.6	81.4	83.1	82.2	81.7	82.6	83.3	82.8	82.4
3	73.5	74.8	74.3	75.5	75.1	74.8	75.4	75.9	75.6	75.4
4	57.8	59.0	59.1	59.6	59.6	59.5	59.7	59.9	59.8	59.8
5	43.9	46.0	45.9	46.5	46.3	46.2	46.5	46.7	46.6	46.5

Table 5.6: %Busy B versus #P

	#P									
#B	10	20	30	40	50	60	70	80	90	100
1	78.6	80.8	80.3	82.0	81.4	81.0	81.9	82.6	82.2	81.9
2	72.7	71.1	75.3	72.2	74.6	76.2	74.3	72.8	74.1	75.1
3	46.7	48.1	49.1	48.9	49.3	49.6	49.4	49.3	49.5	49.7
4	30.5	30.7	31.3	31.2	31.4	31.6	31.5	31.4	31.5	31.6
5	26.4	27.2	27.5	27.7	27.7	27.8	27.8	27.9	27.9	27.9

5.5 Discussion and Limitations

In BP most environments, the Process Design & Analysis phase is manually carried out by business analysts, who must deal with several aspects, such as resource allocation, the activity properties and the relations between them, and may even have to handle the optimization of several objectives. Therefore, in some cases, the manual specification of BP models can consume great quantity of resources, cause failures, and lead to non-optimized models, resulting in a very complex problem (Ferreira and Ferreira, 2006).

Hence, it should be emphasized that the automatic generation of BP models facilitates the human work in most cases, prevents failures in the developed BP models, and enables better optimization to be attained in the enactment phase.

Furthermore, the specification of process properties in a declarative way allows the user to specify what is to be done, and the proposed AI-based tool is in charge of determining how it is to be done in order to satisfy the problem specifications, and to attain the optimization of certain objective functions.

Additionally, notice that in the BP design phase, BP models are traditionally specified in a static way through an imperative language, which is difficult to adapt to changing procedures or to different business goals (Ferreira and Ferreira, 2006). Conversely, the dynamic generation of imperative BP models from static

declarative specifications (specifically constraint-based specifications) just before starting the BP enactment is proposed, once some values for the enactment parameters, e.g., resource availabilities, are known. In this way, the BP models can dynamically be adapted to different environments.

Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way, as demonstrated in Sect. 5.4. Therefore, a wide study of several aspects can be carried out by simulation, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by generating several kinds of problems.

It should also be clarified that the BP models are generated for execution purposes, and hence clarity of meaning for the users of the generated models is not considered relevant in the current proposal.

However, the proposed approach also presents a few limitations. First, the business analysts must deal with a new language for the constraint-based specification of BPs, therefore a period of training is required in order to let the business analysts become familiar with ConDec-R specifications. Secondly, the optimized BP models are generated by considering estimated values for the activity duration and resource availability, hence then current proposal is only appropriate for processes in which the duration of the activities and the resource availability can be estimated. However, P&S techniques can be applied to replan the activities in the enactment phase by considering the actual values of the parameters. Moreover, ConDec-R specifications deal with both control-flow and resource perspectives, and also temporal data. Incorporating the non-temporal data perspective is subject to future work. Notice that already without non-temporal data many problems can be solved.

There are several objectives which can be considered in BPMSs. In this chapter, minimizing the overall completion time only is considered. However, this proposal can be extended in order to consider further objectives, such as cost or other temporal measures.

5.6 Related Work

Related to the presented proposal is research on the generation of BP models, e.g., (Alves et al., 2008; González-Ferrer et al., 2009; R-Moreno et al., 2007; Hoffmann et al., 2010; De Castro and Marcos, 2009; Ferreira and Ferreira, 2006). While the proposals of (Alves et al., 2008; González-Ferrer et al., 2009) provide the BP information through an execution/interchange language, XPDL, the proposed approach, in turn, uses a declarative approach based on a formal logic (LTL). In a related way, in (R-Moreno et al., 2007), planning tools are used for the semi-automatic generation of BP models, by considering the knowledge introduced

through BP Reengineering languages. In (R-Moreno et al., 2007), they propose an object-oriented structure modelling tool that follows their own rule-based approach, while the use of an extension of ConDec, a widely referenced language in the context of BPM (e.g., (Ly et al., 2008; Montali, 2009)), which also allows a higher level of abstraction is proposed in the current chapter. Additionally, (Hoffmann et al., 2010) proposes a planning formalism for the modelling of BPs through an SAP specification (Status and Action Management, SAM), which is a variant of PDDL. Unlike the proposed approach, neither the resource perspective nor the optimization of several instances are considered since in (Hoffmann et al., 2010) each non-deterministic action (i.e., activity) cannot be repeated in the generated solution. Moreover, (De Castro and Marcos, 2009) presents a service-oriented approach which transforms high-level BP models into web service composition models. This approach uses UML to specify the BP models from an MDA point of view, which lacks an implementation view of BP models (Owen and Raj, 2003), in contrast to ConDec, which is a graphical and specific language for the modelling of BPs. Furthermore, (Ferreira and Ferreira, 2006) proposes to refine BP models by combining learning and planning techniques, starting from processes which are not fully described. Unlike the proposed approach, (Ferreira and Ferreira, 2006) needs past process executions and examples provided by the user to apply learning techniques. Moreover, (Ferreira and Ferreira, 2006) does not consider the optimization of any objective function in the generation of the plans. Furthermore, in (Ferreira and Ferreira, 2006) executable plans are generated, while the generation of BP models is proposed in the current approach which are specified in a standard language, i.e., BPMN, which can also be improved by business analysts if necessary.

On the other hand, related to the combined used of declarative and imperative models, (Rychkova et al., 2008b) proposes the use of declarative BP models for specifying processes independently of a particular environment in order to align optional process customizations. This information is complemented with imperative BP specifications which contain information related to the control flow of the processes, often specific to a given environment. In (Rychkova et al., 2008b) both declarative and imperative BP models need to be (manually) specified, while in the proposed approach the optimized imperative models are automatically generated. Lastly, (Caron and Vanthienen, 2011) analyzes the need of transitions between different BP modelling paradigms, i.e., declarative, imperative and hybrid proposals, supporting the presented proposal.

Chapter 6

Planning and Scheduling of Business Processes in Run-Time

6.1 Introduction

6.1.1 Motivation

The execution of most BPs entails, in some way, scheduling decisions since the activities to be executed may compete for some shared resources. In these cases, it is necessary to allocate the resources in a suitable way, usually optimizing some objectives. Scheduling decisions are typically made by the BPM systems (BPMSs) during process execution by automatically assigning activities to resources (Russell et al., 2005).

To lesser measure, planning problems are present in BP executions when, in some points, several possible execution branches exist, and the selection of the suitable one depends on the BP goal and/or on the optimization of some criterions. Since BP models are typically specified in an imperative way, most of the planning decisions are taken in the modelling phase. Specifically, the ordering and the selection of the activities to be executed (planning) in the BP enactment are specified in the BP design time, when only estimated values for several parameters can be analyzed.

There are BPs which entail complex planning decisions which can greatly be influenced by the values of several unpredictable parameters, whose actual value is known in run time.

6.1.2 Contribution

In this chapter, a proposal for modelling and enacting BPs that involve planning and scheduling (P&S) decisions (cf. Sect. 2.2.3) is presented. The main contri-

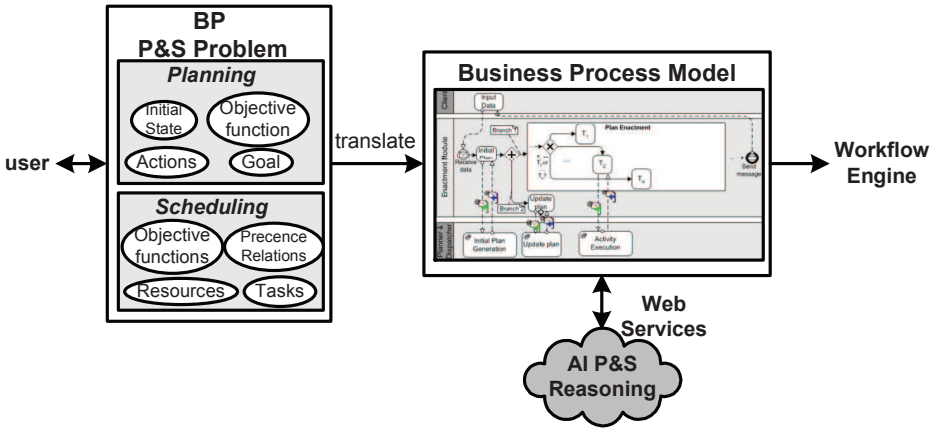


Figure 6.1: Planning & Scheduling of general BPs in run-time.

bution is that both P&S decisions are taken in BP run-time, providing the process management with efficiency and flexibility, and avoiding the drawbacks of taking these decisions during the design phase.

In Fig. 6.1, a graphical representation of the current proposal is shown. The user specifies the information of a P&S problem, that must include several aspects: the initial state of the problem, the goal that must be reached, the activities that can be executed in order to reach the goal (actions), together with the precedence relations between them, the required resources, and the objective functions to be optimized. Moreover, the estimated values of several parameters, such as activity durations or resource availabilities, must be indicated. However, these estimated values can be different from the actual values. The presented approach manages this aspect through replanning techniques in run-time. Furthermore, in the proposed approach, the BP representing a P&S problem needs to be translated to an imperative BP model in BPMN language[1], that still represents a P&S problem since the selection and the ordering of the activities, together with the temporal resources allocation, are carried out in the enactment phase. The resulting BPMN model includes a pool containing web services based on AI P&S techniques that drives the BP execution considering the optimization functions and the actual values of the parameters. In most cases, BPMN models can be translated into an execution language (Ouyang et al., 2006), such as BPEL (BPEL, 2007), which enables BP designs to be deployed into BPM systems and let their instances be executed by a BPM engine in an optimized way.

[1]This translation is not automatic, and hence it needs to be manually carried out. As an example, a method for modelling a specific complex P&S problem through BPMN language is detailed in Sect. 6.3.

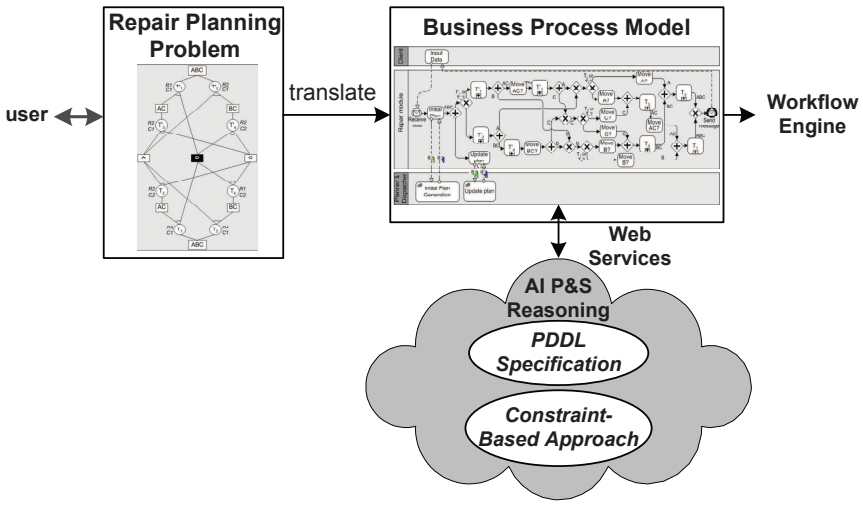


Figure 6.2: Planning & Scheduling of Repair Planning BP in run-time.

The main advantage of the current proposal is the flexible and dynamic management of the BP enactment, since the P&S decisions are taken considering the actual execution values instead of the estimated ones, optimizing the objective functions. It is important to emphasize that the decision-making in run-time provides the problem with a high spatial and temporal complexity, and hence it is necessary to find a good balance between flexibility-optimality, and complexity of the BP.

As an example, a complex and representative problem including P&S, the multi-mode repair planning problem (cf. Sect. 6.3), is managed through the proposed approach (cf. Fig. 6.2). For this problem, the minimization of the total duration and cost when executing the plan in a generic multiple resource environment is considered, taking into account duration and cost of the activities, and resource allocation. In order to solve the repair planning problems, this chapter presents a constraint-based approach (cf. Appendix D.1) and a PDDL 2.2 specification (cf. Appendix D.2) for managing the P&S of the BP activities to achieve an optimal BP enactment.

The main contributions of this chapter can be summarized as follows:

- BP-based architecture for flexible and dynamic management of the BP enactments that, in run-time, require:
  - Planning: selection and ordering of the activities to be executed due to the existence of several possible alternatives, and
  - Scheduling: resources allocation involving temporal reasoning due to the use of shared resources

for optimizing some objective functions (cf. Sect. 6.2).

- AI P&S reasoning based on (1) a constraint-based approach, and on (2) a PDDL 2.2 specification for optimizing the BP enactment, focused on a complex and representative P&S problem: the multi-mode repair planning problem (cf. Sect. 6.3).
- Validation of the proposed approaches through the analysis of different performance measures related to a range of test models of varying complexity (cf. Sect. 6.4).

This chapter is organized as follows: Sect. 6.2 includes an overview of the proposed approach, Sect. 6.3 shows the application of the proposed approach to a complex and representative P&S problem, Sect. 6.4 shows some experimental results, and finally, Sect. 6.5 summarizes related work.

6.2 Framework for the Enactment of BPs Involving P&S Decisions

For BP that entails planning and scheduling decisions, a framework containing three pools is proposed (Fig. 6.3), as presented in the work (Barba and Del Valle, 2010):

1. *Client*: It acts as intermediary between the user and the process. Depending on the concrete problem, the user must specify some relevant information about the process to be executed.
2. *Enactment Module*: It contains all the activities that can be executed. At the end of the enactment, a message containing information about the process execution is sent to the client.
3. *Planner and Dispatcher*: It contains the AI-based web services for the optimal P&S of the BP activities. Depending on the problem to be solved, different AI P&S techniques can be used to obtain an optimal execution plan.

As execution proceeds, the *Enactment Module* and the *Planner and Dispatcher* pool interchange information at several points. Regarding to the proposed architecture, the *Enactment Module* execution starts when a message from the *Client* is received. After that, the first activity to be enacted is the *initial plan*, that requests information about an initial execution plan, trying to obtain a good start point which optimizes the considered objective functions. Next, a parallel gateway divides the execution into two parallel branches:

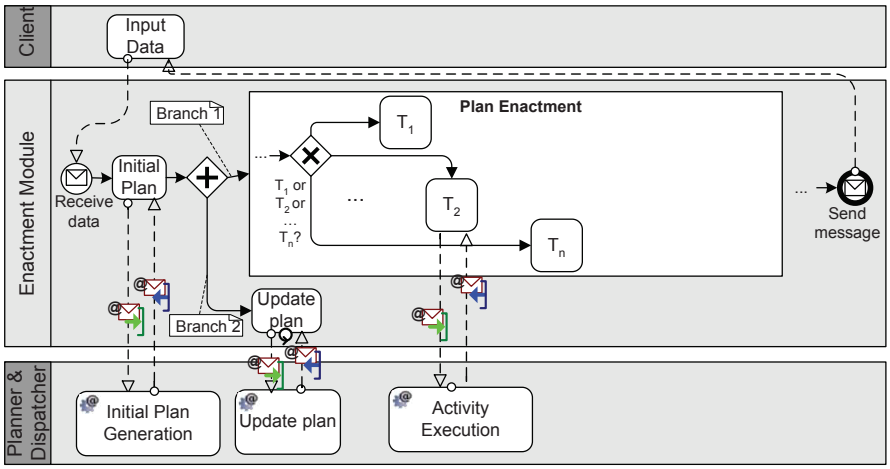


Figure 6.3: Architecture for enacting BPs which involve P&S decisions.

- Plan enactment (Branch 1 in Fig. 6.3): This branch contains the actual P&S activities that can be executed in order to solve the problem. It should be emphasized that all the possible alternative activities to be selected are included in this branch. In this way, P&S methods are used to automatically select and order the suitable activities during the enactment phase, in order to obtain an optimal execution plan. In the *Enactment Module*, when an exclusive data-based gateway is reached during the plan execution, the way that must be followed is established by the information previously received from the *Planner and Dispatcher* pool. In this way, selecting the most suitable activity considering the goal to be reached and the objective function to be optimized is required (planning decision). Moreover, for the activities which require the same resources to be executed, this pool establishes the execution ordering. Therefore, resources need to be allocated to activities by considering the objective function to be optimized (scheduling decision).

- Optimization process (Branch 2 in Fig. 6.3): For the proposed architecture, one of the most important aspects in the plan execution is the optimality. In order to improve it, an optimization process is carried out during all the plan execution, due to, basically, two reasons:

  – The execution plan which was initially generated can be non-optimal, since the generation of optimal plans presents NP-complexity (Garey and Johnson, 1979), and hence it is not possible to ensure the optimality of the generated plans for all the cases in a specific execution time.

– The actual parameters of the enactment can be different from the estimated ones, e.g., different activity duration or resource availability. In these cases, the optimized plan needs to be updated by considering the actual state of the BP execution.

Taking both reasons into account, a loop updating activity that is continuously trying to improve the current solution is included, considering the actual values of the parameters. In order to do this, a web service based on P&S techniques (*Update plan*) is invoked. When a better solution is found, the current plan is updated and this information is used to decide the way to follow in the OR gateways. Lastly, this loop finishes when the complete plan is successfully enacted.

In Sect. 6.3, a specific example is modelled through the proposed generic framework.

6.3 A Case of Study

In the current chapter, a generic framework for modelling and enacting BPs which include P&S decisions is presented (cf. Sect. 6.2). As an example, the multi-mode repair planning problem (cf. Sect. 6.3.1) is managed through this architecture (cf. Sect. 6.3.2).

6.3.1 The Multi-mode Repair Planning Problem

Some of the applications involving P&S issues are maintenance and repair planning, where there may be a cascading set of choices for actions, facilities, tools or personnel, which affect the duration of the plans (Smith et al., 2000). In past years, the effective management of maintenance and repair planning in organizations became more important, since the system complexity is increasing as well as there exist several limitations of the technology which is used to maintain it. Maintenance and repair planning includes a wide scope of problems. Specifically, assembly and disassembly planning are very important in the manufacturing of products and its life cycles. They involve the identification, selection and sequencing of assembly/disassembly operations, which can be specified by their effects on the parts. The identification of assembly/disassembly operations is usually tackled by analyzing the product structure and the feasibility of each possible activity (Homem de Mello and Sanderson, 1991; Calton, 1999), and usually leads to the set of all feasible plans.

In this chapter, the complete repair process of a reparable system (cf. Def. 32) is considered.

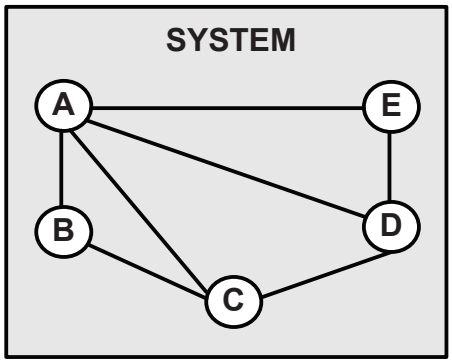


Figure 6.4: Connection graph representing the reparable system ABCDE.

Definition 32. *A **reparable system** is composed by a set of components, and a set of connections between the components.*

A reparable system can be decomposed in certain subsystems (cf. Def. 33), depending on the way that the components (cf. Def. 34) are connected and the type of connections which are included in the connection graph[2].

Definition 33. *A **subsystem** is a subset of a reparable system which is made of more than one components which are connected.*

Definition 34. *A **component** is an atomic part of a reparable system which cannot be disconnected.*

As an example, Fig. 6.4 shows a reparable system made of five components.

The complete system may present an unexpected or anomalous behavior, which is supposed to be due to the fail of one or more components[3]. In these situations, a diagnosis process would be adequate for identifying the faulty component/s.

In order to fix the faulty components, a repair plan must be carried out. This plan is composed of three steps, that can be overlapped in time:

1. Disconnection process: set of activities that are execute to isolate the faulty components.

2. Repair action: activity for fixing the faulty components.

3. Connection process: set of activities that are executed to reconnect the system.

[2]Not all the connections which are included in the connection graph are feasible.

[3]For the sake of simplicity, in the current proposal the unexpected behavior of the reparable system is supposed to be due to the fail of only one component.

Taking these steps into account, a feasible repair plan can be seen as a (minimum) set of activities that begins with the disconnection of the complete system, fixes the faulty components, and finishes with the connection of the complete system. In the current proposal, two kinds of activities are considered, i.e., connection/disconnection activities (cf. Def. 35 and 36) and auxiliary activities (cf. Def. 37 and 38).

Definition 35. *A **connection activity** is an activity which obtains one subsystem by connecting several subsystems or components. It is executed by using an established resource with a particular configuration.*

Definition 36. *A **disconnection activity** is an activity which obtains several subsystems or components by disconnecting one subsystem. It is executed by using an established resource with a particular configuration.*

For the sake of clarity, only two subsystems are considered to be connected in a connection activity and obtained after a disconnection activity.

Moreover, auxiliary activities are required since the connection/disconnection activities are executed at different locations, and the resources can be used with different configurations. Auxiliary activities can be classified into set-up operations (cf. Def. 37) and transportation operations (cf. Def. 38).

Definition 37. *A **set-up operation** is an auxiliary operation which changes the configuration of a resource. It is required when two successive activities with different configuration use that resource.*

Definition 38. *A **transportation operation** is an auxiliary operation which transports a subsystem between locations. It is required when the location where the subsystem is obtained is different from the location where that subsystem is required. In the current chapter, different resources are considered to be placed at different locations.*

All activities which are considered, i.e., connection/disconnection and auxiliary activities, have an associated duration and cost.

In the considered repair planning problem, the connection/disconnection activities (cf. Def. 35 and 36) can be executed in more than one operating mode (multi-mode), each one using different resources or configuration, and possibly different duration and cost. Taking this into account, there can be several options to connect several subsystems to obtain another one, or disconnect one subsystem to obtain several ones.

For the considered repair planning problem, two assumptions are supposed:

- (A1) All activities are reversible, i.e., if a connection activity which connects several subsystems $S_1, S_2, \ldots, S_k$ to obtain another subsystem S exists,

then a disconnection activity which disconnect the subsystem S to obtain the subsystems $S_1, S_2, \ldots, S_k$ is also included, and vice versa.

- (A2) Subsystems that do not include the faulty components are not disconnected.

Taking (A1) and (A2) into account, in the connection process, other subsystems different from the ones generated by the disconnection process can appear, depending on how they are joined. Moreover, disconnection activities only handle subsystems that contain the faulty component, whereas connection activities handle subsystems that may contain or not the faulty component. In general, plans are not linear sequences of activities, unlike reversible plans. Although the disconnection process is linear, the connection process can contain activities that may execute in parallel with others. Furthermore, it is possible that the connection process starts before the disconnection process has finished, and there may be a parallel execution of the two types of activities.

