



HAL
open science

Case Management Process Analysis and Improvement

Shaowei Wang

► **To cite this version:**

Shaowei Wang. Case Management Process Analysis and Improvement. Computers and Society [cs.CY]. Université Clermont Auvergne [2017-2020], 2017. English. NNT: 2017CLFAC087. tel-01823797

HAL Id: tel-01823797

<https://theses.hal.science/tel-01823797>

Submitted on 26 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : D. U : 828



UNIVERSITÉ CLERMONT AUVERGNE

ÉCOLE DOCTORALE DES SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

Thèse

Présentée par

Shaowei WANG

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

SPÉCIALITÉ : Informatique

Titre de la thèse :

**Case Management Process Analysis and
Improvement**

Directeur de thèse : TRAORE Mamadou Kaba

Soutenue publiquement le 08/12/2017 devant le jury :

FRYDMAN Claudia
ZACHAREWICZ Gregory
FAROUK Toumani
TRAORE Mamadou Kaba

Université Aix-Marseille
Université de Bordeaux
Université Clermont Auvergne
Université Clermont Auvergne

Abstract

In today's business world, customer requirements change more rapidly than ever before, and new competitors are increasing every second. Moreover, the ability of managing changes and unpredictability has become a crucial factor for enterprises to make more value and stay competitive [Oracle 2013]. This results in a fact that nowadays enterprises are challenged with not only managing structured business processes, but also more and more unstructured ones. In a common structured business process, everything regarding the process can be predetermined at design time, such as activities, the execution sequence of activities, and so on. However, in an unstructured one the activities cannot be defined precisely beforehand, as well as the sequence to execute. To stay competitive, meet the ever-changing market demands and improve their business process operational efficiency, organizations need a novel process approach that can help them manage changes, dynamics and unpredictability. Under this context, the concept of Case Management is proposed. Different from Business Process Management (BPM) which standardizes and automates structured business processes, CM overcomes the BPM approach limitations and provides an infrastructure for managing changes, dynamics and unpredictability in unstructured business processes. CM proceeds largely depending on evolving circumstances, and decisions are made on the fly. BPM requires a high level of predictability; whereas CM has a lower level of predictability but a higher level of adaptability and flexibility. With CM approach, enterprises are able to manage their unstructured business processes in a more adaptive and flexible manner.

However, for this new area it lacks supporting methods and software tools. Major concerns are: (1) case modeling (the construction of case models); (2) model discovery (the establishment of case models from raw data); (3) model analysis (the analysis of models in both static and dynamic manners, e.g., the derivation of properties before the case is enacted); (4) model improvement (the reduction of cost, the optimization of operational performance, etc.); and (5) model enactment (the execution of a case scenario with case workers in the loop). After a thorough literature review we found that only a few efforts have been done in (1) and (5), and no noticeable contribution has been done in other aspects.

This these presents our CM approach that provides case workers full supports in the whole lifecycle of CM: from establishing case models from raw data to optimizing case models. Process Tree is our choice to formalize the discovered model, and CMMN (Case Management Model and Notation, a case modeling specification is selected as the formalism for presenting and constructing case models. In addition, we adopt the HiLLS (High Level Language for Systems Specification) formalism to

conciliate usability, simulation ability and formal analysis capabilities together. Dynamic model analysis is enabled by DEVS formalism, static model analysis is provided by formal methods, and model enactment is given by the implementation of an object-oriented specification of the case. We propose mainly two modules in this these: one module concerning the discovery of the case model from historical event logs, and another module concerning the improvement and the optimization of the case model.

Keywords: Case Management, CMMN, HiLLS, Model Transformation, Process Improvement

List of publications:

Shaowei Wang and Mamadou Kaba Traoré. 2014. DEVS-based case management. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative (DEVS '14)*. Society for Computer Simulation International, San Diego, CA, USA, Article 34 , 7 pages.

Shaowei Wang and Mamadou Kaba Traoré. 2014. A system-theoretic approach to case management. In *Proceedings of the 2014 SpringSim Poster Session (Posters '14)*. Society for Computer Simulation International, San Diego, CA, USA, Article 3, 2 pages.

Acknowledgment

I would like to thank my fellow doctoral students for their feedback, cooperation and of course friendship, especially to Oumar and Hamzat. Without the discussions with them and their feedback I could not go this far with this these.

I would like also to thank my thesis director, Professor Mamadou Kaba Traoré. As a foreigner PhD candidate, he helped me a lot both in my research work and my personal life. He showed me how to be a researcher, as well as how to do the research work in an excellent manner.

I would like to thank my friends for accepting nothing less than excellence from me. Last but not the least, I would like to thank my family: my parents and to my wife for supporting me spiritually throughout writing this thesis and my personal life in general.

Table of Content

Abstract.....	I
Acknowledgment.....	III
Table of Content.....	IV
Table of Figures.....	VI
List of Abbreviations.....	IX
1. GENERAL INTRODUCTION.....	1
1.1 Context.....	2
1.1.1 Case Management.....	4
1.1.2 Challenges.....	8
1.2 Objectives.....	9
1.3 Contributions.....	11
1.4 Outline of the thesis.....	11
2. RELATED WORKS.....	12
2.1 Introduction.....	13
2.2 Case Management Modeling Solutions.....	13
2.2.1 Activity-based Modeling.....	14
2.2.2 Information-based Modeling.....	15
2.2.3 Communication-based Modeling.....	16
2.2.4 Hybrid Activity and Information Modeling.....	17
2.2.5 Summary.....	20
2.3 Process Improvement Practice in Knowledge Work.....	23
2.4 Conclusion.....	24
3. Background.....	26
3.1 Introduction.....	27
3.2 Definition of A Modeling Language.....	27
3.3 Case Management Model and Notation.....	29
3.3.1 Abstract Syntax.....	30
3.3.2 Concrete Syntax.....	32
3.3.3 Operational Semantics.....	33
3.3.4 Example.....	35
3.4 High-Level Language for System Specification.....	36
3.4.1 Abstract Syntax.....	38
3.4.2 Concrete Syntax.....	40
3.4.3 Semantics.....	41
3.4.4 Example.....	45
3.5 Process Enhancement Approaches.....	48
3.5.1 Process Analytics.....	49
3.5.2 Process Improvement.....	50