A typical objective to be pursued in repair problems is the minimization of the elapsed time of the plan, i.e., the time in which the reparable system is reconnected after the reparation (called makespan). In the current approach, the total cost of the complete repair plan is also considered for minimization. Therefore, a multi-objective optimization is pursued, encompassing both objective functions, time and cost.

And/Or Graph Representation

In the literature, there are several structures that can be used for representing the considered repair planning problem. The And/Or graph (Homem de Mello and Sanderson, 1990) is one of the most suitable ones, since it allows to represent the set of all feasible connection/disconnection plans in a natural and compact way (cf. Def. 39).

Definition 39. *The **And/Or graph** which represents a repair planning problem is composed by:*

- *Or nodes: they correspond to subsets of components of the reparable system, meaning that the root node represents the whole system (cf. Def. 32), while non-leaf nodes represent subsystems (cf. Def. 33), and leaf nodes are individual components (cf. Def. 34).*

- *And nodes: they represent the activities which connect/disconnect the subsystems (cf. Def. 35 and 36). In this way, a downward edge decomposes a system or subsystem in several subsystems, while an upward edge can be*

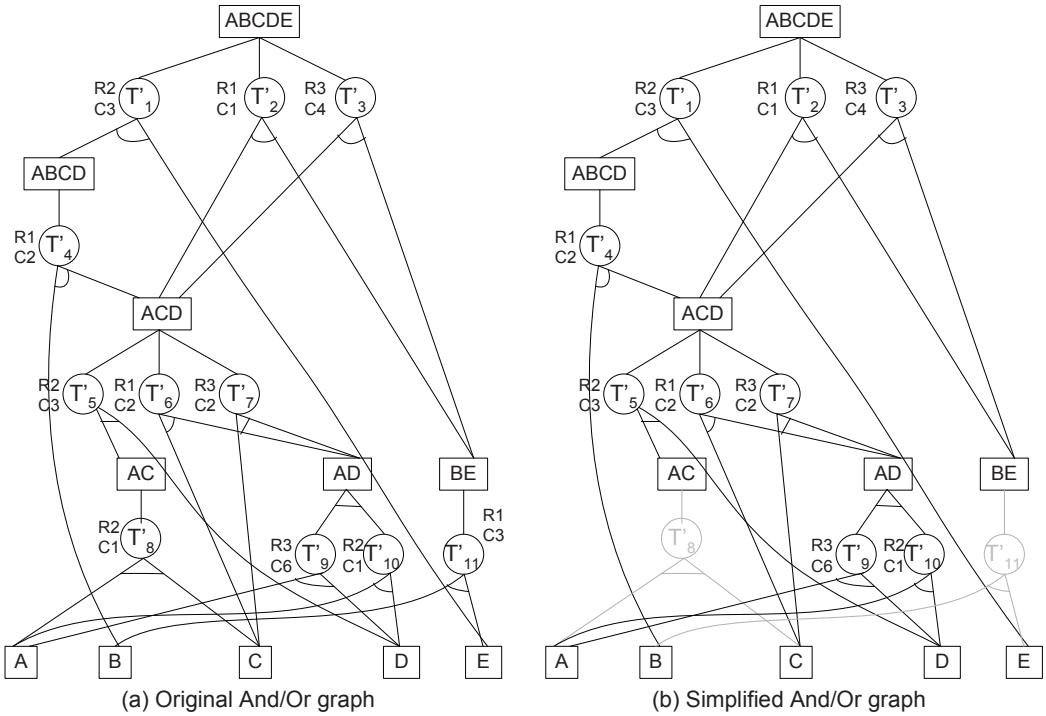


Figure 6.5: The And/Or graph for a reparable system made of five components.

seen as the reverse activity of joining several subsystems into a composed one.

In the current approach, each activity with an operating mode corresponds to a different And node in the graph.

Notice that auxiliary activities (cf. Def. 37 and 38) are not explicitly represented in the And/Or graphs.

In And/Or graphs, each connection/disconnection plan is associated to a tree, that is an And/Or path starting at the root node and ending at the leaf nodes.

Fig. 6.5(a) shows the And/Or graph for a reparable system made of five components, where T' activities represent disconnection activities. For the same Or node, there can be several And nodes (activities) below it, representing different alternatives to connect/disconnect the related subsystem. As example of different modes for the same activity, cf. T'_2 and T'_3 in Fig. 6.5(a). For the problem of Fig. 6.5(a) when D is the faulty component, the subsystems AC and BE are not disconnected since assumption (A2), i.e., subsystems that do not include the faulty components are not disconnected (cf. Sect. 6.3.1), is considered. Therefore, the activities T'_8 and T'_{11} are not selected for the disconnection process in this case (cf. Fig. 6.5(b)). The useless activities, i.e., those And nodes below the Or nodes

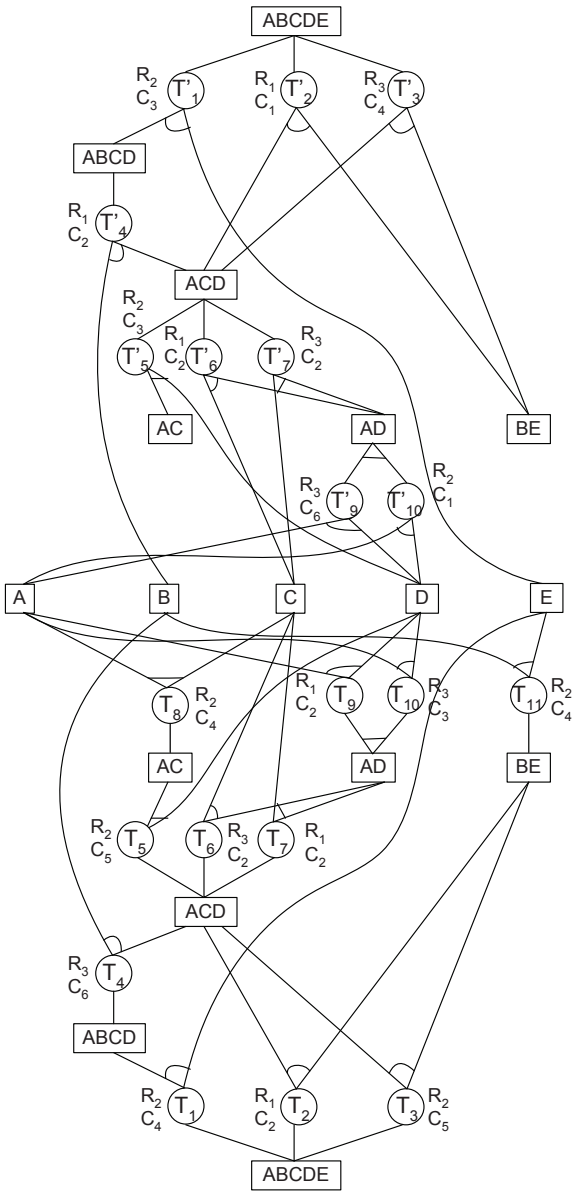


Figure 6.6: The simplified repair And/Or graph for a reparable system made of five components.

corresponding to subsystems which do not contain the faulty component, can be removed from the graph, resulting in a simplified And/Or graph (cf. Def. 40).

Definition 40. *A **simplified And/Or graph** for a specific faulty component is an And/Or graph (cf. Def. 39) in which the And nodes below the Or nodes corre-*

sponding to subsystems which do not contain the faulty component, i.e., useless activities, are not represented.

Moreover, the original And/Or graph (cf. Def. 39) can be opened out by including both connection and disconnection processes, resulting in a repair And/Or graph (cf. Def. 41).

Definition 41. *A **repair And/Or graph** is composed by: (1) a set of subsystems and disconnection activities which represent the disconnection process (top part of the And/Or graph), (2) a set of individual components which represent the isolate components (medium part of the graph), and (3) a set of subsystems and connection activities which represent the connection process (bottom part of the And/Or graph).*

Figure 6.6 presents the simplified repair And/Or graph for the problem of Fig. 6.5 when the faulty component is D, where T' activities represent disconnection activities and T activities represent connection activities.

The original And/Or graph has been extended, so that the new representation includes all the constraints involved in the problem, adding new types of links between And nodes. The new links represent non-precedence constraints: due to the use of shared resources by the tasks and due to the change of configuration in the resources.

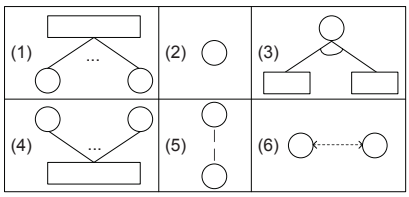


Figure 6.7: Types of Relations

In this way, 6 types of relations are considered (cf. Fig. 6.7), each one representing a link or component of the extended And/Or graph:

Relations of type (1) collect the relation between the information from an Or node and the And nodes below it in the original And/Or graph.

Relations of type (2) consider the durations of connection and disconnection tasks, and correspond to the relationships between the starting and ending times of the connection and disconnection tasks.

Relations of type (3) collect the relation between the information from an And node and the (two) Or nodes below it in the original And/Or graph.

Relations of type (4) consider the relation between the selection of an Or node and all the And nodes above it (possibly only one) in the original And/Or graph.

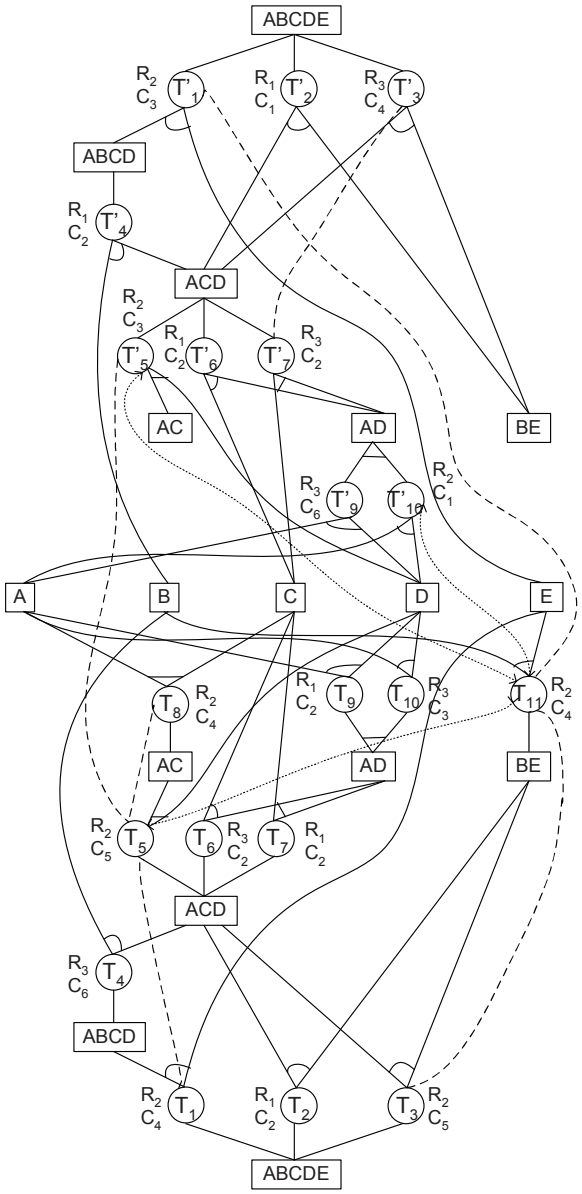


Figure 6.8: The extended simplified repair And/Or graph with relations (5) and (6) between tasks

Relations of type (5) are due to the delay which is needed for managing a change of configuration in a resource between the executions of two successive tasks using the same resource with precedence constraints between them. Those constraints include the relations between reverse disconnection and connection tasks. Notice that for a particular repair plan it is only required relating each task to its closest successor that uses the same resource in the And/Or tree.

Relations of type (6) consider the relation between the tasks which are not related by precedence constraints and which use the same resource.

Types (1), (2), (3) and (4) come from the relations between the nodes included in the original graph, while types (5) and (6) come from the use of (same or different) resources by the different tasks, and they are related to new links between tasks in the extended And/Or graph.

As an example, Fig. 6.8 shows an extension of the And/Or graph which is presented in Fig. 6.6 by including relations of types (5) and (6).

The And/Or graphs represent the system structure through components relations, that remains permanent despite of different required resources or faulty components. Taking this into account, for the same graph there can be several repair problems by varying the faulty components or the required resources.

The And/Or graphs can be used as a basis for representing some problems that involve both P&S, including the repair planning problem, since it allows to show important P&S characteristics, such as alternative activities or precedence relations. Moreover, in general, the And/Or graphs can be easily extended in order to include other aspects. One important advantage of this representation is that the And/Or graph shows the activities that can be executed in parallel.

Multi-objective Optimization

Many problems, such as planning and scheduling problems, can involve multiple conflicting objectives that should be considered at the same time. In multi-objective optimization problems, that have been studied for several decades (Chankong and Haimes, 1983; Miettinen, 1999; Ehrgott, 2005; Coello, 2006), usually no unique solution exists but a set of nondominated solutions can be found. These solutions are also known as Pareto optimal solutions, i.e., for obtaining a better feasible solution in one of the objectives, it is necessary to deteriorate, at least, another one objective. Two typical objectives pursued in planning and scheduling problems, used in this chapter, are the minimization of the total time and cost of the resulting plan. Typically, these two objectives are in conflict, and hence it is not possible to find a plan which is optimum for both objectives. In these cases, optimized plans which present a good balance between the considered objectives are pursued.

In order to solve multi-objective optimization problems, there are, basically, three approaches:

- Defining a new objective function that can be optimized with single objective solvers, such as the weighted-sum method (Zadeh, 1963; Koski, 1985; Kim, 2006), that minimizes $\sum w_i f_i$, where $w_i \geq 0$ for each objective function f_i considered. It is advisable to normalize the objectives with some scaling

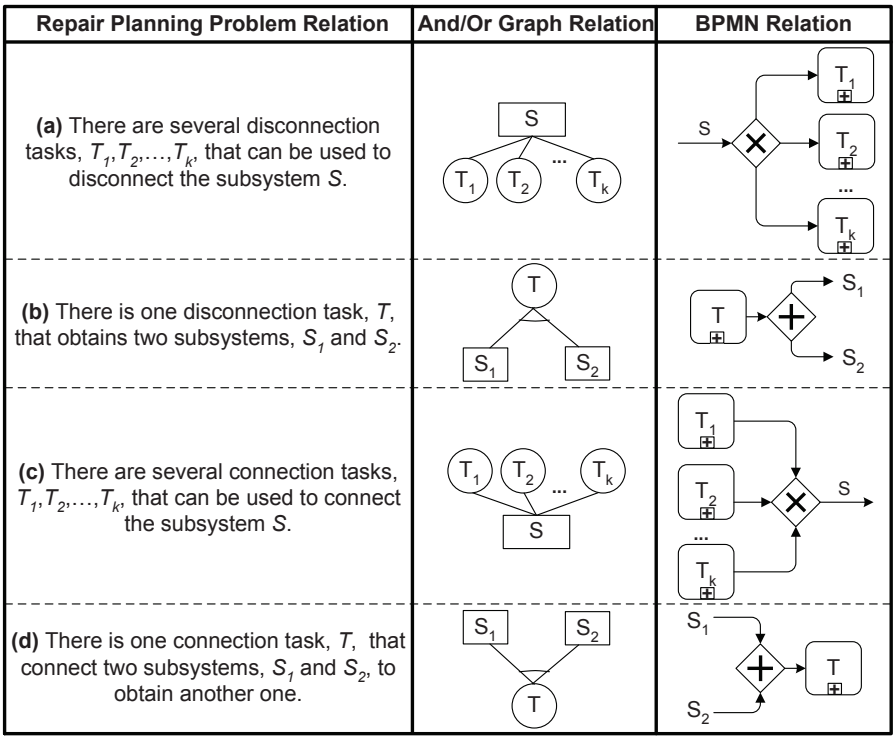


Figure 6.9: Transformation from And/Or Graph relations to BPMN relations

so that different magnitudes do not confuse the method. This method is used in the current approach.

- Optimizing one of the objective functions constraining the other ones (e.g., ε-Constraint Method (Haimes et al., 1971; Chankong and Haimes, 1983; Ehrgott and Ruzika, 2008)).

- Working with a set of Pareto optimal solutions (e.g., Evolutionary Multi-objective Optimization (Goldberg, 1989; Fonseca and Fleming, 1995; Deb, 2008)).

6.3.2 BPMN Model for the Multi-mode Repair Planning Problem

As stated in Sect. 6.3.1, And/Or graphs can be used as a base representation for several AI P&S problems, including the considered repair planning problem. Furthermore, a translation from all the And/Or graph components to a BP modelling language, e.g., BPMN (White and et al., 2004), can be carried out in a direct way, as explained as follows (cf. Fig. 6.9):

(a) The fact that there are several disconnection activities that can be used to disconnect the same subsystem is modelled in BPMN by an exclusive gateway with that subsystem as input and as many outputs as disconnection activities, each one related to each disconnection activity.

(b) The fact that there is one disconnection activity that obtains two subsystems is modelled in BPMN by a parallel gateway with that activity as input and two outputs, each one related to each subsystem.

(c) The fact that there are several connection activities that can be used to connect the same subsystem is modelled in BPMN by an exclusive gateway with as many inputs as connection activities, each one related to each connection activity, and with the resulting subsystem as output.

(d) The fact that there is one connection activity that connects two subsystems is modelled in BPMN by a parallel gateway with two inputs, each one related to each subsystem, and with the connection activity as output.

As stated, the auxiliary activities (cf. Def. 37 and Def. 38) are not explicitly represented in the And/Or graphs. However, they need to be included in the resulting BPMN in order to be executed. Therefore, BPMN activities related to auxiliary activities are added to the BPMN model when necessary, i.e.:

- *set-up operations* are included between two successive activities which use the same resource with different configurations, e.g., T_2' and T_4' in Fig. 6.10. The *Change of configuration* activity, provided that it is necessary, can be executed in parallel with the treatment of the subsystems which are obtained after the activity enactment (cf. Fig. 6.11(b)). The dots in Fig. 6.11(b) represent the treatment of the subsystem/s which result after the execution of the activity. This treatment depends on the specific subsystem. For example, after executing T_2', the subsystems *A* and *BC* are obtained (cf. Fig. 6.11(a)). In this case, the treatment of both subsystems, e.g., Move *BC*?, can be performed in parallel to the change of configuration of resource R_2 (i.e., resource which is used by T_2').

  The *Change of configuration* activity is also controlled by the *Planner & Dispatcher* module since its execution causes the resource lock, that influences the BP enactment. Moreover, the decisions about if this activity is executed or not and when it is executed depend on P&S decisions which are taken by the *Planner & Dispatcher* pool.

  transportation operations are included between two successive activities which manage the same subsystem and use different resources (locations),

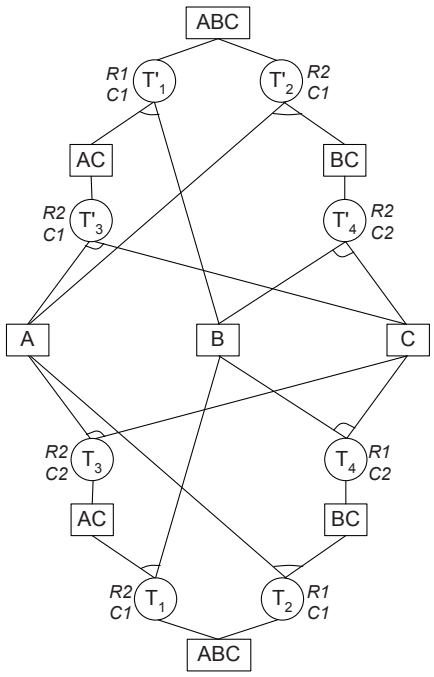
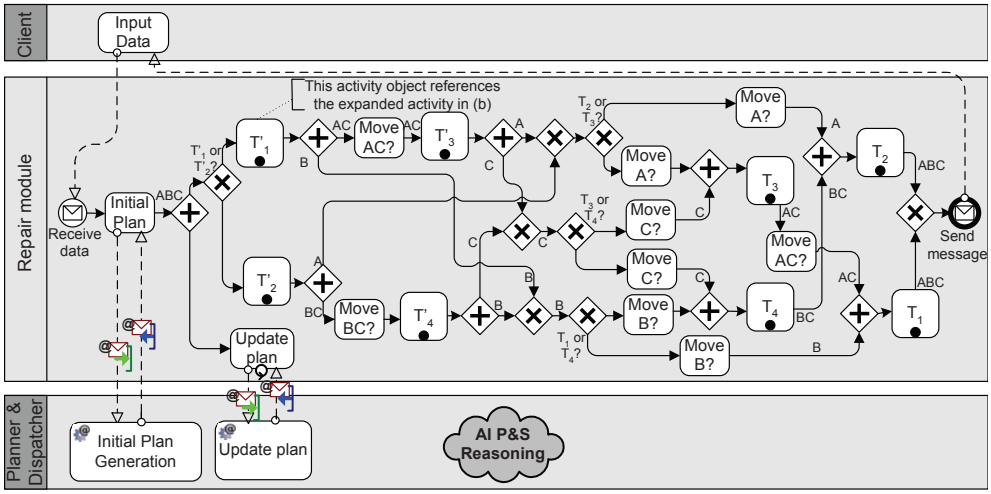


Figure 6.10: Example of the repair And/Or graph for a reparable system made of three components.

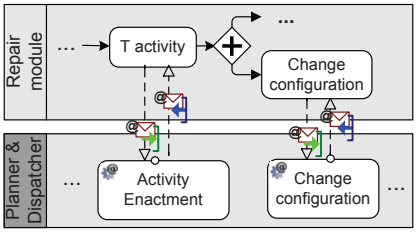
e.g., T_1' and T_3' in Fig. 6.10. Taking into account that the information about the resources where the subsystems are obtained is included in the plan execution, and the information about the resources where they are required depends on the selected branch, it is not necessary to call web services for the *Move subsystem?* execution. If a subsystem is moved or not during the enactment phase depends on the plan execution since different activities can obtain the same subsystem, and this subsystem can be also required for different activities.

Taking the transformations of Fig. 6.9 and the inclusion of auxiliary activities into account, the BPMN which represents the multi-mode repair planning problem of Fig. 6.10 is shown in Fig. 6.11.

In Fig. 6.11(a), the enactment module (cf. Sect. 6.2), i.e., the pool which contains all the activities of the P&S problem which can be executed, is called the repair module. Therefore, the repair module is the one which contains the actual repair plan execution. In this module, the activities are executed considering the P&S established by the *Planner and Dispatcher*, which contains web services based on AI-based methods (specifically on a constraint-based approach (cf. Appendix D.1) and on a planner for solving problems related to a PDDL 2.2 specification (cf. Appendix D.2)). For the sake of clarity, in Fig. 6.11(a) the activ-



(a) BPMN diagram with collapsed activities



(b) BPMN diagram for the expanded activity

Figure 6.11: BPMN diagram of the repair problem of Fig. 6.10.

ities corresponding to connection/disconnection activities appear collapsed, while in Fig. 6.11(b) its expanded representation is shown. In Fig. 6.11(b), *T Activity* represents the actual execution of the connection/disconnection activity. A suitable web service manages it, since the *Planner & dispatcher* must determine the start time of the activity. This is due to the start time of the activity is related to the ordering in which that activity uses a resource, i.e., a scheduling decision which needs to be established by the *Planner & dispatcher* pool. Considering the information about the activity execution, the actual values are stored and consulted for updating the plan and enacting the remaining of the BP.

Notice that this BP contains all the possible alternative activities to be selected, and during the enactment phase, AI P&S methods are used to automatically select and order the suitable activities in order to execute the plan in an optimized way.

Table 6.1: Number of And, Or nodes (average)

	And/Or graph		Simplified And/Or graph			
Prob	#Or	#And	#Or	#And	#And'	#RP
30-1	348	630	223	327	240	1213
30-2	404	828	303	520	365	9200
30-3	415	863	310	546	384	12846
40-1	649	1518	433	833	575	23005
40-2	770	2143	621	1489	984	248408
40-3	756	2060	598	1400	925	197551

6.4 Empirical Evaluation

Section 6.4.1 describes the design underlying the experiments, and Sect. 6.4.2 shows the experimental results and the data analysis.

6.4.1 Experimental Design

Purpose: The purpose of the empirical evaluation is to check the suitability of two approaches for developing the web services which are in charge of P&S the activities of the BPs which represent multi-mode repair planning problems. These approaches are listed as follows:

- A constraint-based approach (called CBP from now), cf. Appendix D.1.
- A PDDL specification of the problem which is used as input of the SGPlan planner (Chen et al., 2006), which compose the generic planning approach (called SGPlan from now), cf. Appendix D.2.

Objects: Table 6.1 shows the number of nodes in the And/Or graphs corresponding to a set of hypothetical products of 30 and 40 components which are used as objects in the current empirical evaluation. Supposing that each individual component must be repaired[4], it includes the average number of Or, And (connection activities), And' (disconnection activities) nodes, and the average number of repair plans (#RP) in the simplified graphs. Each row refers to a set of 80 instances of an And/Or graph for a hypothetical product of 30 or 40 components, with different combinations for the durations of activities, resources and configurations which are used, and faulty component which is selected to be repaired. In order

[4]As stated, for the sake of simplicity only one component is considered to be faulty for each problem.

Table 6.2: Fraction of Optimal solutions which are found by CBP

Prob	f_{Time}	f_{MO20}	f_{MO10}	f_{Cost}
30-1	0,5	0,591	0,898	1
30-2	0	0,011	0,307	0,5
30-3	0	0	0	0
40-1	0	0	0	0
40-2	0,5	0,5	0,5	0,579
40-3	0	0	0	0,045
M30-1	0,5	0,5	0,5	0,954
M30-2	0	0	0,079	0,5
M30-3	0	0	0	0
M40-1	0	0	0	0
M40-2	0,397	0,5	0,5	0,557
M40-3	0	0	0	0,023

to obtain results about the behavior of the model for multi-mode activities, each graph has been extended to include 10% multi-mode activities (M30-1, M30-2, M30-3, M40-1, M40-2 and M40-3).

The cost of each activity is considered to be a function which depends on the resource, the configuration and the duration of this activity. In a related way, the cost of a repair plan is obtained by the sum of the costs of all the activities which are included in the plan.

Experimental Design: Four different objective functions have been selected to be minimized: the cost (f_{Cost}), the makespan (f_{Time}), and two combined objective functions which result of applying the weighted-sum method, i.e., the objective function to minimize is $w_t \times PlanDuration + w_c \times PlanCost$, where w_t and w_c refer to the weight which is given to the total time and the total cost of the repair plan respectively:

- f_{MO10}: In this case, $w_t = 10$ and $w_c = 1$.
- f_{MO20}: In this case, $w_t = 20$ and $w_c = 1$.

The values 10 and 20 have been chosen because the cost of a repair plan which has been considered is, in average, around 15 times the total duration of the plan.

Independent Variables: For the empirical evaluation, the objective function to be minimized, i.e., f_{Time}, f_{MO10}, f_{MO20}, and f_{Cost}, and the problem to be solved, i.e., 30-1, 30-2, 30-3, 40-1, 40-2, 40-3, M30-1, M30-2, M30-3, M40-1, M40-2 and M40-3, are taken as independent variables.

Response Variables: According to the 4 objective functions previously mentioned, some performance measures related to the best generated plan are reported for the considered problems:

Table 6.3: Execution time (average) with CBP

Prob	f_{Time}	f_{MO20}	f_{MO10}	f_{Cost}
30-1	155,2	144,8	152,5	150,3
30-2	300	299,5	300	300
30-3	300	300	300	300
40-1	300	300	300	300
40-2	153,2	151,4	300	300
40-3	300	300	300	300
M30-1	420	368,5	349	121,9
M30-2	600	600	595,2	351,4
M30-3	540,4	600	600	600
M40-1	600	600	600	600
M40-2	416,5	318,25	307	270,4
M40-3	600	593,18	600	587,2

- The fraction of solutions which are found by the constraint-based approach which are proven to be optimal (Table 6.2)[5].
- The average execution time which is spent for finding the solutions by both approaches (Table 6.3 and 6.5).
- The fraction of better solutions (in time and cost) that have been found in both SGPlan and CBP approaches when including 10% multi-mode activities (cf. Tables 6.4 and 6.6) regarding to the same problems but with single-mode activities. This measure indicates if the fact of including multi-mode activities makes improve the solution for the problems possible.
- The fraction of solutions which are found by SGPlan and which are better or equal to the solutions which are found by the CBP approach in time (T column) and cost (C column) (cf. Table 6.7). The equal solutions are also included (less than 3%) due to the SGPlan is the fastest, and hence they can be considered better solutions.

Experimental Execution: The experiments were carried out on a 2,66GHz Intel Core 2 Duo with 4GB RAM.

For testing the constraint-based proposal (CBP) (cf. Appendix D.1), a basic branch-and-bound algorithm is implemented in ILOG Solver (ILOG, 2011). A temporal limit of 300 seconds for single-mode and 600 seconds for 10% multi-mode has been established for the search. In order to guide the search, the order

[5]Unlike the constraint-based approach, SGPlan performs an incomplete search, and hence it is generally not possible to ensure the optimality of a solution.

Table 6.4: Fraction of better solutions for problems including 10% multi-mode activities with CBP

Prob	f_{Time}	f_{MO20}	f_{MO10}	f_{Cost}
30-1	0,01	0	0	0,01
30-2	0,32	0,32	0,32	0,32
30-3	0,19	0,20	0,19	0,19
40-1	0,18	0,18	0,18	0,19
40-2	0,29	0,29	0,21	0,24
40-3	0,52	0,42	0,47	0,44

of selection of the variables to be instantiated is from up to down in the repair And/Or graph. On the other hand, a generic planner, SGPlan (Chen et al., 2006), is used to solve the corresponding problems which are specified in PDDL (cf. Appendix D.2), with several combinations of the functions to be minimized using the weighted-sum method. SGPlan is based on the strong locality of mutex constraints, and proposes to partition the constraints of a planning problem into groups based on their subgoals. SGPlan includes an ordering heuristic and stopping condition based on the marginal improvement of the plan metric.

6.4.2 Experimental Results and Data Analysis

Constraint-based Approach

Table 6.2 shows that the objective functions in which the cost has a high influence, get the best fractions of proven optimal solutions (f_{Cost} gets the best score, followed by f_{MO10}, f_{MO20} and, finally, f_{Time}) and, in most cases, also the results are obtained in the fastest way for problems in which the cost has the highest influence (cf. Table 6.3). This may be due to the differences between the costs of the activities are more significant and discriminating than between the durations, and the cost seems to be less dependent of the ordering of the activities.

Table 6.4 shows the fraction of better solutions (in time and cost) that have been found by the constraint-based approach when including 10% multi-mode activities. A better solution (in time and cost) for multi-mode problem than for single-mode problems has been found in 23,5% of the studied cases. It is possible to see that the more complex is the problem, the more is the advantage of including 10% multi-mode activities, for the problems which are considered.

Table 6.5: Execution time (average) with SGPlan

Prob	f_{Time}	f_{MO20}	f_{MO10}	f_{Cost}
30-1	0,15	0,31	0,31	0,26
30-2	0,43	0,82	0,83	0,75
30-3	0,47	0,86	0,84	0,74
40-1	0,85	1,84	1,90	1,67
40-2	0,65	1,57	1,72	1,51
40-3	1,02	2,21	2,22	2,18
M30-1	0,88	0,31	0,35	0,73
M30-2	4,88	0,93	0,95	4,10
M30-3	4,70	0,87	0,95	3,73
M40-1	11,96	1,84	1,94	11,04
M40-2	9,53	1,70	1,71	9,62
M40-3	11,81	2,40	2,45	14,63

Table 6.6: Fraction of better solutions for problems including 10% multi-mode activities with SGPlan

Prob	f_{Time}	f_{MO20}	f_{MO10}	f_{Cost}
30-1	0,02	0,02	0,02	0,02
30-2	0	0	0	0
30-3	0,04	0,04	0,04	0,04
40-1	0,24	0,24	0,24	0,24
40-2	0,20	0,20	0,20	0,20
40-3	0,25	0,25	0,25	0,25

Planner and PDDL Specification

Table 6.5 shows the execution time (in average) of SGPlan when solving the considered repair planning problems. It is possible to see that for the single-mode problems of 30 components which have been generated, the solution is found in less than 1 second. In a similar way, for the single-mode problems of 40 components which have been generated the solution is found in around 1 or 2 seconds. As expected, this time increases when including multi-mode activities. Comparing Tables 6.3 and 6.5, the generic planning approach seems to be better in the situations when the available time for finding the solutions is low, since this approach is based on incomplete search, i.e., SGPlan includes a stopping condition based on the marginal improvement of the plan metric.

Table 6.6 shows the fraction of better solutions (in time and cost) that have been found by SGPlan when including 10% multi-mode activities. It is possible to see that the fact of including 10% multi-mode activities allows to reach better

solutions in the systems made of 40 components than in the systems made of 30 components which are considered.

Planner and PDDL Specification VS Constraint-based Approach

Table 6.7: Fraction of SGPlan better or equal solutions (compared to CBP)

	f_{Time}		f_{MO20}		f_{MO10}		f_{Cost}	
Prob	T	C	T	C	T	C	T	C
30-1	0,01	0,30	0,03	0,21	0,04	0,19	0,12	0,16
30-2	1	1	1	0,98	1	1	1	1
30-3	0,96	0,99	0,95	0,94	0,96	0,98	0,96	0,99
40-1	0,83	0,85	0,87	0,85	0,83	0,85	0,83	0,85
40-2	0,02	0,20	0,06	0,25	0,05	0,11	0,14	0,10
40-3	0,63	0,77	0,64	0,67	0,63	0,77	0,62	0,71
M30-1	0,17	0,52	0,30	0,55	0,37	0,55	0,16	0,18
M30-2	1	1	1	1	1	1	1	1
M30-3	0,94	0,96	0,94	0,96	0,94	0,96	0,94	0,96
M40-1	0,87	0,92	0,87	0,90	0,87	0,90	0,87	0,90
M40-2	0,01	0,25	0,11	0,16	0,02	0,16	0,12	0,09
M40-3	0,48	0,63	0,50	0,60	0,50	0,60	0,52	0,57

For solving the repair planning problem, several approaches can be considered. In this chapter, we analyze the results which are obtained through:

1. A constraint-based approach (cf. Appendix D.1), which includes a basic branch-and-bound algorithm (i.e., complete search).

2. A generic planning approach (cf. Appendix D.2), which is used by a solver which performs an incomplete search with a stopping condition based on the marginal improvement of the plan metric.

Typically, complete search algorithms are suitable when a great time is available for execution, while incomplete searches usually get better results in a little time.

Table 6.7 shows a comparison between the quality of the solutions which are found by both approaches. It is possible to see that the quality of the solutions which are found by both approaches strongly depends on the type of problem, obtaining, in general, values close to 0 and 1 in most cases. Moreover, in general, CBP takes more advantages that SGPlan when multi-mode activities are included (cf. Tables 6.4 and 6.6).

6.5 Related Work

Several research groups have integrated AI techniques with BPM systems. As follows, some of the similar works are briefly described, explaining the main differences between them and the current approach.

Some related works integrate P&S tools in BPM systems for the modelling phase, such as (Alves et al., 2008; González-Ferrer et al., 2009; R-Moreno et al., 2007; Hoffmann et al., 2010; De Castro and Marcos, 2009; Ferreira and Ferreira, 2006). These works generate the BP model during the build-time, without taking into account the actual values of the parameters which are known in run-time. Consequently, the initial optimization plan can be obsolete due to the non-updating aspect. On a different way, the current proposal, in order to update the plan in run-time, maintains all the possible alternatives in the BP model, taking into account the actual values of the parameters for updating the P&S of the activities.

On the other hand, most of the related works integrate scheduling tools in BPM systems for the enactment phase, in order to take dispatching decisions as to which activity should be executed using a resource when it becomes free (dynamic scheduling). As follows, a representative set of them are briefly summarized. One of the first works, (Zhao and Stohr, 1999), developed a framework for temporal workflow management, including turnaround time predication, time allocation, and activity prioritization. In a related way, (Son and Kim, 2001) proposes a schema for maximizing the number of workflow instances satisfying a predetermined deadline, based on a method for finding out the critical activities. Moreover, (Ha et al., 2006) proposes a set of process execution rules based on individual worklists, and it develops algorithms for the activity assignment in order to maximize the overall process efficiency, taking the resources capacities into account. Furthermore, (Rhee et al., 2008) presents a Theory of Constraints (TOC)-based method for improving the efficiency of BPs. Additionally, (Tjoa et al., 2010) presents an approach for dynamic activity and resource allocation using risk-aware business process simulations to facilitate and improve the planning and the analysis of the BP activities.

There are several differences between the works (Zhao and Stohr, 1999; Son and Kim, 2001; Ha et al., 2006; Rhee et al., 2008) and the current approach. First, in the current proposal the selection of the activities to be executed (planning) is done in run-time, besides the resource allocation, while the previously summarized works only consider the prioritization of the activities using the same resource (scheduling). Secondly, the current approach consider several objective functions, while the other works only focus on the temporal aspects of the processes for allocating the resources.

Related to the proposed approach, (Marrella and Mecella, 2011) presents a technique which is based on continuous planning in order to automatically cope with unexpected changes. The dynamic domain of the processes is specified through a second-order logic formalism (SitCalc), and a high level interpreter (IndiGolog) is used to control the execution of the actions in the BP enactment. Unlike our approach, any objective function is considered to be optimized. Moreover, while our approach is based on translating P&S problems to the standard BPMN language (and hence the models can be usually executed by a BPM engine), in (Marrella and Mecella, 2011) the process activities need to be specified in SitCalc and be executed through IndiGolog.

To the best of our knowledge, there is not any proposal for selecting the suitable activities in run-time in order to optimize some functions in BP environments as well as considering reaching several goals (planning).

Related to And/Or graphs for representing P&S problems, a similar representation can be found in (Barták and O., 2007), that proposes an extension of temporal networks by parallel and alternative branching to represent some kinds of P&S problems. Furthermore, in (Bae et al., 2004) the concept of blocks that can classify BP flows into several patterns: iterative block, serial block and parallel block including *AND* and *OR* structures, is proposed.

Related to assembly planning, most approaches used for choosing optimal assembly plans employ different kind of rules in order to avoid difficult activities or awkward intermediate subassemblies (Homem de Mello and Sanderson, 1991; Goldwasser and Motwani, 1999). In other context, disassembly planning has been object of different studies, varying from maintenance or repair purposes to recycle or recovery of useful materials (Li and Zhang, 1995; Lambert, 1997, 1999). Different techniques have been used for solving those problems, from mathematical programming to a variety of methods related to artificial intelligence (Lambert, 2003).

Chapter 7

Conclusions

The research works which are presented in this document are based on the combination of several research areas which have been widely referenced and explored: planning and scheduling (P&S), constraint programming (CP), and business process management (BPM).

Although the interest in the application of artificial intelligence techniques, and in particular P&S and CP, to improve the management of business processes has grown in last years, there are some interesting and innovative points which were unexploited and which are addressed in the current Thesis Dissertation.

Specifically, a constraint-based approach for **generating optimized BP enactment plans from declarative process specifications** (cf. Chapter 3) is proposed in order to optimize the performance of a process, according to objective functions like minimizing the overall completion time. This approach overcomes the problems which are related to imperative BP model specifications due to the tacit nature of human knowledge is often an obstacle to eliciting accurate process models (Ferreira and Ferreira, 2006). As mentioned in previous chapters, the generated optimized BP enactment plans can be apply to greatly improve the overall BPM life cycle (Weske, 2007). Two of these applications are detailed in the current Thesis Dissertation: (1) giving users of flexible BPMSs recommendations during run-time, and (2) generating optimized imperative BP models.

The former approach consists on **giving users of flexible BPMSs recommendations during run-time** (cf. Chapter 4) so that performance objective functions of processes are optimized. Due to their flexible nature, frequently several ways to execute declarative process models exist. Typically, given a certain partial trace (i.e., reflecting the current state of the process instances), users can choose from several enabled activities which activity to execute next. This selection, however, can be quite challenging since performance goals of the process should be considered, and users often do not have an understanding of the overall process. Moreover, optimization of objective functions requires that resource capacities are

considered. Therefore, recommendation support is needed during BP execution, especially for inexperienced users (van Dongen and van der Aalst, 2005). In our proposal, recommendations on possible next steps are generated taking the partial trace and the optimized plans into account. Furthermore, in the proposed approach, replanning is supported if actual traces deviate from the optimized enactment plans.

The latter approach, i.e., **generating optimized imperative BP models from declarative specifications** (cf. Chapter 5) is motivated due to BP models are usually defined manually by business analysts through imperative languages. Furthermore, allocating resources is an additional challenge since scheduling may significantly impact business process performance. Therefore, the manual specification of process models can be very complex and time-consuming, potentially leading to non-optimized models or even errors can be generated. Moreover, the result of process modelling is typically a static plan of actions, which is difficult to adapt to changing procedures or to different business goals. With the proposed approach, process models can be adapted to changing procedures or to different business goals, since imperative process models can dynamically be generated from static constraint-based specifications. Moreover, the automatic generation of BP models can deal with complex problems of great size in a simple way. Therefore, a wide study of several aspects can be carried out, such as those related to the requirement of resources of different roles, or the estimated completion time for the BP enactment, by starting from different declarative specifications.

Lastly, a proposal for **modelling and enacting BPs that involve P&S decisions** (cf. Chapter 6) is presented. The execution of most BPs entails, in some way, scheduling decisions since the activities to be executed may compete for some shared resources. In these cases, it is necessary to allocate the resources in a suitable way, usually optimizing some objectives. During process execution, scheduling decisions are typically made by the BPM systems (BPMSs), by automatically assigning activities to resources (Russell et al., 2005). To lesser measure, planning problems are present in BP executions when, in some points, several possible execution branches exist, and the selection of the suitable one depends on the BP goal and/or on the optimization of some functions. Since BP models are typically specified in an imperative way, most of the planning decisions are taken in the modelling phase. Specifically, the ordering and the selection of the activities to be executed (planning) in the BP enactment are specified in the BP design time, when only estimated values for several parameters can be analyzed. However, there are BPs which entail complex planning decisions which can greatly be influenced by the values of several unpredictable parameters, whose actual value is known in run time. The main contribution of the proposed approach is that both P&S decisions are taken in BP run-time, providing the process mana-

gement with efficiency and flexibility, and avoiding the drawbacks of taking these decisions during the design phase.

The motivation and the interest related to all the approaches which are presented in the current document are strongly justified. Furthermore, discussions related to the advantages, drawbacks and limitations of each proposal are included. In addition, the validation of the proposed approaches through the analysis of different performance measures related to a range of test models of varying complexity is included. Moreover, the most related work together with the overcomings and innovations of the proposed approaches are also presented.

Chapter 8

Future Work

In the current document, several research works are presented. All the proposed approaches can be extended by including interesting aspects which remain to be exploited in its related fields.

In a first place, regarding the generation of optimized enactment plans from declarative process specifications (cf. Chapter 3), the next ideas are intended to be explored:

- The inclusion of more general templates in the considered BP declarative specifications, e.g., choice or metric temporal constraints (Montali, 2009).
- The analysis of further objective functions, e.g., cost or robustness, and the optimization of all the overall these functions at the same time (multi-objective optimization).
- The consideration of the non-temporal data perspective in the declarative process specification (Montali, 2009).

All these extensions have a great influence in giving users of flexible BPMSs recommendations for the optimized execution of BPs (cf. Chapter 4) and in the generation of optimized BP models from declarative specifications (cf. Chapter 5), since the generation of optimized enactment plan are used as a starting point for both proposals.

Furthermore, there are other interesting applications of the generated optimized enactment plans which are intended to be explored, such that:

- Simulation: Simulation of BPs can be effectively used for analyzing processes and for improving BP models. BP simulation presents a ”fast-forward” view on a current business process, so that the generated simulation models can accurately reflect the real-world process of interest. One interesting application of BP simulation is to identify unbalances between the resources

required for executing a particular process and the available resources (Reijers and van der Aalst, 1999). Moreover, the effects of alternative resources schedules can be investigated. As future work, a simulation engine can analyze the generated BP enactment plans for making simulations from the declarative BP model, and the results can be studied in order to analyze and to enhance the current BP model (Process Design & Analysis phase of BPM life cycle (Weske, 2007)).

- Time Prediction: There are many scenarios where it is useful to have reliable time predictions (van der Aalst et al., 2011). As future work, the generated optimized BP enactment plans can be used by a prediction engine since, as stated, time information is available. For a given process instance state, the expected completion time for instances and activities can be calculated by taking the end time of the remaining activities of the optimized plans into account.

Related to giving users of flexible BPMSs recommendations for the optimized execution of BPs (cf. Chapter 4), the integration of the proposed approach with a testsuite for recommendations in ConDec (Westergaard, 2011) is intended to be achieved. In this testsuite, a model of user behavior can be specified. The proposal of this integration consists of checking the suitability of the proposed approach when simulating the actual execution of the BPs.

Regarding the generation of optimized BP models from declarative specifications (cf. Chapter 5), obtaining configurable process models from several optimized plans is intended. These configurable process models can be generated by extracting the common parts of the original process models, creating a single copy of them, and appending the differences as branches of configurable connectors (La Rosa et al., 2010). In this way, the generated configurable models capture all the behavior of the original models.

On the other hand, related to the proposal for planning and scheduling BPs in run-time (cf. Chapter 6), it is intended to analyze further kinds of BP problems involving planning and scheduling aspects. Furthermore, a bidirectional translation between planning languages, as PDDL, and BP model or execution languages, is intended to be explored. Moreover, further objective functions can be considered.

Appendices

Appendix A

ConDec-R Templates

As stated, in general, if not restricted by any constraints BP activities are assumed to be executed several times (Pesic and van der Aalst, 2006). Henceforth, $nt(A)$ refers to the number of times that the repeated activity A is executed; A_i represents the P&S activity related to the i-th execution of A; and $st(A_i)$ and $et(A_i)$ represent the start and the end times of A_i, respectively. It should be clarified that the constraints $\forall i : 1 \leq i < nt(A) : et(A_i) \leq st(A_{i+1})$ hold for each repeated activity A.

In Fig. A.1, some representative examples of ConDec-R templates are graphically represented. Specifically, three precedence relations between two repeated activities, A and B, are shown. As stated earlier, several executions of the same BP activity can be modelled as a sequence of single P&S activities. In this figure, the P&S activity A_i represents the i-th execution of the repeated activity A (A_i), and the arrow represents:

- A precedence relation between two P&S activities A_i and B_j, when it appears between two activities, which means that $et(A_i) \leq st(B_j)$.
- A precedence relation between a P&S activity A_i and a set S of P&S activities, when it appears between an activity and a dotted rectangle which encloses a set of activities, which means that $\exists B_j \in S : et(A_i) \leq st(B_j)$.
- A precedence relation between a set S of P&S activities and a P&S activity B_j, when it appears between a dotted rectangle which encloses a set of activities, and an activity, which means that $\exists A_i \in S : et(A_i) \leq st(B_j)$.

In a similar way, a special arrow (wider than the other arrows and with a big dot in its origin) which appears between two P&S activities, A and B, shows that A must be executed **immediately** before B ($et(A) = st(B)$). In a similar way, this can be defined for a set of activities. More details about Fig. A.1 are shown in the definition of the related templates.

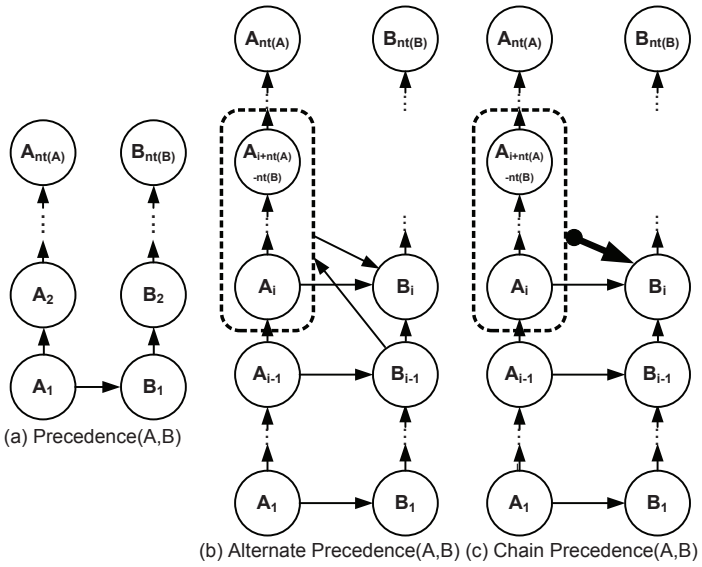


Figure A.1: Precedence templates when $nt(B) > 0$.

The ConDec-R templates, based on ConDec templates, together with some examples of valid and invalid traces[1], are listed as follows[2]. A full description of the ConDec templates is included in the report (van der Aalst and Pesic, 2006b). These templates can be easily modified to include further possibilities.

- Existence(N,A): A must be executed more than or equal to N times, $nt(A) \geq N$.
- Absence(N,A): A must be executed less than N times, $nt(A) < N$.
- Exactly(N,A): A must be executed exactly N times, $nt(A) = N$.
- Responded Existence(A,B): If A is executed, then B must also be executed either before or after A, $nt(A) > 0 \Rightarrow nt(B) > 0$. For example, when Responded Existence(A,B) holds, <B>, <AB> or <BA> are valid traces, and <A> is an invalid trace since the execution of A requires the execution of B.
- CoExistence(A,B): The execution of A requires the execution of B, and vice versa, $nt(A) > 0 \Longleftrightarrow nt(B) > 0$. For example, when CoExistence(A,B) holds, <AB> or <BA> are valid traces, and <B> is an invalid trace since the execution of B requires the execution of A.

[1]For the sake of clarity, no parallelism between the activities is considered in the examples, i.e., trace $<A^1A^2\ldots A^n>$ means that $\forall i : 1 \leq i < n, et(A^i) = st(A^{i+1})$.

[2]For simplification, only non-branched templates are shown.

- Precedence(A,B): Before the execution of B, A must have been executed, $nt(B) > 0 \Rightarrow (nt(A) > 0) \wedge (et(A_1) \leq st(B_1))$. As can be seen in Fig. A.1(a), this relation implies that A_1 must precede B_1 in the case that $nt(B) > 0$. For example, when Precedence(A,B) holds, <ABBBA> is a valid trace, and <BAABB> is an invalid trace since the first B is executed before any A.

- Response(A,B): After the execution of A, B must be executed, $nt(A) > 0 \Rightarrow (nt(B) > 0) \wedge (st(B_{nt(B)}) \geq et(A_{nt(A)}))$. For example, when Response(A,B) holds, <BAABB> is a valid trace, and <ABBBA> is an invalid trace since after the last execution of A, B is not executed.

- Succession(A,B): Relations Precedence(A,B) and Response(A,B) hold. For example, when Succession(A,B) holds, <ABABB> is a valid trace, and <BABBA> is an invalid trace since the first B is executed before any A (moreover, after the last execution of A, B is not executed).

- Alternate Precedence(A,B): Before the execution of B, A must have been executed, and between each two executions of B, A must be executed. This implies that:

  1. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) \geq nt(B)$.
  2. Between each two executions of B, A must be executed at least once. Specifically, between the $(i-1)$-th and the i-th execution of B, the earliest execution of A that can exist is i, and hence A_{i-1} must precede B_{i-1} (as can be seen in Fig. A.1(b)). In a similar way, between the $(i-1)$-th and the i-th execution of B, the latest execution of A that can exist is $i + nt(A) - nt(B)$, and hence B_i must precede $A_{i+nt(A)-nt(B)+1}$. This can also be seen in Fig. A.1(b), where the possible activities to be executed between the $(i-1)$-th and the i-th execution of B are framed within the dotted rectangle. $\forall i : 2 \leq i \leq nt(B) : \exists j : i \leq j \leq i + nt(A) - nt(B) : st(A_j) \geq et(B_{i-1}) \wedge et(A_j) \leq st(B_i)$.
  3. Before the execution of B, A must be executed: $st(B_1) \geq et(A_1)$.

  For example, when Alternate Precedence(A,B) holds, <ABAABABA> is a valid trace, and <ABAABBAA> is an invalid trace since between the second and the third execution of B, there is not any A.

- Alternate Response(A,B): After the execution of A, B must be executed, and between each two executions of A, there must be at least one execution of B. This implies:

1. The number of times that B is executed must be greater than or equal to the number of times that A is executed: $nt(B) \geq nt(A)$.
2. Between each two executions of A, B must be executed at least once. Specifically, between the i-th and the $(i+1)$-th execution of A, the earliest execution of B that can exist is i, and hence B_{i-1} must precede A_i. In a similar way, between the i-th and the $(i+1)$-th execution of B, the latest execution of A that can exist is $i+nt(B)-nt(A)-1$, and hence A_i must precede $B_{i+nt(B)-nt(A)}$. $\forall i : 1 \leq i < nt(A) : \exists j : i \leq j \leq i+nt(B)-nt(A)-1 : st(B_j) \geq et(A_i) \wedge et(B_j) \leq st(A_{i+1})$.
3. After the execution of A, B must be executed: $st(B_{nt(B)}) \geq et(A_{nt(A)})$.

For example, when Alternate Response(A,B) holds, <BABABBAB> is a valid trace, and <BAABBABB> is an invalid trace since between the first and the second execution of A, there is not any B.

- Alternate Succession(A,B): Both the relations AlternatePrecedence(A,B) and AlternateResponse(A,B) hold. For example, when Alternate Succession(A,B) holds, <ABABAB> is a valid trace, and <ABABBA> is an invalid trace since between the second and the third execution of B, there is not any A.

- Chain Precedence(A,B): **Immediately** before the execution of B, A must be executed. It implies that:

  1. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) \geq nt(B)$.
  2. Immediately before each execution of B, A must be executed. Specifically, before the i-th execution of B, the earliest execution of A that can exist is i. In a similar way, before the i-th execution of B, the latest execution of A that can exist is $i+nt(A)-nt(B)$. $\forall i : 1 \leq i \leq nt(B) : \exists j : i \leq j \leq i+nt(A)-nt(B) : et(A_j)$=$st(B_i)$.

  This is shown in Fig. A.1(c), where a special arrow (wider than the other arrows and with a big dot in its origin) shows that A must be executed **immediately** before B. For example, when Chain Precedence(A,B) holds, <ABAABABA> is a valid trace, and <ABAABBAA> is an invalid trace since immediately before the third execution of B, there is not any A.

- Chain Response(A,B): **Immediately** after the execution of A, B must be executed. It implies:

  1. The number of times that B is executed must be greater than or equal to the number of times that A is executed: $nt(B) \geq nt(A)$.

2. Immediately after each execution of A, B must be executed. Specifically, before the i-th execution of A, the earliest execution of B that can exist is i. In a similar way, after the i-th execution of A, the latest execution of B that can exist is $i + nt(B) - nt(A) - 1$. $\forall i : 1 \leq i \leq nt(A) : \exists j : i \leq j \leq i + nt(B) - nt(A) - 1 : st(B_j) = et(A_i)$.

For example, when Chain Response(A,B) holds, <BABABBAB> is a valid trace, and <BAABBABB> is an invalid trace since immediately after the first execution of A, there is not any B.

- Chain Succession(A,B): Both the relations Chain Precedence(A,B) and Chain Response(A,B) hold. For example, when Chain Succession(A,B) holds, <ABABAB> is a valid trace, and <ABABBA> is an invalid trace since immediately before the third execution of B, there is not any A.

- Responded Absence and Not CoExistence(A,B): If B is executed, then A cannot be executed, and vice versa, $((nt(A) > 0) \cdot (nt(B) > 0)) == 0$. For example, when Responded Absence(A,B) holds, <A> or <B> are valid traces, and <BA> is an invalid trace.

- Negation Response, Negation Precedence, Negation Succession(A,B): After the execution of A, B cannot be executed, i.e., the last execution of B must finish before the start of the first execution of A ($nt(A) > 0 \wedge nt(B) > 0) \Rightarrow et(B_{nt(B)}) \leq st(A_1)$). For example, when Negation Succession(A,B) holds, <BBBA> is a valid trace, and <BBAB> is an invalid trace since the third B is executed after A.

- Negation Alternate Precedence(A,B): Between two executions of B, A cannot be executed, $nt(B) \geq 2 \Rightarrow \forall i : 1 \leq i \leq nt(A) : et(A_i) \leq st(B_1) \vee st(A_i) \geq et(B_nt(B))$. For example, when Negation Alternate Precedence(A,B) holds, <AABBA> is a valid trace, and <ABABA> is an invalid trace since between the first and the second execution of B, A is executed.

- Negation Alternate Response(A,B): Between two executions of A, B cannot be executed, $nt(A) \geq 2 \Rightarrow \forall 1 \leq i \leq nt(B) : et(B_i) \leq st(A_1) \vee st(B_i) \geq et(A_{nt(A)})$. For example, when Negation Alternate Response(A,B) holds, <BBAAB> is a valid trace, and <BABAB> is an invalid trace since between the first and the second execution of A, B is executed.

- Negation Alternate Succession(A,B): Both the relations Negation Alternate Precedence(A,B) and Negation Alternate Response(A,B) hold. For example, when Negation Alternate Succession(A,B) holds, <AABB> is a valid

trace, and <AABBA> is an invalid trace since between the second and the third execution of A, B is executed.

- Negation Chain Succession(A,B): B cannot be executed immediately after the execution of A, $\forall i : 1 \leq i \leq nt(B) : \neg\exists j : 1 \leq j \leq nt(A) : et(A_j) = st(B_i)$. For example, when Negation Chain Succession(A,B) holds, <BACBA> is a valid trace, and <BABA> is an invalid trace since the second B is executed immediately after A.

The ConDec-R templates can be classified either in unary (only one parameter, e.g., ExistenceN or AbsenceN) or binary (two parameters, e.g., Response or Chain Succession) templates.

Appendix B

Filtering Rules for ConDec-R Templates

The constraint-based approach which is proposed for generating optimized BP enactment plans from ConDec-R specifications (cf. Chapter 3) includes specific filtering rules (i.e., responsible for removing values which do not belong to any solution from the domains of variables) for the definition of the high-level relations between the BP activities through global constraints. In this way, the constraints stated in the ConDec-R specification (cf. Def. 20 on page 39) are included in the CSP model through the related filtering rules. These filtering rules facilitate the specification of the problem through global constraints at the same time as they enable the efficiency in the search for solutions to increase.

The developed filtering rules for the basic ConDec (also ConDec-R) high-level constraints, i.e., non-branched templates, are presented in this section. For each relation: (1) the definition of the template, (2) the pseudocode, and (3) the complexity of the related filtering rule, are detailed.

B.1 Existence(A, N)

As stated, A must be executed more than or equal to N times, $nt(A) \geq N$.

```
Existence(A,N) is added ->
    If N > LB(nt(A)) then
        LB(nt(A)) <- N
```

Figure B.1: Filtering Rule for the Existence Template

The *Existence* rule (Fig. B.1) is invoked when the template is added to the constraint model, hence its trigger ”Existence(A,N) is added”.

Proposition 5. *If implemented properly, the time complexity of the rule Existence, which includes all possible recursive calls, is* $\Theta(1)$.

Proof. The *Existence* rule is fired only when the constraint is added, and the time complexity of its execution is constant, hence the complexity of this rule is $\Theta(1)$. □

B.2 Absence(A, N)

As stated, A must be executed fewer than N times, $nt(A) < N$.

```
Absence(A,N) is added ->
    If N-1 < UB(nt(A)) then
        UB(nt(A)) <- N-1
```

Figure B.2: Filtering Rule for the Absence Template

The *Absence* rule (Fig. B.2) is invoked when the template is added to the constraint model, hence its trigger ”Absence(A,N) is added”.

Proposition 6. *If implemented properly, the time complexity of the rule Absence, which includes all possible recursive calls, is* $\Theta(1)$.

Proof. The *Absence* rule is fired only when the constraint is added, and the time complexity of its execution is constant, hence the complexity of this rule is $\Theta(1)$. □

B.3 Exactly(A, N)

As stated, A must be executed exactly N times, $nt(A) = N$.

```
Exactly(A,N) is added ->
    If ! IsBound(nt(A)) then
        VAL(nt(A)) <- N
```

Figure B.3: Filtering Rule for the Exactly Template

The *Exactly* rule (Fig. B.3) is invoked when the template is added to the constraint model, hence its trigger ”Existence(A,N) is added”.

Proposition 7. *If implemented properly, the time complexity of the rule Exactly, which includes all possible recursive calls, is* $\Theta(1)$.

Proof. The *Exactly* rule is fired only when the constraint is added, and the time complexity of its execution is constant, hence the complexity of this rule is $\Theta(1)$. □

B.4 Responded Existence(A, B)

As stated, if A is executed, then B must also be executed either before or after A, $nt(A) > 0 \Rightarrow nt(B) > 0$.

```
Responded Existence(A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed ->
    If LB(nt(A)) > 0 then
        If LB(nt(B)) < 1
            LB(nt(B)) <- 1
    If UB(nt(B)) == 0 then
        If UB(nt(A)) > 0
            UB(nt(A)) <- 0
```

Figure B.4: Filtering Rule for the Responded Existence Template

The *Responded Existence* rule (Fig. B.4) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 8. *If implemented properly, the worst-case time complexity of the rule RespondedExistence, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *RespondedExistence* rule can be fired, at most, n times. This is due to the fact that only a change in the nt variable of a repeated activity can fire this rule, and there are n repeated activities. Moreover, the time complexity of the *RespondedExistence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

B.5 CoExistence(A, B)

As stated, the execution of A forces the execution of B, and vice versa, $nt(A) > 0 \Longleftrightarrow nt(B) > 0$.

The *CoExistence* rule (Fig. B.5) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 9. *If implemented properly, the worst-case time complexity of the rule CoExistence, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *CoExistence* rule can be fired, at most, n times. This is due to the fact that only a change in the nt variable of a repeated activity can fire this rule, and there are n repeated activities. Moreover, the time complexity of the *CoExistence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

```
CoExistence(A,B) is added OR bounds of
nt(A) changed OR bounds of nt(B) changed ->
    If LB(nt(A)) > 0 then
        If LB(nt(B)) < 1
            LB(nt(B)) <- 1
    If UB(nt(B)) == 0 then
        If UB(nt(A)) > 0
            UB(nt(A)) <- 0
    If LB(nt(B)) > 0 then
        If LB(nt(A)) < 1
            LB(nt(A)) <- 1
    If UB(nt(A)) == 0 then
        If UB(nt(B)) > 0
            UB(nt(B)) <- 0
```

Figure B.5: Filtering Rule for the CoExistence Template

B.6 Precedence(A, B)

As stated, before the execution of B, A must have been executed, $nt(B) > 0 \Rightarrow (nt(A) > 0) \wedge (et(A_1) \leq st(B_1))$. As can be seen in Fig. A.1(a), this relation implies that A_1 must precede B_1 in the case that $nt(B) > 0$.

```
Precedence(A,B) is added OR bounds of
nt(A) changed OR bounds of nt(B) changed OR bounds of st(B1)
changed OR bounds of et(A1) changed ->
    If LB(nt(B)) > 0 then
        nt(A) <- nt(A) - {0}
    If UB(nt(A)) == 0 then
        VAL(nt(B)) <- 0
    If LB(et(A1)) > LB(st(B1))then
        LB(st(B1))) <- LB(et(A1))
    If UB(et(A1)) > UB(st(B1)) then
        UB(et(A1)) <- UB(st(B1))
```

Figure B.6: Filtering Rule for the Precedence Template

The *Precedence* rule (Fig. 10) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 10. *If implemented properly, the worst-case time complexity of the rule Precedence, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *Precedence* rule can be fired, at most, $3 \times n$ times. This is due to the fact that only a change in the first execution (*st* and *et* variables of Act_1) or in the *nt* variable of a repeated activity can fire this rule. Moreover, the time complexity of the *Precedence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

B.7 Response(A, B)

As stated, after the execution of A, B must be executed, $nt(A) > 0 \Rightarrow (nt(B) > 0) \wedge (st(B_{nt(B)}) \geq et(A_{nt(A)}))$.

```
Response(A,B) is added OR bounds of
nt(A) changed OR bounds of nt(B) OR bounds of st(BUB(nt(B)))
changed OR bounds of et(AUB(nt(A))) changed ->
    If LB(nt(A)) > 0 then
        nt(B) <- nt(B) - {0}
    If UB(nt(B)) == 0 then
        VAL(nt(A)) <- 0
    If LB(et(AUB(nt(A)))) > LB(st(BUB(nt(B)))))then
        LB(st(BUB(nt(B)))) <- LB(et(AUB(nt(A))))
    If UB(et(AUB(nt(A)))) > UB(st(BUB(nt(B)))) then
        UB(et(AUB(nt(A)))) <- UB(st(B,UB(nt(B))))
```

Figure B.7: Filtering Rule for the Response Template

The *Response* rule (Fig. 11) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 11. *If implemented properly, the worst-case time complexity of the rule Response, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *Response* rule can be fired, at most, $3 \times n$ times. This is due to the fact that only a change in the last execution (*st* and *et* variables of $Act_{nt(Act)}$) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *Response* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

B.8 Succession(A, B)

As stated, relations $Precedence(A,B)$ and $Response(A,B)$ hold.

Proposition 12. *If implemented properly, the worst-case time complexity of the rule Succession, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *Succession* rule can be implemented as the conjunction of *Precendence* and *Response* rules. The complexity of both rules is $O(n)$ (see Prop. 10 and 11), and hence the worst-case time complexity of *Succession* rule is $O(2 \times n)$, equal to $O(n)$. □

B.9 Alternate Precedence(A, B)

As stated, before the execution of B, A must have been executed, and between each two executions of B, A must be executed. It implies that:

1. The number of times that A is executed must be greater than or equal to the number of times that B is executed: $nt(A) >= nt(B)$.

2. Between each two executions of B, A must be executed at least once. Specifically, between the $(i-1)$-th and the i-th execution of B, the earliest execution of A that can exist is i, and hence A_{i-1} must precede B_{i-1} (as can be seen in Fig. A.1(b)). In a similar way, between the $(i-1)$-th and the i-th execution of B, the latest execution of A that can exist is $i + nt(A) - nt(B)$, and hence B_i must precede $A_{i+nt(A)-nt(A)+1}$. This can also be seen in Fig. A.1(b), where the possible activities to be executed between the $(i-1)$-th and the i-th execution of B are framed within the dotted rectangle. $\forall i : 2 \leq i \leq nt(B) : \exists j : i \leq j \leq i + nt(A) - nt(B) : st(A_j) \geq et(B_{i-1}) \wedge et(A_j)) \leq st(B_i)$.

3. Before B, A must be executed: $st(B_1) \geq et(A_1)$.

Proposition 13. *If implemented properly, the worst-case time complexity of the rule AlternatePrecedence, which includes all possible recursive calls, is* $O(n \times nt^3)$*, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable* $nt(Act)$ *domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *AlternatePrecedence* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *AlternatePrecedence* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

B.10 Alternate Response(A, B)

As stated, after the execution of A, B must be executed, and between each two executions of A, there must be at least one execution of B. It implies:

1. The number of times that B is executed must be greater than or equal to the number of times that A is executed: $nt(B) >= nt(A)$.

```
Alternate Precedence (A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
st(Ai) for any i changed OR bounds of et(Ai) for any i changed OR
bounds of st(Bi) for any i changed OR bounds of et(Bi) for any i
changed ->
   if (LB(nt(A)) < LB(nt(B))) {LB(nt(A)) <- LB(nt(B))}
   if (UB(nt(B)) > UB(nt(A))) {UB(nt(B)) <- UB(nt(A))}
   for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi;
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi
   }
   for (int i = 1; i < LB(nt(B)); i++){
      int dif = UB(nt(A)) - LB(nt(B));
      SchedulingActivity a = Ai+dif+1;
      SchedulingActivity b = Bi;
      if (LB(et(b)) > LB(st(a))) {LB(st(a)) <- LB(et(b))}// Bi -> Ai+dif+1
      if (UB(st(a)) < UB(et(b))) {UB(et(b)) <- UB(st(a))}// Bi -> Ai+dif+1
   }
   for (int i = 2; i <= UB(nt(B)); i++){  // force exists A between Bi-1 and Bi
      int dif = UB(nt(A)) - max(i,LB(nt(B)));
      SchedulingActivity b1 = Bi-1;
      SchedulingActivity b2 = Bi;
      int j = i; // Candidate As between Bi-1 and Bi
      SchedulingActivity aFor;
      int possible = 0;
      while (j <= (i + dif) && possible < 2) {
         SchedulingActivity aPos = Aj;
         //If Bi-1->aPos->Bi possible
        if (UB(st(aPos))>=LB(et(b1)) && LB(et(aPos))<=UB(st(b2))){
            possible++;
            aFor = aPos;
        }
        j++;
      } // end while j
      if (possible == 1){ // force Bi-1 -> aFor -> Bi
        // Bi-1 -> aFor
        if (LB(et(b1))>LB(st(aFor))){LB(st(aFor)) <- LB(et(b1))}
        if (UB(st(aFor))<UB(et(b1)) {UB(et(b1)) <- UB(st(aFor))}
        // aFor -> Bi
        if (LB(et(aFor))>LB(st(b2))){LB(st(b2) <- LB(et(aFor))}
        if (UB(st(b2))<UB(et(aFor))){UB(et(aFor)) <- UB(st(b2))}
      } // end if
      if(possible == 0)
        return Failure;
   }//end for i
```

Figure B.8: Filtering Rule for the Alternate Precedence Template

2. Between each two executions of A, B must be executed at least once. Specifically, between the i-th and the $(i+1)$-th execution of A, the earliest execution of B that can exist is i, and hence B_{i-1} must precede A_i. In a similar way, between the i-th and the $(i+1)$-th execution of B, the latest execution of A that can exist is $i+nt(B)-nt(A)-1$, and hence A_i must precede $B_{i+nt(B)-nt(A)}$. $\forall i : 1 \leq i < nt(A) : \exists j : i \leq j \leq i+nt(B)-nt(A)-1 : st(B_j) \geq et(A_i) \wedge et(B_j) \leq st(A_{i+1})$.

3. After A, B must be executed: $st(B_{nt(B)}) \geq et(A_{nt(A)})$.

```
Alternate Response (A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
st(Ai) for any i changed OR bounds of et(Ai) for any i changed OR
bounds of st(Bi) for any i changed OR bounds of et(Bi) for any i
changed ->
   if (LB(nt(B)) < LB(nt(A))) {LB(nt(B)) <- LB(nt(A))}
   if (UB(nt(A)) > UB(nt(B))) {UB(nt(A)) <- UB(nt(B))}
   for (int i = 2; i <= UB(nt(A)); i++){
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi-1;
      if (LB(et(b)) > LB(st(a))) {LB(st(a)) <- LB(et(b))}// Bi-1 -> Ai
      if (UB(st(a)) < UB(et(b))) {UB(et(b)) <- UB(st(a))}// Bi-1 -> Ai
   }
   for (int i = 1; i <= LB(nt(A)); i++){
      int dif = UB(nt(B)) - LB(nt(A));
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi+dif;
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi+dif
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi+dif
   }
   for (int i = 2; i <= UB(nt(A)); i++){  // force exists B between Ai-1 and Ai
      int dif = UB(nt(B)) - max(i,LB(nt(A)))
      SchedulingActivity a1 = Ai-1;
      SchedulingActivity a2 = Ai;
      int j = i - 1; // Candidate Bs between Ai-1 and Ai
      SchedulingActivity bFor;
      int possible = 0;
      while (j <= (i + dif - 1) && possible < 2) {
         SchedulingActivity bPos = Bj;
         //If Ai-1->bPos->Ai possible
         if (UB(st(bPos))>=LB(et(a1)) && LB(et(bPos))<=UB(st(a2))){
            possible++;
            bFor = bPos;
         }
         j++;
        } // end while j
        if (possible == 1){ // force Ai-1->bFor-> Ai
            // Ai-1 -> bFor
            if (LB(et(a1))>LB(st(bFor))){LB(st(bFor)) <- LB(et(a1))}
            if (UB(st(bFor))<UB(et(a1)) {UB(et(a1)) <- UB(st(bFor))}
            // bFor -> Ai
            if (LB(et(bFor))>LB(st(a2))){LB(st(a2) <- LB(et(bFor))}
            if (UB(st(a2))<UB(et(bFor))){UB(et(bFor)) <- UB(st(a2))}
        } // end if
        if(possible == 0)
            return Failure;
     }// end for i
```

Figure B.9: Filtering Rule for the Alternate Response Template

Proposition 14. *If implemented properly, the worst-case time complexity of the rule AlternateResponse, which includes all possible recursive calls, is* $O(n \times nt^3)$*, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable* $nt(Act)$ *domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *AlternateResponse* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *AlternateResponse* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

B.11 Alternate Succession(A, B)

As stated, relations $AlternatePrecedence(A,B)$ and $AlternateResponse(A,B)$ hold.

Proposition 15. *If implemented properly, the worst-case time complexity of the rule AlternateSuccession, which includes all possible recursive calls, is* $O(n \times nt^3)$*, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable* $nt(Act)$ *domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *AlternateSuccession* rule can be implemented as the conjunction of *AlternatePrecendence* and *AlternateResponse* rules. The complexity of both rules is $O(n \times nt^3)$ (see Prop. 13 and 14), and hence the worst-case time complexity of *AlternateSuccession* rule is $O(2 \times n \times nt^3)$, equal to $O(n \times nt^3)$. □

B.12 Chain Precedence(A, B)

As stated, **immediately** before *B*, *A* must be executed. It implies that:

1. The number of times that *A* is executed must be greater than or equal to the number of times that *B* is executed: $nt(A) >= nt(B)$.

2. Immediately before each execution of *B*, *A* must be executed. Specifically, before the *i*-th execution of *B*, the earliest execution of *A* that can exist is *i*. In a similar way, before the *i*-th execution of *B*, the latest execution of *A* that can exist is $i + nt(A) - nt(B)$. $\forall i : 1 \leq i \leq nt(B) : \exists j : i \leq j \leq i + nt(A) - nt(B) : et(A_j)$=$st(B_i)$.

Proposition 16. *If implemented properly, the worst-case time complexity of the rule ChainPrecedence, which includes all possible recursive calls, is* $O(n \times nt^3)$*, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable* $nt(Act)$ *domain. This upper bound obtains the same value for all the ConDec-R activities.*

```
Chain Precedence (A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
et(Ai) for any i changed OR bounds of st(Bi) for any i changed OR
bounds of et(Bi) for any i changed ->
   if (LB(nt(A)) < LB(nt(B))) {LB(nt(A)) <- LB(nt(B))}
   if (UB(nt(B)) > UB(nt(A))) {UB(nt(B)) <- UB(nt(A))}
   for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi;
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi
   }
   for (int i = 1; i <= UB(nt(B)); i++){  // force exists A before Bi
      int dif = UB(nt(A)) - max(i,LB(nt(B)));
      SchedulingActivity b = Bi;
      int j = i; // Candidate As before Bi
      SchedulingActivity aFor;
      int possible = 0;
      while (j <= (i + dif) && possible < 2) {
         SchedulingActivity aPos = Aj;
         //If aPos'->Bi possible
         if (LB(et(aPos))<=UB(st(b)) && LB(st(b))<=UB(et(aPos)))
            possible++;
            aFor = aPos;
         }
         j++;
        } // end while
        if (possible == 1){ // force aFor '-> Bi
            forcedValue = true;
            // aFor '-> Bi
            if (LB(et(aFor))>LB(st(b))){LB(st(b) <- LB(et(aFor))}
            if (UB(et(aFor))>UB(st(b))){UB(st(b) <- UB(et(aFor))}
            if (LB(st(b))<LB(et(aFor))){LB(et(aFor)) <- LB(st(b))}
            if (UB(st(b))<UB(et(aFor))){UB(et(aFor)) <- UB(st(b))}
        } // end if
        if(possible == 0)
            return Failure;
     }// end for i
```

Figure B.10: Filtering Rule for the Chain Precedence Template

Proof. The *ChainPrecedence* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *ChainPrecedence* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

B.13 Chain Response(A, B)

As stated, **immediately** after *A*, *B* must be executed. It implies:

1. The number of times that *B* is executed must be greater than or equal to the number of times that *A* is executed: $nt(B) >= nt(A)$.

2. Immediately after each execution of A, B must be executed. Specifically, before the i-th execution of A, the earliest execution of B that can exist is i. In a similar way, after the i-th execution of A, the latest execution of B that can exist is $i + nt(B) - nt(A) - 1$. $\forall i : 1 \leq i <= nt(A) : \exists j : i \leq j \leq i + nt(B) - nt(A) - 1 : st(B_j) = et(A_i)$.

```
Chain Response (A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
et(Ai) for any i changed OR bounds of st(Bi) for any i changed OR
bounds of et(Bi) for any i changed ->
   if (LB(nt(B)) < LB(nt(A))) {LB(nt(B)) <- LB(nt(A))}
   if (UB(nt(A)) > UB(nt(B))) {UB(nt(A)) <- UB(nt(B))}
   for (int i = 1; i <= LB(nt(A)); i++){
      int dif = UB(nt(B)) - max(i,LB(nt(A)))
      SchedulingActivity a = Ai;
      SchedulingActivity b = Bi+dif;
      if (LB(et(a)) > LB(st(b))) {LB(st(b)) <- LB(et(a))}// Ai -> Bi+dif
      if (UB(st(b)) < UB(et(a))) {UB(et(a)) <- UB(st(b))}// Ai -> Bi+dif
   }
   for (int i = 1; i <= UB(nt(A)); i++){  // force exists B after Ai
      int dif = UB(nt(B)) - max(i,LB(nt(A)))
      SchedulingActivity a = Ai;
      int j = i - 1; // Candidate Bs after Ai
      SchedulingActivity bFor;
      int possible = 0;
      while (j <= (i + dif - 1) && possible < 2) {
         SchedulingActivity bPos = Bj;
         //If Ai->bPos possible
         if (UB(st(bPos))>=LB(et(a)) && LB(et(bPos))<=UB(st(a))){
            possible++;
            bFor = bPos;
         }
         j++;
        } // end while
        if (possible == 1){ // force Ai'->bFor
            // Ai '-> bFor
            if (LB(et(a))>LB(st(bFor))){LB(st(bFor)) <- LB(et(a))}
            if (UB(et(a))>UB(st(bFor))){UB(st(bFor)) <- UB(et(a))}
            if (LB(st(bFor))<LB(et(a)) {LB(et(a)) <- LB(st(bFor))}
            if (UB(st(bFor))<UB(et(a)) {UB(et(a)) <- UB(st(bFor))}
        } // end if
        if(possible == 0)
            return Failure;
     }// end for i
```

Figure B.11: Filtering Rule for the Chain Response Template

Proposition 17. *If implemented properly, the worst-case time complexity of the rule ChainResponse, which includes all possible recursive calls, is* $O(n \times nt^3)$*, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable* $nt(Act)$ *domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *ChainResponse* rule can be fired, , at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables

of Act_i) or in the nt variable of an activity can fire this rule. Moreover, the time complexity of the *ChainResponse* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

B.14 Chain Succession(A, B)

As stated, relations $ChainPrecedence(A,B)$ and $ChainResponse(A,B)$ hold.

Proposition 18. *If implemented properly, the worst-case time complexity of the rule ChainSuccession, which includes all possible recursive calls, is $O(n \times nt^3)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *ChainSuccession* rule can be implemented as the conjunction of *Chain Precendence* and *ChainResponse* rules. The complexity of both rules is $O(n \times nt^3)$ (see Prop. 16 and 17), and hence the worst-case time complexity of *Chain Succession* rule is $O(2 \times n \times nt^3)$, equal to $O(n \times nt^3)$. □

B.15 Responded Absence(A, B) and Not CoExistence (A, B)

As stated, if B is executed, then A cannot be executed, and vice versa, $((nt(A) > 0) \cdot (nt(B) > 0)) == 0$.

```
Responded Absence(A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed ->
    If LB(nt(A)) > 0 then
        nt(B) <- 0
    If LB(nt(B)) > 0 then
        nt(A) <- 0
```

Figure B.12: Filtering Rule for the Responded Absence Template

The *Responded Absence* rule (Fig. B.12) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 19. *If implemented properly, the worst-case time complexity of the rule RespondedAbsence, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *RespondedAbsence* rule can be fired, at most, n times. This is due to the fact that only a change in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *RespondedAbsence* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

B.16 Negation Response(A, B), Negation Precedence(A, B) and Negation Succession(A, B)

As stated, after the execution of A, B cannot be executed, $(nt(A) > 0 \wedge nt(B) > 0) \Rightarrow st(B_{nt(B)}) \leq et(A_1)$.

```
Negation Response(A,B) is added OR
bounds of nt(A) changed OR bounds of nt(B) changed OR bounds of
et(A1) changed OR bounds of st(BUB(nt(B))) changed ->
    If LB(nt(A)) > 0 && LB(nt(B)) > 0 then
        If LB(st(BUB(nt(B)))) > LB(et(A1))then
            LB(et(A1)) <- LB(st(BUB(nt(B))))
        If UB(et(A1)) < UB(st(BUB(nt(B)))) then
            UB(st(BUB(nt(B)))) <- UB(et(A1))
```

Figure B.13: Filtering Rule for the Negation Response Template

The *Negation Response* rule (Fig. B.13) is invoked when the template is added to the constraint model or when the domain bounds of some variables are updated.

Proposition 20. *If implemented properly, the worst-case time complexity of the rule NegationResponse, which includes all possible recursive calls, is $O(n)$, where n is the number of Repeated (ConDec-R) Activities of the problem.*

Proof. The *NegationResponse* rule can be fired, at most, $3 \times n$ times. This is due to the fact that only a change in the first execution (*et* variable of Act_1) or in the last execution (*st* variable of $Act_{nt(Act)}$) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *NegationResponse* rule execution is constant, and hence the worst-case complexity of this rule is $O(n)$. □

B.17 Negation Alternate Precedence(A, B)

As stated, between two executions of B, A cannot be executed, $nt(B) \geq 2 \Rightarrow \forall i : 1 \leq i \leq nt(A) : et(A_i)) \leq et(B_1) \vee st(A_i) \geq st(B_{nt(B)})$.

Proposition 21. *If implemented properly, the worst-case time complexity of the rule NegationAlternatePrecedence, which includes all possible recursive calls, is $O(n \times nt^2)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.*

```
Negation Alternate Precedence (A,B) is
added OR bounds of nt(A) changed OR bounds of nt(B) changed OR
bounds of et(B1) changed OR bounds of st(Bnt(B)) changed OR bounds
of st(Ai) for any i changed OR bounds of et(Ai) for any i changed
->
   if (LB(nt(B)) >= 2) then
    for (int i = 1; i <= UB(nt(A)); i++){
      SchedulingActivity a = Ai;
      SchedulingActivity bF = B1; // bFirst
      SchedulingActivity bL = BUB(nt(B)); // bLast
      if (UB(st(a)) < LB(st(bL))) then // a can not start after bL => a must finish before bF
        if (UB(et(bF)) < UB(et(a))) then
            UB(et(a)) <- UB(et(bF))
      if (LB(et(a)) > UB(et(bF))) then // a can not finish before bF => a must start after bL
        if (LB(st(bL)) < LB(st(a))) then
            LB(st(a)) <- LB(st(bL))
   }
```

Figure B.14: Filtering Rule for the Negation Alternate Precedence Template

Proof. The *NegationAlternatePrecedence* rule can be fired, at most, $n+2\times n\times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *NegationAlternatePrecedence* rule execution is $O(nt)$, and hence the worst-case time complexity of this rule is $O(n\times nt^2)$. □

B.18 Negation Alternate Response(A, B)

As stated, between two executions of *A*, *B* cannot be executed, $nt(A)\geq 2\Rightarrow \forall 1\leq i\leq nt(B) : et(B_i)\leq st(A_1)\vee st(B_i)\geq et(A_{nt(A)})$.

```
Negation Alternate Response (A,B) is
added OR bounds of nt(A) changed OR bounds of nt(B) changed OR
bounds of et(A1) OR bounds of st(Ant(A)) changed OR bounds of
st(Bi) for any i changed OR bounds of et(Bi) for any i changed ->
   if (LB(nt(A)) >= 2) then
    for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity b = Bi;
      SchedulingActivity aF = A1; // aFirst
      SchedulingActivity aL = AUB(nt(A)); // aLast
      if (UB(st(b)) < LB(st(aL))) then // b can not start after aL => b must finish before aF
        if (UB(et(aF)) < UB(et(b))) then
            UB(et(b)) <- UB(et(aF))
      if (LB(et(b)) > UB(et(aF))) then // b can not finish before aF => b must start after aL
        if (LB(st(aL)) < LB(st(b))) then
            LB(st(b)) <- LB(st(aL))
   }
```

Figure B.15: Filtering Rule for the Negation Alternate Response Template

Proposition 22. *If implemented properly, the worst-case time complexity of the rule NegationAlternateResponse, which includes all possible recursive calls, is*

$O(n \times nt^2)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.

Proof. The *NegationAlternateResponse* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover, the time complexity of the *NegationAlternateResponse* rule execution is $O(nt)$, and hence the worst-case time complexity of this rule is $O(n \times nt^2)$. □

B.19 Negation Alternate Succession(A, B)

As stated, relations $NegationAlternatePrecedence(A,B)$ and $NegationAlternate Response(A,B)$ hold.

Proposition 23. *If implemented properly, the worst-case time complexity of the rule NegationAlternateSuccession, which includes all possible recursive calls, is $O(n \times nt^2)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *NegationAlternateSuccession* rule can be implemented as the conjunction of *NegationAlternatePrecendence* and *NegationAlternateResponse* rules. The complexity of both rules is $O(n \times nt^2)$ (see Prop. 21 and 22), and hence the worst-case time complexity of *NegationAlternateSuccession* rule is $O(2 \times n \times nt^2)$, equal to $O(n \times nt^2)$. □

B.20 Negation Chain Succession(A, B)

As stated, *B* cannot be executed immediately after the execution of *A*, $\forall i : 1 \leq i \leq nt(B) : \neg\exists j : 1 \leq j \leq nt(A) : et(A_j) = st(B_i)$.

Proposition 24. *If implemented properly, the worst-case time complexity of the rule NegationChainSuccession, which includes all possible recursive calls, is $O(n \times nt^3)$, where n is the number of Repeated Activities of the problem, and nt is the upper bound of the variable $nt(Act)$ domain. This upper bound obtains the same value for all the ConDec-R activities.*

Proof. The *NegationChainSuccession* rule can be fired, at most, $n + 2 \times n \times nt$ times. This is due to the fact that a change in any execution of any activity (*st* and *et* variables of Act_i) or in the *nt* variable of an activity can fire this rule. Moreover,

```
Negation Chain Succession (A,B) is
added OR bounds of nt(A) changed OR bounds of nt(B) changed OR
et(Ai) for any i is bound OR st(Bi) for any i is bound ->
   for (int i = 1; i <= UB(nt(A)); i++){
      SchedulingActivity a = Ai;
      if(IsBound(et(a))){
        int aVal = Value(et(a));
        for (int j = 1; j <= UB(nt(B)); j++){
           SchedulingActivity b = Bj;
           if(MemberOf(aVal,st(b)))
            RemoveValue(aVal,st(b));
        }
      }
   }
   for (int i = 1; i <= UB(nt(B)); i++){
      SchedulingActivity b = Bi;
      if(IsBound(st(b))){
        int bVal = Value(st(b));
        for (int j = 1; j <= UB(nt(A)); j++){
           SchedulingActivity a = Aj;
           if(MemberOf(bVal,et(a)))
            RemoveValue(bVal,et(a));
        }
      }
   }
```

Figure B.16: Filtering Rule for the Negation Chain Succession Template

the time complexity of the *NegationChainSuccession* rule execution is $O(nt^2)$, and hence the worst-case time complexity of this rule is $O(n \times nt^3)$. □

Appendix C

Algorithms for Generating BPMN Models

In order to develop the algorithm to generate the BP models from the optimized enactment plans (cf. Alg. 6), certain related types are stated, as shown in Fig. C.1 (UML diagram). It should be clarified that, at this point of the process, the CSP variables are instantiated, hence all the information is known (*nt* variable for each BP activity, *st* variable for each scheduling activity, resource in which each scheduling activity is executed, etc). The types which appear in the UML diagram are as follows:

- OptimizedPlan: This represents the generated optimized enactment plan. Moreover, it contains the information related to the input problem. This type contains properties regarding a set of roles, a set of repeated activities (ConDec-R activities), and a set of templates which relate the repeated activities.

- RepeatedAct: This represents the ConDec-R activities. Each repeated activity contains information about the required role, the estimated duration, the set of scheduling activities which represent the execution of each BP activity, and the number of times this repeated activity is executed (property nt).

- Role: This represents a role, and it is composed of the set of resources available for this role.

- Resource: This represents a resource. This type contains properties regarding a list of scheduling activities which are executed in that resource, ordered by the start time.

- Template: This represents the high-level relations which are given between the repeated activities. In order to consider the branched constraints (Sect. 3.2), two specializations are included to allow the relations between one source and several sinks (TemplateSinks), and between several sources and one sink (TemplateSources). The method *includePred* of a template updates the information of the BPMN model by including the precedence relations which are implied by that template (more details are given later in this section during the presentation of the algorithms). For the generation of the BPMN model, the template relations are considered for the connection of the BPMN activities.

- P&SAct: This represents each execution of a repeated activity. This type contains properties regarding the start and the end times of the activity, together with the resource used by the scheduling activity. Since each P&SAct is related to a specific BPMNAct, the P&SAct type provides the method *toBPMNAct* in order to obtain the related BPMNAct from a P&SAct.

- BPMNModel: This represents the BPMN model that is generated. This model is composed of a set of BPMN activities, a set of pools, a set of gates, and a set of connections.

- BPMNAct: This represents a BPMN activity. This type contains properties regarding the pool and the lane where the activity is allocated, together with the estimated duration and start time.

- Pool: This represents a BPMN pool. Each pool is associated to a specific role and is composed of a set of lanes.

- Lane: This represents a BPMN lane. Each lane is associated to a specific resource.

- Gate: This represents a BPMN gate. In order to consider parallel merging gateways, a specialization, named ParallelM, is developed.

- ParallelM: This represents a parallel merging gateway. This type contains properties regarding a set of inputs (BPMN activities), and one output (BPMN activity).

- Connection: This represents a precedence connection between two BPMN activities, a and b.

The types which are presented in the UML diagram of Fig. C.1 are used for the development of the algorithms for the automatic generation of optimized BPMN models from enactment plans (cf. Algs. 6, 7, 8, 9 and 10). Generic types

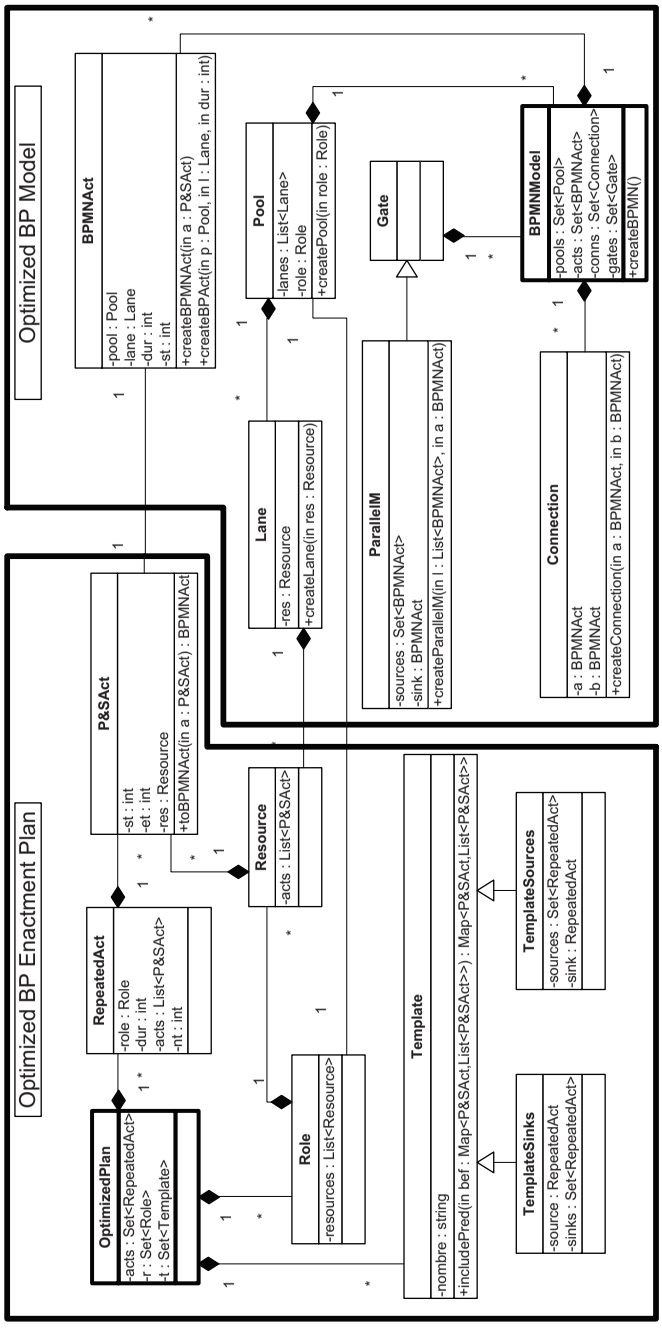


Figure C.1: UML Diagram of Types for the Optimized BPMN Generation.

are used, hence T<P> represents the generic type T, with the generic parameter instantiated to P. These algorithms are explained below.

The main algorithm, Alg. 6, constructs a BPMN model from the optimized BP enactment plan. This plan is represented by a sorted set of scheduling activ-

Algorithm 6: Construct an Optimized BP Model from an Optimized BP Enactment Plan

input : SortedSet <P&SAct> *acts*, ordered by *st*
Set<Template> *t*
Set<Role> *r*
output: BPMNModel *bp*

```
1  Map<P&SAct,Set<P&SAct >> pred ← CreateDependencies(acts,t,r);
2  bp.gates ← ∅;
3  bp.pools ← {createPool(role) | ∀role ∈ r};
4  bp.acts ← {createBPMNAct(a) | ∀a ∈ acts};
5  BPMNAct start ← createBPMNAct(P0,L0,0);
6  bp.conns ← {createConnection(start,ini) | ∀ini ∈ bp.acts, ini.st = 0};
7  foreach psact in acts do
8      if pred(psact).size == 1 then
9          P&SAct aPred ← pred(psact).get(0);
10         bp.conns ← bp.conns ∪
           createConnection(toBPMNAct(aPred),toBPMNAct(psact));
11     else
12         Set<BPMNAct> inputs ← {toBPMNAct(a) | ∀a ∈ pred(psact)};
13         bp.gates ←
           bp.gates ∪ createParallelM(inputs,toBPMNAct(psact));
14 BPMNAct end ← createBPMNAct(P0,L0,0);
15 Set<BPMNAct>
   finals ← {toBPMNAct(a) | ∀a ∈ P&SAct, ¬∃b ∈ P&SAct, a ∈ pred(b)};
16 if inputs.size == 1 then
17     BPMNAct final ← finals.get(0);
18     bp.conns ← bp.conns ∪ createConnection(final,end);
19 else
20     bp.gates ← bp.gates ∪ createParallelM(finals,end);
21 return bp;
```

ities ordered by start time, *acts*; a set of the templates which relate the repeated activities, *t*; and a set of the considered roles, *r*. The map *pred* associates a set of direct predecessors (cf. Def. 29 on page 86) for each scheduling activity (cf. Alg. 7, explained later in this section), in order to generate the BPMN model (line 1). Moreover, a pool associated to each role is created, together with the corresponding lanes (line 3). In a similar way, a BPMN activity associated to each scheduling activity is created (line 4). The start and end activities of the model

can be associated to any pool (P_0 in Alg. 6) and to any lane (L_0 in Alg. 6) (line 5 and 14 respectively). In line 6, a connection between the start BPMN activity and each BPMN activity whose estimated start time is equal to 0, is created. Lines 7-13 establish the connections and gateways between the BPMN activities in the following way: if the BPMN activity has only one direct predecessor, a connection is included; if the BPMN activity has several direct predecessors, a parallel merging gateway is included. In line 15, all the final activities are selected to be direct predecessors of the end activity. These activities are related by either a connection, in the case that there is only one ending activity (lines 16-18); or by a parallel merging gateway, in the case that there are several ending activities (lines 19-20).

Algorithm 7: CreateDependencies

input : SortedSet<P&SAct> *acts* ordered by *st*
Set<Template> *temp*
Set<Role> *roles*
output: Map<P&SAct,Set<P&SAct >> *pred*

```
1  foreach r in roles do
2      foreach res in r.resources do
3          List<P&SAct> actsRes ← res.acts;
4          foreach i in i:1..actsRes.size-1 do
5              pred(actsRes_{i+1}) ← actsRes_i;
6  foreach t in temp do
7      t.includePred(pred);
8  Map<P&SAct,Set<P&SAct >> indirectPred ← ∅;
9  foreach act in acts do
10     foreach p in pred(act) do
11         pred(act) ← pred(act) \ (pred(p) ∪ indirectPred(p));
12         indirectPred(act) ←
           indirectPred(act) ∪ (pred(p) ∪ indirectPred(p));
13 return pred;
```

Additionally, Algorithm 7 generates a map in which each scheduling activity is a associated to a set of scheduling activities that are its direct predecessors (cf. Def. 29 on page 86). First, the precedences required due to the use of the same resource are included (lines 1-5). Secondly, the precedences required due to the high-level relations between the repeated activities which are stated in the templates, are included (lines 6-7). The method *includePred* for some representative

templates is detailed in Algs. 8, 9 and 10. Lastly, the indirect predecessors (cf. Def. 30 on page 86) are removed from the map in order to avoid redundant connections, by taking into account that the sorted set *acts* is ordered by *st* (lines 8-12).

Algorithm 8: *includePred* method for the branched *Precedence* template

input : Map<P&SAct,Set<P&SAct >> *pred*
Set<RepeatedAct> *sources*
RepeatedAct *sink*
output: Map<P&SAct,Set<P&SAct >> *pred*

1 Set<P&SAct> $meet \leftarrow \{a_1 \mid \forall a \in sources, a_1.et \leq sink_1.st\}$;
2 P&SAct $sel \leftarrow \underset{a \in meet}{\text{argmin}}(a.et)$;
3 $pred(sink_1) \leftarrow pred(sink_1) \cup sel$;
4 **return** *pred*;

With respect to the *includePred* method, some representative templates are selected for illustration purposes (other templates can be described in a similar way). In Alg. 8, the template regarding the branched *Precedence* template, is shown. The location of a branched precedence template between several sources and one sink implies that the first execution of at least one of the sources must finished before the start of the first execution of the sink. In line 1, the set of scheduling activities which comply with the Precedence template (i.e, the first executions of the sources which end before the start of the first execution of the sink) are included in the set *meet*. At least one scheduling activity will be included in this set since the Precedence template is satisfied, however it may be possible to find more than one. In order to generate a BPMN model which is compatible with both the optimized enactment plan and the ConDec-R specification, as is the purpose of our approach, any scheduling activity of the set *meet* can be selected to be the predecessor of the sink in the BPMN model. One scheduling activity of the set *meet* is then selected to be the predecessor of the sink. Specifically, the scheduling activity which presents more slack is selected (line 2) in order to construct a robust BPMN model. In line 3, the selected predecessor is included in the map, and is associated to the predecessors of the first execution of the sink. The fact that an activity *B* can start after another activity *A* has finished (ES, default option), is stated by including *A* in the set *pred* of *B* (line 3) of Alg. 9.

The branched *AlternatePrecedence* template between several sources and one sink implies that before the execution of the sink, at least one of the sources must be executed, and between each two executions of the sink, at least one of the sources must be executed. As discussed, there exist two variants for the same

Algorithm 9: *includePred* method for the branched *AlternatePrecedence* Template

input : Map<P&SAct,Set<P&SAct >> *pred*
Set<RepeatedAct> *sources*
RepeatedAct *sink*
output: Map<P&SAct,Set<P&SAct >> *pred*

1 Set<P&SAct> $meet \leftarrow \{a_1 \mid \forall a \in sources, a_1.et \leq sink_1.st\}$;
2 P&SAct $sel \leftarrow \underset{a \in meet}{\text{argmin}}(a.et)$;
3 $pred(sink_1) \leftarrow pred(sink_1) \cup sel$;
4 **foreach** *i in 2..sink.nt* **do**
5 Set<P&SAct> $meet \leftarrow \{a_j \mid \forall a \in sources, \forall j \in 1..a.nt, sink_{i-1}.et \leq a_j.st \wedge a_j.et \leq sink_i.st\}$;
6 P&SAct $sel \leftarrow \underset{a \in meet}{\text{argmax}}((a.st - sink_{i-1}.et) + (sink_i.st - a.et))$;
7 $pred(sel) \leftarrow pred(sel) \cup sink_{i-1}$;
8 $pred(sink_i) \leftarrow pred(sink_i) \cup sel$;
9 **return** *pred*;

temporal relation, which are represented by adding SS or ES at the end of the name of the template. In the *AlternatePrecedence* template, two temporal relations must be indicated: first, what "sink before source" means, and secondly, what "source before sink" means. Therefore, the branched template *AlternatePrecedenceES-ES* (default option) specifies that "sink before source" means that the **end** time of the sink must be less than or equal to the **start** time of the source, and "source before sink" means that the **end** time of the source must be less than or equal to the **start** time of the sink. The *includePred* method for the branched template *AlternatePrecedenceES-ES* is shown in Alg. 9. For lines 1-3, the idea is the same as that in Alg. 8. Moreover, between each two successive executions of the sink, $sink_{i-1}$ and $sink_i$, one scheduling activity must be executed. Several scheduling activities related to the sources can meet this condition (line 5). As before, the scheduling activity which presents more slack is selected (line 6) to be the predecessor of $sink_i$ (line 8), and at the same time $sink_{i-1}$ is selected as the predecessor of the selected scheduling activity.

In a similar way, the branched template *AlternatePrecedenceSS-ES* specifies that "sink before source" means that the **start** time of the sink must be less than or equal to the **start** time of the source, and "source before sink" means that the **end** time of the source must be less than or equal to the **start** time of the sink. The *includePred* method for the branched template *AlternatePrecedenceSS- ES* is shown in Alg. 10. This algorithm is identical to Alg. 9, except for line 7.

Algorithm 10: *includePred* method for the branched *AlternatePrecedenceSS-ES* Template

input : Map<P&SAct,Set<P&SAct >> *pred*
Set<RepeatedAct> *sources*
RepeatedAct *sink*
output: Map<P&SAct,Set<P&SAct >> *pred*

1 Set<P&SAct> $meet \leftarrow \{a_1 \mid \forall a \in sources, a_1.et \leq sink_1.st\}$;
2 P&SAct $sel \leftarrow \underset{a \in meet}{\text{argmin}}(a.et)$;
3 $pred(sink_1) \leftarrow pred(sink_1) \cup sel$;
4 **foreach** *i in 2..sink.nt* **do**
5 Set<P&SAct> $meet \leftarrow \{a_j \mid \forall a \in sources, \forall j \in 1..a.nt, sink_{i-1}.\mathbf{st} \leq a_j.st \wedge a_j.et \leq sink_i.st\}$;
6 P&SAct $sel \leftarrow \underset{a \in meet}{\text{argmax}}((a.st - sink_{i-1}.et) + (sink_i.st - a.et))$;
7 $pred(sel) \leftarrow pred(sel) \cup \mathbf{pred}(sink_{i-1})$;
8 $pred(sink_i) \leftarrow pred(sink_i) \cup sel$;
9 **return** *pred*;

As mentioned earlier, the fact that an activity B can start after another activity A has finished (ES, default option), is stated by including A in the set *pred* of B. Additionally, the fact that an activity B can only start after another activity A has **started**, label SS, is stated by including the set $pred(A)$ in the set *pred* of B, as can be seen in line 7 of Alg. 10.

The complexity analysis of all the algorithms previously described is included in C.1.

C.1 Complexity Analysis

This section presents the complexity analysis of the algorithms previously described.

Proposition 25. *If implemented properly, the worst-case time complexity of Algorithm 8 is $O(n)$, where n is the number of Repeated Activities of the problem.*

Proof. The worst-case time complexity of line 1 is $O(n)$, since $\#sources \leq n$. The worst-case time complexity of line 2 is also $O(n)$, since $\#meet \leq n$. The time complexity of line 3 is constant. Therefore, the worst-case time complexity of Algorithm 8 is $O(n) + O(n) + \Theta(1)$, equal to $O(n)$. □

Proposition 26. *If implemented properly, the worst-case time complexity of Algorithms 9 and 10 is $O(n \times nt)$, where n is the number of Repeated Activities of the problem, and nt is the maximum number of times that a repeated activity is executed.*

Proof. The worst-case time complexity of line 1 is $O(n)$, since $\#sources \leq n$. The worst-case time complexity of line 2 is also $O(n)$, since $\#meet \leq n$. The time complexity of line 3 is constant. The worst-case time complexity of lines 4-8 is $O(n \times nt)$ since lines 5-7 (with complexity $O(n)$ from the proof of Proposition 25) are executed at most *nt* times. Therefore, the worst-case time complexity of Algorithms 9 and 10 is $O(n) + O(n) + \Theta(1) + O(n \times nt)$ equal to $O(n \times nt)$. □

Proposition 27. *If implemented properly, the worst-case time complexity of Algorithm 7 is $O(t \times n \times nt + nps^2)$, where n is the number of Repeated Activities of the problem, nt is the maximum number of times that a repeated activity is executed, t is the number of templates that appear in the definition of the problem, and nps is the number of scheduling activities in the optimized plan.*

Proof. The time complexity of line 1-5 is $\Theta(nps)$, since each scheduling activity is considered exactly once (each activity uses a specific resource of a specific role). The worst-case time complexity of lines 6-7 is $O(t \times n \times nt)$, since the worst-case time complexity of the method *includePred* is $O(n \times nt)$ (Proposition 26), and this method is invoked t times. The worst-case time complexity of lines 9-12 is $O(nps^2)$, since for each scheduling activity, its predecessors (at most, nps) are considered. Therefore, the worst-case time complexity of Algorithm 7 is $O(t \times n \times nt + nps^2)$. □

Proposition 28. *If implemented properly, the worst-case time complexity of Algorithm 6 is $O(t \times n \times nt + nps^2)$, where n is the number of Repeated Activities of the problem, nt is the maximum number of times that a repeated activity is executed, t is the number of templates that appear in the definition of the problem, and nps is the number of scheduling activities in the optimized plan.*

Proof. The worst-case time complexity of line 1 is $O(t \times n \times nt + nps^2)$, by Proposition 27. The worst-case time complexity of line 3 is $O(n)$, since $\#role \leq n$. The time complexity of line 4 is $\Theta(nps)$. The worst-case time complexity of lines 6 and 15 is $O(nps)$. The time complexity of lines 7-13 is $\Theta(nps)$. Therefore, the worst-case time complexity of Algorithm 6 is $O(t \times n \times nt + nps^2)$. □

Appendix D

AI Techniques for Solving the Multi-mode Repair Planning Problem

AI-based techniques for solving the multi-mode repair planning problem are presented, specifically: (1) a constraint-based approach (cf. Appendix D.1), and (2) a PDDL specification (cf. Appendix D.2).

D.1 Constraint-based Approach

In this section, the constraint-based approach for solving the multi-mode repair planning problem is presented (cf. (Barba et al., 2009a)). According to the considered problems (cf. Sect. 6.3), the time and resource constraints, typical from scheduling, would be modified to conditional constraints taking into account that tasks (and subsystems) may not appear in the solution. Most of the ideas are taken from (Del Valle et al., 2010), but the assumptions considered in this work will result in modifying most constraints and in adding others: apart from the optimization of the duration, the minimization of the total cost is pursued, resulting in a multi-objective optimization problem, and multi-mode tasks are considered (cf. (Barba et al., 2009a)).

D.1.1 Variables of the CSP

Four kinds of CSP variables have been defined: selection, resource, time and cost variables.

Selection variables. For each And node, two boolean variables represent if the connection and disconnection tasks are selected for the solution, $s(T)$ and $s(T')$

respectively. Furthermore, for each Or node, two boolean variables represent if the subsystem S appears in the connection and disconnection processes, $s(S)$ and $s'(S)$ respectively.

Resource variables. For each And node, $M(T)$ and $M(T')$ show the resources used, and $Cf(T)$ and $Cf(T')$ are the necessary configuration on them for the connection and disconnection tasks respectively. These values are data of the problem. On the other hand, the resource where a subsystem is obtained after the corresponding disconnection and connection task, are represented by the variables $m'(S)$ and $m(S)$ respectively, that are variables of the CSP.

Time variables. For each And node, the durations of the associated tasks $Dur(T)$ and $Dur(T')$ are established. Due to the auxiliary operations, $\Delta_{cht}(M, Cf, Cf')$ denotes the time needed for changing the configuration of the resource M from Cf to Cf', and $\Delta_{mov}(S, M, M')$ denotes the time needed for transporting the subsystem S from resource M to resource M'. Finally, a component C to be repaired is associated to a temporal delay $\Delta_{subst}(C)$, corresponding to the reparation or substitution of the faulty component. These values are data of the problem.

On the other hand, for each And node, the CSP variables related to the time are: its starting times, $t_i(T)$ and $t_i(T')$ and ending times, $t_f(T)$ and $t_f(T')$. For each Or node, the times when it is obtained after connection, $t_{OR}(S)$, and disconnection, $t'_{OR}(S)$.

Cost variables For each And node, it is considered: its connection and disconnection cost, $Cost(T_i)$ and $Cost(T'_i)$ respectively. Regarding to the auxiliary operations, $Cost_{cht}(M, Cf, Cf')$ denotes the cost of changing the configuration of the resource M from Cf to Cf', and $Cost_{mov}(S, M, M')$ denotes the cost of transporting the subsystem S from resource M to resource M'. Furthermore, a component C to be repaired is associated to a cost $Cost_{subst}(C)$, corresponding to the reparation or substitution of the faulty component.

On the other hand, for each And node, the selection of the corresponding task T may be associated some additional costs, as explained in Sect. D.1.2: first, the variable $cost_{mov}(T_i)$ represents the possible costs which are associated to the movement of subsystems to $m(T)$, if this resource is different from the one where the subsystem were previously obtained; and secondly, the variable $cost_{cht}(T_i)$ represents the possible costs of change of configuration, if $m(T)$ has been previously used with a different one. These variables are linked to the And nodes because the costs are due to the selection of the corresponding task.

Finally, a variable that represents the total cost of a plan, $cost_{total}$, has been used in order to minimize this objective function. Therefore, a multi-objective optimization is pursued, encompassing both objective functions, time and cost.

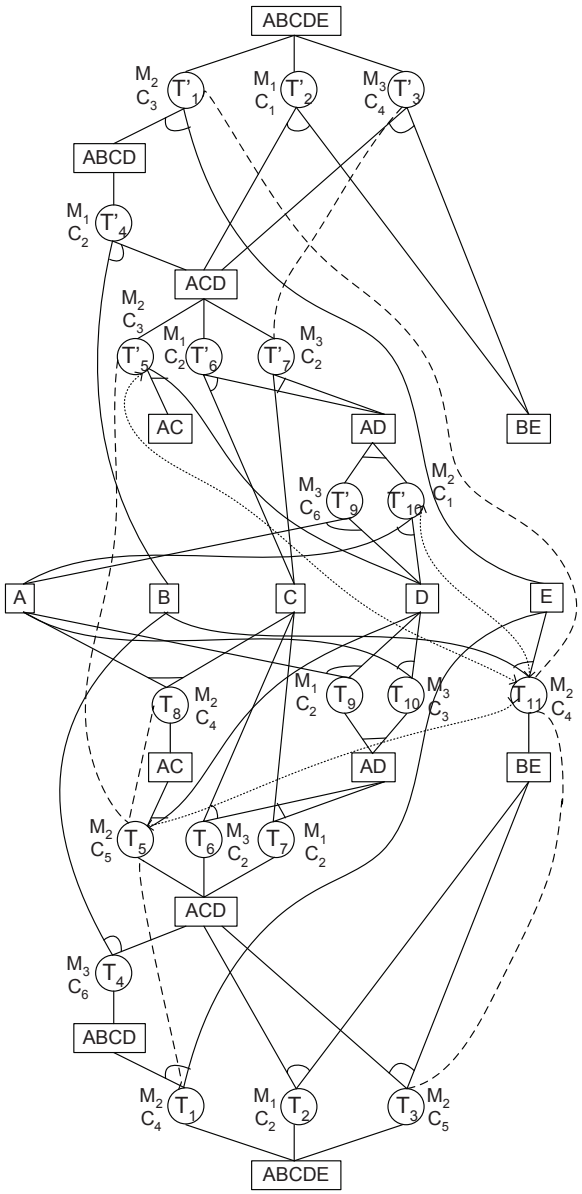


Figure D.1: The extended simplified repair And/Or graph with relations (5) and (6) between tasks

D.1.2 Constraints of the CSP

Taking the variables of the proposed CSP model into account (cf. Sect. D.1.1), a classification about the types of constraints can be done: selection, resource, time and cost constraints. These constraints are detailed as follows, and some examples related to the graph of Fig. D.1 are given.

Selection Constraints collect the relations between the boolean variables that represent if the tasks are selected for the solution and if the subsystems appears in the repair process. A special case is for the complete system and for the faulty component, which will be always part of the solution, so $s'(ABCDE) = s(ABCDE) = s'(D) = s(D) = true$.

Related to relations of type (1) (cf. Sect. 6.3.1), constraints which relate the selection of disconnection tasks T' and connection tasks T with the selection of subsystems are included: $s'(S) \Leftrightarrow XOR_{T'_i \in succ(S)}(s(T'_i))$ and $s(S) \Leftrightarrow XOR_{T_i \in succ(S)}(s(T_i))$ (example $s'(ABCDE) \Leftrightarrow (XOR(s(T'_1), s(T'_2), s(T'_3)))$).

Related to relations of type (3) (cf. Sect. 6.3.1), the obligatory selection of the two Or nodes if the And node is selected is considered: $s(T') \Rightarrow s'(S_1) \wedge s'(S_2)$ and $s(T) \Rightarrow s(S_1) \wedge s(S_2)$ (example $s(T'_1) \Rightarrow s'(ABCD) \wedge s'(E)$).

Related to relations of type (4) (cf. Sect. 6.3.1), constraints which relate the selection of disconnection tasks T' and connection tasks T with the selection of subsystems are included: $s'(S) \Leftrightarrow XOR_{T'_i \in pred(S)}(s(T'_i))$ and $s(S) \Leftrightarrow XOR_{T_i \in pred(S)}(s(T_i))$ (example $s(ACD) \Leftrightarrow XOR(s(T_2), s(T_3), s(T_4))$).

Resource Constraints consider the relations between the resources used in the connection and disconnection tasks, and the resources where the subsystems are obtained after them.

Related to relations of type (1) (cf. Sect. 6.3.1), the resource m where a subsystem is generated after a connection task is the resource used by this task: $s(T_i) \Rightarrow m(S) = M(T_i)$ (example $s(T_{10}) \Rightarrow m(AD) = M(T_{10})$).

Related to relations of type (3) (cf. Sect. 6.3.1), the resource m' where a subsystem is generated after a disconnection task is the resource used by this task: $s(T'_i) \Rightarrow m'(S_1) = m'(S_2) = M(T'_i)$ (example $s(T'_9) \Rightarrow m'(A) = m'(D) = M(T'_9)$).

Time Constraints collect the relations between the start and the end times of the tasks, and the time when the subsystems are obtained. Initially, $t'_{OR}(ABCDE) = 0$.

Related to relations of type (1) (cf. Sect. 6.3.1), these constraints establish the disconnection times t'_{OR} and connection times t_{OR} of Or nodes related to the start times of the disconnection tasks or the end times of the connection tasks: $s(T'_i) \Rightarrow t_i(T'_i) \geq t'_{OR}(S) + \Delta_{mov}(S, m'(S), M(T'_i))$ and $s(T_i) \Rightarrow t_f(T_i) = t_{OR}(S)$ (example $s(T'_1) \Rightarrow t_i(T'_1) \geq t'_{OR}(ABCDE) + \Delta_{mov}(ABCDE, m'(ABCDE), M(T'_1))$).

Related to relations of type (2) (cf. Sect. 6.3.1), these constraints consider the end time of the tasks related to the start time and its durations: $s(T'_i) \Rightarrow t_f(T'_i) = t_i(T'_i) + Dur(T'_i)$ and $s(T_i) \Rightarrow t_f(T_i) = t_i(T_i) + Dur(T_i)$ (example $s(T'_1) \Rightarrow t_f(T'_1) = t_i(T'_1) + Dur(T'_1)$).

Related to relations (3) (cf. Sect. 6.3.1), constraints which relate the equality constraints between the disconnection times of the Or nodes t'_{OR} and the end time of a disconnection task T' which is placed above them in the original And/Or

graph, together with the precedence between the connection time of the Or nodes t_{OR} and the start times of the connection tasks T (And nodes) are included: $s(T_i') \Rightarrow t_f(T_i') = t'_{OR}(S_1) = t'_{OR}(S_2)$.

Furthermore, the possible delays due to the transportation of subsystems between different resources are considered: $s(T_i) \Rightarrow t_i(T_i) \geq t_{OR}(S_1) + \Delta_{mov}(S_1, m(S_1), M(T_i))$ and $s(T_i) \Rightarrow t_i(T_i) \geq t_{OR}(S_2) + \Delta_{mov}(S_2, m(S_2), M(T_i))$ (example $s(T'_{10}) \Rightarrow t_f(T'_{10}) = t'_{OR}(A) = t'_{OR}(D)$).

Related to relations of type (5) (cf. Sect. 6.3.1), constraints which relate a task T_i and its closest predecessor task T_j using the same resource m, taking into account the possible change of configuration are considered: $(s(T_i) \wedge s(T_j)) \Rightarrow t_i(T_j) \geq t_f(T_i) + \Delta_{cht}(m, Cf(T_i), Cf(T_j))$ (example $(s(T'_1) \wedge s(T'_{10})) \Rightarrow t_i(T'_{10}) \geq t_f(T'_1) + \Delta_{cht}(M2, Cf(T'_1), Cf(T'_{10}))$).

Moreover, since the solution may contain non-reverse tasks, each disconnection task must be related to each closest successor connection task that uses the same resource. Furthermore, when both tasks use the same configuration, the resulting constraint is superfluous and can be eliminated.

For each two tasks T_i and T_j requiring the same resource m, with no precedence constraint among them and which may belong to the same repair plan, the constraints of type (6) (cf. Sect. 6.3.1) express the two possible orders of execution of the tasks: $(s(T_i) \wedge s(T_j)) \Rightarrow (t_i(T_i) \geq t_f(T_j) + \Delta_{cht}(m, Cf(T_j), Cf(T_i)) \vee\ t_i(T_j) \geq t_f(T_i) + \Delta_{cht}(m, Cf(T_i), Cf(T_j)))$ (example $(s(T_8) \wedge s(T_{11})) \Rightarrow t_i(T_8) \geq t_f(T_{11}) + \Delta_{cht}(M2, Cf(T_{11}), Cf(T_8)) \vee t_i(T_{11}) \geq t_f(T_8) + \Delta_{cht}\ (M2, Cf(T_8), Cf(T_{11}))$).

For the *Or* leaf nodes (including those that do not include the faulty component) t'_{OR} and t_{OR} are equals, except for the faulty component, in which the delay corresponding to the reparation is considered.

Cost Constraints: In the repair process of a component in a complete system, the cost of a plan can be established by the aggregated costs associated to the execution of the selected tasks. The total cost of selecting a task T_i involves:

- the execution cost of the task, $Cost(T_i)$ (related to relation (2))
- the cost associated to the possible movement of one or two subsystems from one resource to another, $cost_{mov}(T_i)$:
  - in disconnection tasks T_i', it is necessary to take into account the possible movement of the subsystem related to the Or nodes above it in the original And/Or graph, related to relation (1), $cost_{mov}(T_i') = Cost_{mov}(S, m'(S), M(T_i'))$
  - in connection tasks T_i, it is necessary to take into account the possible movement of the two subsystems related to Or nodes below it in the original And/Or graph, related to relation (3), $cost_{mov}(T_i) = Cost_{mov}(S_1, m(S_1), M(T_i)) + Cost_{mov}(S_2, m(S_2), M(T_i))$

Table D.1: Cost Constraints

Type	Constraint
(1)	$s(T_1') \Rightarrow cost_{mov}(T_1') = Cost_{mov}(ABCDE, m'(ABCDE), M(T_1'))$
(1)	...
(1)	$s(T_{10}') \Rightarrow cost_{mov}(T_{10}') = Cost_{mov}(AD, m'(AD), M(T_{10}'))$
(3)	$s(T_1) \Rightarrow cost_{mov}(T_1) = Cost_{mov}(ABCD, m(ABCD), M(T_1)) + Cost_{mov}(E, m(E), M(T_1))$
(3)	...
(3)	$s(T_{11}) \Rightarrow cost_{mov}(T_{11}) = Cost_{mov}(B, m(B), M(T_{11})) + Cost_{mov}(E, m(E), M(T_{11}))$
(5),(6)	$s(T_{10}') \Rightarrow cost_{cht}(T_{10}') = Cost_{cht}(M(T_{10}'), Cf(argmax_{Ta \in \{T_1', T_{11}\}}\{T_f(T_a) \mid s(T_a) \wedge t_f(T_a) \leq t_i(T_{10}')\}), Cf(T_{10}'))$
(5),(6)	...
(5),(6)	$s(T_8) \Rightarrow cost_{cht}(T_8) = Cost_{cht}(M(T_8), Cf(argmax_{Ta \in \{T_{10}', T_{11}\}}\{T_f(T_a) \mid s(T_a) \wedge t_f(T_a) \leq t_i(T_8)\}), Cf(T_8))$

- the possible cost associated to a change of configuration on $M(T_i)$, $cost_{cht}(T_i)$. If $M(T_i)$ has been used before by another task with a different configuration, it is necessary to change it. An additional complexity of the considered problems is that the cost of the change of configuration depends on the sequence of tasks which are executed using each resource, and hence the **precedent task** executed on $m(T_i)$ should be considered, being necessary to analyze two groups of tasks: set of possible immediate predecessors of T_i using the same resource (precedent tasks with Relation (5)); and set of tasks linked to T_i by the relation (6), which is explained in Sect. D.1.2.

  Taking this into account, $cost_{cht}(T_i) = Cost_{cht}(M(T_i), Cf(PM(T_i)), Cf(T_i))$, where $PM(T_i)$ is the precedent task executed on $m(T_i)$, needs to be met. If T_i is executed the first one on its resource, then $cost_{cht}(T_i) = 0$.

On the other hand, the total cost of a plan can be defined as: $cost_{total} = \sum_{T_i} s(T_i)(Cost(T_i) + cost_{mov}(T_i) + cost_{cht}(T_i))$.

In Table D.1, some cost constraints of the And/Or graph of the Fig. D.1 are shown.

Table D.2: Predicates for the repair planning problem

Predicate	Description
(is-built-connection ?s - subsystem)	The subsystem has been obtained in the connection process and it has not been used yet for another operation.
(is-built-disconnection ?s - subsystem)	The subsystem has been obtained in the disconnection process and it has not been used yet for another operation.
(at ?s - subsystem ?l - location)	The subsystem *s* is in location *l*.
(has-config ?r - resource ?c - configuration)	The resource *r* has *c* configuration.
(task-connection ?s1 ?s2 ?s - subsystem ?r - resource ?c - configuration)	There exists a connection task that is executed in resource *r* with *c* configuration and it connects *s1* and *s2* to obtain *s*.
(task-disconnection ?s ?s1 ?s2 - subsystem ?r - resource ?c - configuration)	There exists a disconnection task that is executed in *r* with *c* configuration and it disconnects *s* to obtain *s1* and *s2*.
(fault ?s - subsystem)	It is the faulty component.
(leaf ?s - subsystem)	It is a leaf in the And/Or graph.
(free ?r - resource)	The resource is free.

Notice that the combinatorial character of the problem is due to the XOR constraints of types (1) and (4) and the disjunctive constraints of type (6). These types of constraints correspond, respectively, to the selection of alternative tasks and to the use of shared resources by them that are not related through precedence constraints.

D.2 PDDL Specification

In this section, a PDDL 2.2 specification for solving the multi-mode repair planning problem is proposed (cf. (Barba et al., 2009b)). As stated before, PDDL specifications include two separated files: a domain file for predicates and actions; and a problem file for objects, initial state and goal specification.

The definition of the domain for a PDDL specification contains different items (cf. Sect. 2.2.2):

Predicates: Several predicates has been considered in the current problem (Table D.2). The aim of defining separated predicates *is-built-connection* and *is-built-disconnection* is to facilitate the search working of the planner.

Table D.3: Functions for the repair planning problem

Function	Description
(cht ?r - resource ?c1 ?c2 - configuration)	Required time for change *r* from *c1* to *c2*.
(cost-cht ?r - resource ?c1 ?c2 - configuration)	Required cost for change the resource *r* from *c1* to *c2*.
(mov ?s - subsystem ?l1 ?l2 - location)	Required time for moving subsystem *s* from location *l1* to *l2*.
(cost-mov ?s - subsystem ?l1 ?l2 - location)	Required cost for moving subsystem *s* from location *l1* to *l2*.
(length-connection ?s1 ?s2 ?s - subsystem ?r - resource ?conf - configuration)	Required time for connecting subsystems *s1* and *s2* to obtain *s*.
(cost-connection ?s1 ?s2 ?s - subsystem ?r - resource ?conf - configuration)	Required cost for connecting subsystems *s1* and *s2* to obtain *s*.
(length-disconnection ?s ?s1 ?s2 - subsystem ?r - resource ?conf - configuration)	Required time for disconnecting the subsystem *s* to obtain *s1* and *s2*.
(cost-disconnection ?s ?s1 ?s2 - subsystem ?r - resource ?conf - configuration)	Required cost for disconnecting the subsystem *s* to obtain *s1* and *s2*.
(repair ?s - subsystem)	Required time for repairing the subsystem *s*.
(cost-repair ?s - subsystem)	Required cost for repairing the subsystem *s*.
(accumulated-cost ?r - resource)	This fluent is the cost accumulated to each resource at each time. It is used to define the objective functions related to cost.

Functions (Fluents): They are used in actions preconditions or effects and their values are given in the problem file (Table D.3).

(Durative) Actions/Operators: The execution of a durative action has associated a duration. (Fig. D.2 and D.3). The *connection* action acts on subsystems *s1* and *s2* to obtain *s*, using the resource *r* with configuration *conf*, with duration given by the function *length-connection*. The cost of this action is added to the fluent *accumulated-cost* of *r*. Similarly, the *disconnection* action is defined (it does not appear in the figure). The repair action repairs the subsystem *s* on resource *r*, with duration given by the function *repair*. The cost of this action is added to the fluent *accumulated-cost* of *r*. In the disconnection process, when a subsystem that does not contain the faulty component is obtained, it is not separated anymore,

so it disappears of the disconnection process to join to the connection process (*disconnection-to-connection* action). This action does not correspond to any actual activity, but it is proposed to facilitate the search working of the planner. The *move* action moves the subsystem *s* from location *l1* to *l2*, with duration given by the function *mov*. The cost of this action is added to the fluent *accumulated-cost* of *r2*. Finally, the *change-configuration* action change the configuration of *r* from *c1* to *c2*. The cost of this action is added to the fluent *accumulated-cost* of *r*.

```
 (:durative-action CONNECTION
    :parameters (?s1 ?s2 ?s - subsystem ?r - resource ?c - configuration)
    :duration (= ?duration  (length-connection ?s1 ?s2 ?s ?r ?c))
    :condition
        (and  (at start (task-connection ?s1 ?s2 ?s ?r ?c))
            (at start (free ?r))
            (at start (has-config ?r ?c))
            (at start (at ?s1 ?r))
            (at start (at ?s2 ?r))
            (at start (is-built-connection ?s1))
            (at start (is-built-connection ?s2)))
    :effect
        (and  (at start (not (free ?r)))
            (at start (not (at ?s1 ?r)))
            (at start (not (at ?s2 ?r)))
            (at start (not (is-built-connection ?s1)))
            (at start (not (is-built-connection ?s2)))
            (at end (free ?r))
            (at end (at ?s ?r))
            (at end (is-built-connection ?s))
            (at end (increase (accumulated-cost ?r)(cost-connection ?s1 ?s2 ?s ?r ?c)))))
(:durative-action REPAIR
    :parameters (?comp - subsystem ?r - resource)
    :duration (= ?duration (repair ?comp))
    :condition
        (and    (at start (fault ?comp))
            (at start (is-built-disconnection ?comp))
            (at start (at ?comp ?r)))
    :effect
        (and (at start (not (is-built-disconnection ?comp)))
            (at end (is-built-connection ?comp))
            (at start (not (at ?comp ?r)))
            (at end (at ?comp ?r))
            (at start (not (fault ?comp)))
            (at end (increase (accumulated-cost ?r) (cost-repair ?comp)))))
 (:action DISCONNECTION-TO-CONNECTION
    :parameters (?s - subassembly)
    :precondition (and (leaf ?s)
                (is-built-disconnection ?s))
    :effect (and (not (is-built-disconnection ?s))
                (is-built-connection ?s))))
```

Figure D.2: PDDL specification for the *connection*, *repair* and *disconnection-to-connection* actions.

The PDDL 2.2 problem defines the next items (cf. Sect. 2.2.2):

```
 (:durative-action MOVE
    :parameters (?s - subsystem ?r1 - resource ?r2 - resource)
    :duration (= ?duration  (mov ?s ?r1 ?r2))
    :condition (at start (at ?s ?r1))
    :effect (and (at start (not (at ?s ?r1)))
                 (at end (at ?s ?r2))
                 (at end (increase (accumulated-cost ?r2) (cost-mov ?s ?r1 ?r2)))))
(:durative-action CHANGE-CONFIGURATION
    :parameters (?r - resource ?c1 ?c2 - configuration)
    :duration (= ?duration  (cht ?r ?c1 ?c2))
    :condition (and (at start (has-config ?r ?c1))
            (at start (free ?r)))
    :effect (and (at start (not (free ?r)))
            (at end (free ?r))
            (at start (not (has-config ?r ?c1)))
            (at end (has-config ?r ?c2))
            (at end (increase (accumulated-cost ?r) (cost-cht ?r ?c1 ?c2)))))
```

Figure D.3: PDDL specification for the *move* and *change-configuration* actions.

```
(define (problem grafo) (:domain repair)
(:objects M1 M2 - resource
      C0 C1 - configuration
      ABCDE ABCD ACD AC AD BE A B C D E - subsystem
      INIT_LOC - warehouse)
(:init
(at ABCDE INIT_LOC)(is-built-disconnection ABCDE) (fault D)
(free M1) (free M2) (has-config M1 C0) (has-config M2 C0) (leaf A) ...
(task-disconnection ABCDE ABCD E M1 C1)
(=(length-disconnection ABCDE ABCD E M1 C1) 10)
(= (cost-disconnection ABCDE ABCD E M1 C1) 124) ...
(= (cht M1 C0 C1) 0) (= (cost-cht M1 C0 C1) 0) (= (cht M1 C3 C1) 4)
(= (cost-cht M1 C1 C3) 36) ... (= (mov AD INIT_LOC M1) 0)
(= (cost-mov AD INIT_LOC M1) 0) (= (mov AD M1 M2) 1)
(= (cost-mov AD M1 M2) 15)...
(:goal (is-built-connection ABCDE))
(:metric minimize (+ (* (total-time) 10) (+ (accumulated-cost M1)
                        (accumulated-cost M2))))
```

Figure D.4: PDDL Problem specification of Fig. D.1.

Objects: For the problem of Fig. D.1, the objects can be seen in Fig. D.4. Initially, the system can be in anywhere, that it is represented by the location *INIT_LOC*, and the resources can have any configuration, that it is represented by *C0*.

Initial state: Some of them related to the problem of Fig. D.1 can be seen in Fig. D.4.

Goal specification: In the repair problem, the objective is to obtain the complete system in the connection process (Fig. D.4).

Objective function: In business process environments, several objective functions can be defined depending on the problem to solve. In the current proposal,

some objective functions have been selected to be minimized, one of them appears in Fig. D.4.

In this section, a PDDL 2.2 specification for solving a specific P&S problem, the multi-mode repair planning problem, has been explained. Similar P&S problems can be solved in a related way.

Bibliography

Aguilar-Savén, R., 2004. Business process modelling: Review and framework. International Journal of Production Economics 90 (2), 129 – 149.

Allen, J., 1983. Maintaining knowledge about temporal intervals. In: Proc. Communications of the ACM. pp. 832–843.

Alves, F. S. R., Guimares, K. F., Fernandes, M. A., 2008. Integrating planning and scheduling based on genetic algorithms to an workflow system. In: Proc. CEC. pp. 3766–3775.

Awad, A., Goré, R., Thomson, J., Weidlich, M., 2011. An iterative approach for business process template synthesis from compliance rules. In: Proc. Caise. pp. 406–421.

Bae, J., Bae, H., Kang, S., Kim, Y., 2004. Automatic control of workflow processes using eca rules. IEEE Transactions on Knowledge and Data Engineering 16 (8), 1010–1023.

Bao, F., Chintabathina, S., Morales, A., Rushton, N., Watson, R., Zhang, Y., 2011. A temporally expressive planner based on answer set programming with constraints: Preliminary design. In: Proc. LPNMR. pp. 398–414.

Baptiste, P., Le Pape, C., Nuijten, W., 1999. Satisfiability tests and time-bound adjustments for cumulative scheduling problems. Annals of Operations Research 92, 305 – 333.

Barba, I., Del Valle, C., 2010. Planning and Scheduling of Business Processes in Run-Time: A Repair Planning Example. In: Proc. ISD. Springer, pp. 75–88.

Barba, I., Del Valle, C., 2011a. A Constraint-based Approach for Planning and Scheduling Repeated Activities. In: Proc. COPLAS. pp. 55–62.

Barba, I., Del Valle, C., 2011b. A Planning and Scheduling Perspective for Designing Business Processes from Declarative Specifications. In: Proc. Icaart. Vol. 1. pp. 562–569.

Barba, I., Del Valle, C., Borrego, D., 2009a. A Constraint-based Model for Multi-objective Repair Planning. In: Proc. ETFA. pp. 234–241, art. no. 5347038.

Barba, I., Del Valle, C., Borrego, D., 2009b. PDDL Specification for Multi-objective Repair Planning. In: Proc. CAEPIA 2009 Workshop on Planning, Scheduling and Constraint Satisfaction. pp. 21–33.

Barba, I., Weber, B., Del Valle, C., 2011. Supporting the Optimized Execution of Business Processes through Recommendations. In: Proc. BPI. Springer LNCS (In press).

Barjis, J., Verbraeck, A., 2010. The relevance of modeling and simulation in enterprise and organizational study. In: Proc. EOMAS. pp. 15–26.

Barták, R., Cepek, O., 2008. Incremental filtering algorithms for precedence and dependency constraints. International Journal on Artificial Intelligence Tools 17 (1), 205–221.

Barták, R., Cepek, O., 2010. Incremental propagation rules for a precedence graph with optional activities and time windows. Transactions of the Institute of Measurement and Control 32 (1), 73–96.

Barták, R., O., C., 2007. Temporal networks with alternatives: complexity and model. In: Proc. FLAIRS. pp. 641–646.

Beck, J., Fox, M., 1998. A generic framework for constraint-directed search and scheduling. AI Magazine 19 (4), 101 – 130.

Beck, J., Fox, M., 2000. Constraint-directed techniques for scheduling alternative activities. International Journal on Artificial Intelligence 121, 211–250.

Bellman, R., 1957. Dynamic Programming. Princeton University Press, Princeton, NJ.

Berry, P., Drabble, B., 2000. Swim: An ai-based system for organizational management. In: The 2nd NASA Intl. WS on Planning and Scheduling for Space. NASA.

Blazewic, J., Pesh, E., Sterna, M., 2000. The disjunctive graph machine representation of the job shop scheduling problem. European Journal of Operational Research 127 (2), 317–331.

Blum, A., Furst, M., 1997. Fast planning through planning graph analysis. Artificial Intelligence 90 (1-2), 281–300.

BPEL, 2007. Web Services Business Process Execution Language Version 2.0: OASIS Standard. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, [Online; accessed 09-November-2011].

BPMN, 2011. Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0/, [Online; accessed 09-November-2011].

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41 (3), 157 – 183.

Brucker, P., Knust, S., 2006. Complex Scheduling (GOR-Publications). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Calton, T., 1999. Advancing design-for-assembly. the next generation in assembly planning. In: IEEE International Symposium on Assembly and Task Planning. pp. 57 – 62.

Caron, F., Vanthienen, J., 2011. An exploratory approach to process lifecycle transitions from a paradigm-based perspective. In: Proc. BPMDS and EMMSAD. pp. 178–185.

Chankong, V., Haimes, Y., 1983. Multiobjective Decision Making Theory and Methodology. Elsevier.

Chaturvedi, A., Hutchinson, G., Nazareth, D., 1993. Supporting complex real-time decision making through machine learning. Decision Support Systems 10 (2), 213–233.

Chen, Y., Hsu, C., Wah, B., 2006. Temporal planning using subgoal partitioning and resolution in sgplan. Journal of Artificial Intelligence Research 26, 323–369.

Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S., 2009. Exploiting inductive logic programming techniques for declarative process mining. In: Proc. ToPNoC. pp. 278–295.

Clarke Jr., E., Grumberg, O., Peled, D., 1999. Model Checking. The MIT Press.

Coello, C., 2006. Evolutionary multi-objective optimization: A historical view of the field. IEEE Computational Intelligence Magazine 1 (1), 28–36.

Davenport, T. H., 1993. Process innovation: reengineering work through information technology. Harvard Business School Press.

De Castro, V., Marcos, E., 2009. Towards a service-oriented mda- based approach to the alignment of business processes with it systems: from the business model to a web service composition model. International Journal of Cooperative Information Systems 18 (2), 225 – 260.

Deb, K., 2008. Introduction to evolutionary multiobjective optimization. 5252 LNCS, 59–96.

Dechter, R., 2003. Constraint Processing. Morgan Kaufmann Publishers.

Del Valle, C., Márquez, A., Barba, I., 2010. A CSP model for simple non-reversible and parallel repair plans. Journal of Intelligent Manufacturing 21 (1), 165–174.

Demeyer, R., Van Assche, M., Langevine, L., Vanhoof, W., 2010. Declarative workflows to efficiently manage flexible and advanced business processes. In: Proc. PPDP. pp. 209–218.

Dourish, P., Holmes, J., MacLean, A., Marqvardsen, P., Zbyslaw, A., 1996. Freeflow: Mediating between representation and action in workflow systems. In: Proc. CSCW. pp. 190–198.

Drabble, B., Tate, A., 1994. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In: Proc. AIPS. pp. 243–248.

Drexl, A., Gruenewald, J., 1993. Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions (Institute of Industrial Engineers) 25 (5), 74–81.

Dumas, M., van der Aalst, W., ter Hofstede, A. (Eds.), 2005. Process-Aware Information Systems: Bridging People and Software through Process Technology. Wiley-Interscience, Hoboken, NJ.

Dynadec, 2011. Comet Downloads. http://dynadec.com/support/downloads/, [Online; accessed 09-November-2011].

Ehrgott, M., 2005. Multicriteria Optimization. Springer Berlin.

Ehrgott, M., Ruzika, S., 2008. Improved ε-constraint method for multiobjective programming. Journal of Optimization Theory and Applications 138 (3), 375–396.

Elgammal, A., Turetken, O., Van Den Heuvel, W., Papazoglou, M., 2011. On the formal specification of regulatory compliance: A comparative analysis. In: Proc. ICSOC Workshops. pp. 27–38.

Ellis, C., Nutt, G., 1993. Modeling and enactment of workflow systems. In: Proc. PETRI NETS. Springer Berlin / Heidelberg, pp. 1–16.

Erol, K., Hendler, J., Nau, D., 1994. HTN planning: Complexity and expressivity. In: Proc. of 20th AAAI Conference. pp. 1123–1128.

Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S., 2009. Declarative versus imperative process modeling languages: The issue of understandability. In: Proc. BPMDS 2009 and EMMSAD 2009. pp. 353–366.

Fahland, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S., 2010. Declarative versus imperative process modeling languages: The issue of maintainability. In: Proc. BPM Workshops. pp. 477–488.

Feo, T., Resende, M., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133.

Fernandes, R., Lange, F., Burchett, R., Happ, H., Wirgau, K., 1983. Large scale reactive power planning. IEEE transactions on power apparatus and systems PAS-102 (5), 1083–1088.

Ferreira, H., Ferreira, D., 2006. An integrated life cycle for workflow management based on learning and planning. International Journal of Cooperative Information Systems 15 (4), 485 – 505.

Fikes, R., Nilsson, N., 1971. Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208.

Fonseca, C., Fleming, P., 1995. An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation 3 (1), 1–16.

Friedrich, G., Fugini, M., Mussi, E., Pernici, B., Tagni, G., 2010. Exception Handling for Repair in Service-Based Processes. IEEE Transactions on Software Engineering 36 (2), 198–215.

Frost, D., Dechter, R., 1994. Dead-end driven learning. In: Proc. of the National Conference on Artificial Intelligence. pp. 294–300.

Gabriel, G., Grandcolas, S., 2009. Searching optimal parallel plans: A filtering and decomposition approach. pp. 576–580.

Gantt, H., 1913. Work, wages, and profits. Engineering Magazine Co.

Garey, M. R., Johnson, D. S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co.

Garrido, A., Arangu, M., Onaindia, E., 2009. A constraint programming formulation for planning: From plan scheduling to plan generation. Journal of Scheduling 12 (3), 227–256.

Garrido, A., Onaindia, E., Sapena, O., 2008. Planning and scheduling in an e-learning environment. a constraint-programming-based approach. Engineering Applications of Artificial Intelligence 21 (5), 733–743.

Georgakopoulos, D., Hornick, M., Sheth, A., 1995. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. Distributed and Parallel Databases 3, 119–153.

Gerevini, A., Long, D., 2006. Preferences and soft constraints in pddl3. In: ICAPS 2006 Ws on Preferences and Soft Constraints in Planning. pp. 46 – 53.

Ghallab, M., et al., 1998. Pddl - the planning domain definition language. Tech. rep., CVC TR-98-003/DCS TR-1165.

Ghallab, M., Nau, D., Traverso, P., 2004. Automated Planning: Theory and Practice. Morgan Kaufmann, Amsterdam.

Glance, N., Pagani, D., Pareschi, R., 1996. Generalised process structure grammars (GPSG) for flexible representations of work. In: Press, A. (Ed.), Proc. CSCW. pp. 190–198.

Glover, F., 1989. Tabu search part i. Orsa Journal on Computing 1 (3), 190–206.

Goedertier, S., Vanthienen, J., 2009. An overview of declarative process modeling principles and languages. In: Proc. ABIS. pp. 51 – 58.

Goldberg, D., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading.

Goldmann, S., Münch, J., Holz, H., 2000. Distributed Process Planning Support with MILOS. International Journal of Software Engineering and Knowledge Engineering 10 (4), 511–525.

Goldwasser, M. H., Motwani, R., 1999. Complexity measures for assembly sequences. International Journal of Computational Geometry and Applications 9, 371 – 418.

Gomes, C., Van Hoeve, W., Selman, B., 2006. Constraint programming for distributed planning and scheduling. Vol. SS-06-04. pp. 157–158.

González-Ferrer, A., Fernández-Olivares, J., Castillo, L., 2009. JABBAH: A Java Application Framework for the Translation Between Business Process Models and HTN. In: Proc. ICKEPS. pp. 28–37.

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.

Haimes, Y., Lasdon, L., Wismer, D., 1971. On a bicriterion formulation of the problems of integrated system identification and system optimization. IEEE Transactions on Systems, Man, and Cybernetics 1, 296–297.

Haisjackl, C., Weber, B., 2010. User Assistance During Process Execution - An Experimental Evaluation of Recommendation Strategies. In: Proc. BPI.

Hallé, S., Villemaire, R., 2008. Runtime monitoring of message-based workflows with data. In: EDOC. IEEE Computer Society, pp. 63–72.

Hammond, K., 1990. Case-based planning: A framework for planning from experience. Cognitive Science 14 (3), 385–443.

Hoffmann, J., Edelkamp, S., 2005. The deterministic part of ipc-4: an overview. Journal of Artificial Intelligence Research 24 (1), 519–579.

Hoffmann, J., Weber, I., Kraft, F., 2010. SAP speaks PDDL. In: Proc. AAAI. pp. 1096–1101.

Homem de Mello, L., Sanderson, A., 1990. And/or graph representation of assembly plans. IEEE Transactions on Robotics and Automation 6(2), 188–189.

Homem de Mello, L., Sanderson, A., 1991. A correct and complete algorithm for the generation of mechanical assembly sequences. IEEE Transactions Robotic & Automation 27 (2), 228 – 240.

ILOG, 2011. IBM ILOG CPLEX CP Optimizer. http://www-01.ibm.com/software/integration/optimization/cplex-cp-optimizer/, [Online; accessed 09-November-2011].

Jarvis, P., et al., 2000. Applying intelligent workflow management in the chemicals industries. In: The workflow handbook 2001, Published in association with the Workflow Management Coalition (WfMC). L. Fisher (Ed.).

Joeris, G., 2000. Decentralized and flexible workflow enactment based on task coordination agents. In: Springer-Verlag (Ed.), Proc. Caise. pp. 41–62.

Joshi, S., Kersting, K., Khardon, R., 2011. Decision-theoretic planning with generalized first-order decision diagrams. Artificial Intelligence 175 (18), 2198–2222.

Kim, I.Y., D. W. O., 2006. Adaptive weighted sum method for multiobjective optimization: A new method for pareto front generation. Structural and Multidisciplinary Optimization 31 (2), 105–116.

Kirkpatrick, S., Gelatt, C., Vecchi, M., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

Koehler, J., 1998. Planning under resource constraints. In: Proc. ECAI. pp. 489–493.

Koski, J., 1985. Defectiveness of weighting method in multicriterion optimization of structures. Communications in Numerical Methods in Engineering 1 (6), 333–337.

La Rosa, M., Dumas, M., Uba, R., Dijkman, R., 2010. Merging business process models. pp. 96–113.

Laborie, P., Ghallab, M., 1995. Planning with sharable resource constraints. In: Proc. IJCAI. pp. 1643–1649.

Laborie, P., Rogerie, J., Shaw, P., Vilím, P., 2009. Reasoning with Conditional Time-Intervals. Part II: An Algebraical Model for Resources. In: Proc. FLAIRS.

Lambert, A., 1997. Optimal disassembly of complex products. International Journal of Production Research 35, 2509 – 2523.

Lambert, A., 1999. Optimal disassembly sequence generation for combined material recycling and part reus. In: IEEE International Symposium on Assembly and Task Planning. pp. 146 – 151.

Lambert, A., 2003. Disassembly sequencing: a survey. International Journal of Production Research 41 (16), 3721–3759.

Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S., 2007. Inducing declarative logic-based models from labeled traces. In: Proc. BPM. pp. 344–359.

Larrosa, J., 2000. Boosting search with variable elimination. In: Proc. CP. pp. 291–305.

Larrosa, J., Meseguer, P., 2003. Algoritmos para satisfaccin de restricciones. Inteligencia Artificial: Revista Iberoamericana de Inteligencia Artificial 20, 31–42.

Le Pape, C., Couronne, P., Vergamini, D., Gosselin, V., 1994. Time-versus-capacity compromises in project scheduling. In: Proc. PlanSIG. pp. 498–502.

Lecoutre, C., Vion, J., 2008. Enforcing arc consistency using bitwise operations. Constraint Programming Letters 2, 21–35.

Lekavy, M., Návrat, P., 2007. Expressivity of STRIPS-Like and HTN-Like Planning. In: Proc. KES-AMSTA. pp. 121–130.

Li, W., Zhang, C., 1995. Design for disassembly analysis for environmentally conscious design and manufacturing. In: ASME International Mechanical Engineering Congress. pp. 969 – 976.

Liu, Y., Jiang, Y., 2006. Lp-tpop: Integrating planning and scheduling through constraint programming. In: Proc. PRICAI. pp. 844–848.

Liu, Y., Müller, S., Xu, K., 2007. A static compliance-checking framework for business process models. IBM Systems Journal 46 (2), 335–362.

Lombardi, M., Milano, M., 2010. Constraint based scheduling to deal with uncertain durations and self-timed execution. In: Proc. CP. pp. 383–397.

Lu, R., Sadiq, S., Padmanabhan, V., Governatori, G., 2006. Using a temporal constraint network for business process execution. In: Australian Computer Society, I. (Ed.), Proc. ADC. pp. 157–166.

Ly, L., Rinderle, S., P., D., 2008. Integration and verification of semantic constraints in adaptive process management systems. Data & Knowledge Engineering 64 (1), 3 – 23.

Mackworth, A., 1977. Consistency in networks of relations. Artificial Intelligence 8 (1), 99–118.

Marrella, A., Mecella, M., 2011. Continuous Planning for Solving Business Process Adaptivity. In: Proc. BPMDS and EMMSAD. pp. 118–132.

Miettinen, K., 1999. Nonlinear Multiobjective Optimization. Kluwer Academic Dordrecht.

Mitchell, M., 1998. An Introduction to Genetic Algorithms. MIT Press.

Monette, J., Deville, Y., Van Hentenryck, P., 2009. Just-in-time scheduling with constraint programming. pp. 241–248.

Montali, M., 2009. Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach. Ph.D. thesis, Department of Electronics, Computer Science and Telecommunications Engineering, University of Bologna.

Mouhoub, M., Sadaoui, S., Sukpan, A., 2003. Chronological backtracking versus formal methods for solving CSPs. In: Proc. of the International Conference on Artificial Intelligence IC-AI 2003. pp. 270–275.

Moura, A., De Souza, C., Cire, A., Lopes, T., 2008. Heuristics and constraint programming hybridizations for a real pipeline planning and scheduling problem. pp. 455–462.

Nareyek, A., Freuder, E., Fourer, R., Giunchiglia, E., Goldman, R., Kautz, H., Rintanen, J., Tate, A., 2005. Constraints and ai planning. IEEE Intelligent Systems 20 (2), 62 – 72.

Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, J., Wu, D., Yaman, F., 2003. Shop2: An htn planning system. Journal of Artificial Intelligence Research 20, 379–404.

Nuijten, W., Aarts, E., 1996. Sequencing with earliness and tardiness penalties: a review. European Journal of Operational Research 90 (2), 269 – 284.

Ouyang, C., van der Aalst, W., Dumas, M., ter Hofstede, A., 2006. Translating BPMN to BPEL. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.79.3072&rep=rep1&type=pdf, [Online; accessed 09-November-2011].

Owen, M., Raj, J., 2003. BPMN and Business Process Management Introduction to the New Business Process Modeling Standard. http://www.omg.org/bpmn/Documents/6AD5D16960.BPMN_and_BPM.pdf, [Online; accessed 09-November-2011].

Özbayrak, M., Bell, R., 2003. A knowledge-based decision support system for the management of parts and tools in FMS. Decision Support Systems 35 (4), 487–515.

Penberthy, J., Weld, D., 1994. Temporal planning with continuous change. In: Proc. AAAI. pp. 1010–1015.

Pesic, M., 2008. Constraint-Based Workflow Management Systems: Shifting Control to Users. Ph.D. thesis, Eindhoven University of Technology, Eindhoven.

Pesic, M., Schonenberg, M., Sidorova, N., van der Aalst, W., 2007. Constraint-Based Workflow Models: Change Made Easy. In: OTM Conferences (1). pp. 77–94.

Pesic, M., van der Aalst, W., 2006. A declarative approach for flexible business processes management. In: Proc. BPM Workshops. pp. 169–180.

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H., 2011. Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In: Proc. ER-BPM (accepted). [Online; http://zugal.info/wp-content/uploads/2011/07/er_bpm_2011.pdf; accessed 8-September-2011].

Pinedo, M., 2008. Scheduling - Theory, Algorithms, and Systems. Springer.

Pisinger, D., Ropke, S., 2010. Large neighborhood search. International Series in Operations Research & Management Science 146 (13), 399–420.

PLANET, 2003. Network home page: European network of excellence in ai planning. http://planet.dfki.de/index.html, [Online; accessed 09-November-2011].

Prosser, P., 1993. An empirical study of phase transitions in binary constraint satisfaction problems. Artificial Intelligence 81, 81–109.

R-Moreno, M., Borrajo, D., Cesta, A., Oddi, A., 2007. Integrating planning and scheduling in workflow domains. Expert Systems with Applications 33 (2), 389–406.

Reijers, H., 2003. Design and Control of Workflow Processes. Springer-Verlag Berlin, Heidelberg.

Reijers, H., van der Aalst, W., 1999. Short-term simulation: bridging the gap between operational control and strategic decision making. In: Proc. IASTED Conference on Modeling and Simulation. p. 417421.

Rhee, S., Cho, N., Bae, H., 2008. Increasing the efficiency of business processes using a theory of constraints. Information Systems Frontiers, 1–13.

Rhee, S., Cho, N., Bae, H., 2010. Increasing the efficiency of business processes using a theory of constraints. Information Systems Frontiers 12 (4), 443–455.

Rossi, F., van Beek, P., Walsh, T. (Eds.), 2006. Handbook of Constraint Programming. Elsevier.

Rozinat, A., Wynn, M., van der Aalst, W., ter Hofstede, A., Fidge, C., 2009. Workflow simulation for operational decision support. Data & Knowledge Engineering 68 (9), 834–850.

Russell, N., van der Aalst, W., ter Hofstede, A., 2006. Workflow Exception Patterns. In: Proc. Caise. pp. 288–302.

Russell, N., van der Aalst, W., ter Hofstede, A., Edmond, D., 2005. Workflow resource patterns: Identification, representation and tool support. In: Proc. Caise. pp. 216–232.

Rychkova, I., Regev, G., Wegmann, A., 2008a. High-level design and analysis of business processes: The advantages of declarative specifications. In: Proc. RCIS. pp. 99–110.

Rychkova, I., Regev, G., Wegmann, A., 2008b. Using declarative specifications in business process design. International Journal of Computer Science and Applications 5 (3b), 45 – 68.

Sabin, D., Freuder, E., 1994. Contradicting convectional wisdom in constraint satisfaction. In: Proc. ECAI. pp. 125–129.

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

Salido, M., 2010. Introduction to planning, scheduling and constraint satisfaction. Journal of Intelligent Manufacturing 21 (1), 1–4.

Schonenberg, H., Weber, B., van Dongen, B., van der Aalst, W., 2008. Supporting flexible processes through recommendations based on history. In: Proc. BPM. pp. 51–66.

Smith, D., Frank, J., Jónsson, A., 2000. Bridging the gap between planning and scheduling. Knowledge Engineering Review 15(1), 47–83.

Smith, D., Weld, D., 1999. Temporal planning with mutual exclusion reasoning. In: Proc. IJCAI. pp. 139–144.

Son, J., Kim, M., 2001. Improving the performance of time-constrained workflow processing. Journal of Systems and Software 58 (3), 211–219.

Thompson, G., Goodale, J., 2006. Variable employee productivity in workforce scheduling. European Journal of Operational Research 170 (2), 376–390.

Timpe, C., 2002. Solving planning and scheduling problems with combined integer and constraint programming. OR Spectrum 24 (4), 431–448.

Tjoa, S., Jakoubi, S., Goluch, S., Kitzler, G., 2010. Planning dynamic activity and resource allocations using a risk-aware business process management approach. In: Proc. ARES. pp. 268–274.

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.

Tu, P., Son, T., Pontelli, E., 2007. CPP: A constraint logic programming based planner with preferences. In: Proc. LPNMR. pp. 290–296.

van der Aalst, W., 2003. Patterns and XPDL: A critical Evaluation of the XML Process Definition Language. In: QUT Technical report FIT-TR-2003-06. pp. 1–30.

van der Aalst, W., Jablonski, S., 2000. Dealing with workflow change: identification of issues and solutions. International Journal of Computer Systems Science and Engineering 15 (5), 267–276.

van der Aalst, W., Pesic, M., 2006a. DecSerFlow: Towards a Truly Declarative Service Flow Language. In: LNCS 4184. pp. 1–23.

van der Aalst, W., Pesic, M., 2006b. Specifying, discovering, and monitoring service flows: Making web services process-aware. In: Technical Report BPM-06-09, BPMcenter.org.

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

van der Aalst, W., Schonenberg, M., Song, M., 2011. Time prediction based on process mining. Information Systems 36 (2), 450–475.

van der Aalst, W., ter Hofstede, A., Weske, M., 2003. Business Process Management: A Survey. In: Proc. BPM. pp. 1–12.

van der Aalst, W., van Hee, K., 2002. Workflow Management: Models, Methods, and Systems. MIT Press.

van Dongen, B., 2007. Process Mining and Verification. Ph.D. thesis, Eindhoven University of Technology, Eindhoven.

van Dongen, B., van der Aalst, W., 2005. A Meta Model for Process Mining Data. In: Proc. Caise, 17th Edition. pp. 309–320.

van Hentenryck, P., 1999. The OPL Optimization Programming Language. MIT Press.

Vanderfeesten, I., Reijers, H., van der Aalst, W., 2008. Product based workflow support: A recommendation service for dynamic workflow execution. In: Technical Report BPM-08-03, BPMcenter.org.

Vidal, V., Geffner, H., 2006. Branching and pruning: An optimal temporal pocl planner based on constraint programming. Artificial Intelligence 170 (3), 298–335.

Vossen, T., Ball, M., Lotem, A., Nau, D., 1999. On the use of integer programming models in ai planning. The Knowledge Engineering Review 15 (1).

Wainer, J., Bezerra, F., Barthelmess, P., 2004. Tucupi: a flexible workflow system based on overridable constraints. In: Proc. SAC. pp. 498–502.

Wainer, J., De Lima Bezerra, F., 2003. Constraint-based flexible workflows. In: Proc. CRIWG. pp. 151–158.

Weld, D., 1994. Introduction to least commitment planning. AI Magazine 15 (4), 27–61.

Weske, M., 2007. Business Process Management: Concepts, Methods, Technology. Springer.

Westergaard, M., 2011. Access/cpn 2.0: A high-level interface to coloured petri net models. In: Proc. PETRI NETS. pp. 328–337.

White, S., et al., 2004. Business Process Modeling Notation (BPMN), Working draft, Version 1.0.

Wolfman, S., Weld, D., 1999. Combining linear programming and satisfiability solving for resource planning. The Knowledge Engineering Review 15 (1).

Younes, H., Simmons, R., 2003. VHPOP: Versatile heuristic partial order planner. Journal of Artificial Intelligence Research 20, 405–430.

Zadeh, L., 1963. Optimality and non-scalar-valued performance criteria. IEEE Transactions on Automatic Control 8, 59–60.

Zhao, J., Stohr, E., 1999. Temporal workflow management in a claim handling system. SIGSOFT: Software Engineering Notes 24 (2), 187–195.

Zisman, M., 1977. Representation, Specification and Automation of Office Procedures. Ph.D. thesis, Wharton School.