3.6 Process Discovery	62
3.6.1 Process Tree	63
3.7 Model Transformation	66
3.7.1 Declarative Approach.....	68
3.7.2 Imperative Approach.....	69
3.7.3 Hybrid Approach.....	69
3.8 Conclusion	71
4. CASE MODEL IMPROVEMENT	73
4.1 Introduction.....	74
4.2 Case Model Transformation.....	76
4.2.1 System Structure	78
4.2.2 System Behavior	80
4.3 Experimental Frame.....	89
4.3.1 RootStage → HSystem	92
4.3.2 Stage/Task → HSystem	94
4.3.3 Milestone/EventListener → HSystem	104
4.3.4 CaseFileItem → HSystem.....	107
4.3.5 Complementary information	110
4.3.6 Example	111
4.4 Performance Metrics	115
4.4.1 Lean-related Performance Metrics.....	116
4.4.2 TOC-related Performance Metrics.....	119
4.5 Conclusion	120
5. CASE MODEL DISCOVERY	122
5.1 Introduction.....	123
5.2 Translations from Process Tree to CMMN	123
5.3 Model Transformation Algorithm	128
5.4 Conclusion	129
6. CASE STUDY	130
6.1 Introduction.....	131
6.2 Background	131
6.3 Case Model Discovery	132
6.4 Generated HiLLS Model and Experimental Frame	136
6.5 To-be Model	141
6.6 Results Analysis	142
6.7 Conclusion	146
7. GENERAL CONCLUSION	147
7.1 Conclusion	148
7.2 Perspectives.....	150
REFERENCE.....	151

Table of Tables

Table 2-1: Benchmark of case management solutions.....	21
Table 3-1: Common data visualization tools	51
Table 3-2: The summary of performance metrics	61
Table 3-3: The semantics of process tree	65
Table 4-1: The mappings of system structure	78
Table 4-2: States of CMMN modeling elements	81
Table 4-3: Transformation rules of ECT – Part 1.....	86
Table 4-4: Transformation rules of ECT – Part 2.....	87
Table 4-5: Transformation rules of ECT – Part 3.....	88
Table 4-6: Transformation rules of ECT – Part 4.....	89
Table 4-7: The complete information of selected events	112
Table 5-1: The mappings between Process Tree and CMMN - 1	124
Table 5-2: The mappings between Process Tree and CMMN - 2	125
Table 6-1: The historical event data	132
Table 6-2: Information for the to-be model	142

Table of Figures

Figure 1-1: The spectrum of business processes [Kemsley 2011]	6
Figure 3-1: The definition of a modeling language specification	28
Figure 3-2: The core of the CMMN metamodel	30
Figure 3-3: An overview of the CMMN concrete syntax	32
Figure 3-4: An overview of the lifecycles of CMMN essential modeling elements ...	34
Figure 3-5: A CMMN case model.....	36
Figure 3-6: An overview of HiLLS.....	37
Figure 3-7: The HiLLS metamodel.....	38
Figure 3-8: An overview of the HiLLS concrete syntax [Maiga 2015].....	40
Figure 3-9: Basic system concepts.....	42
Figure 3-10: The DEVS-based M&S framework [Zeigler et al. 2000]	44
Figure 3-11: General components of an experimental frame [Zeigler et al. 2000]	45
Figure 3-12: An atomic HiLLS model	46
Figure 3-13: A coupled HiLLS model	47
Figure 3-14: A general scenario for process improvement	52
Figure 3-15: The framework for process reengineering	53
Figure 3-16: The PDCA cycle.....	57
Figure 3-17: The metamodel of process tree	64
Figure 3-18: A simple process tree model.....	66
Figure 3-19: Basic components of a model transformation.....	67
Figure 3-20: A fragment of ATL transformation rules	70
Figure 4-1: A global view of our CMM approach	74
Figure 4-2: The flow chart of our CMM approach	75
Figure 4-3: The whole picture of the CMM approach.....	76
Figure 4-4: The workflow of CMI	77
Figure 4-5: The whole picture of CMI.....	77
Figure 4-6: The system structure of a HiLLS model	80
Figure 4-7: An example of configuration transitions	83
Figure 4-8: Different types of configuration transitions	83
Figure 4-9: Examples of $t2f$ and $t2p$ configuration transitions	84
Figure 4-10: An example of $f2p$ configuration transitions.....	85
Figure 4-11: An example of ECT	89
Figure 4-12: The couplings between different HiLLS models	90
Figure 4-13: The flow chart of generating EFs.....	91
Figure 4-14: The configuration transition of event ($CFM, create, -, -$).....	112
Figure 4-15: The configuration transition of event ($CFM, suspend, 1, [100, 180]$)..	113
Figure 4-16: The configuration transition of event ($IR, complete, 4, [10, 15]$).....	113
Figure 4-17: The EOC specified in EventGenerator.....	114

Figure 4-18: The IC specified in EventGenerator.....	114
Figure 4-19: The graphical representation of Analyzer	115
Figure 5-1: The CMD module	123
Figure 5-2: The algorithm of the Process Tree to CMMN model transformation	128
Figure 6-1: The resulting process tree model	133
Figure 6-2: The model transformation example – 1	134
Figure 6-3: The model transformation example - 2	134
Figure 6-4: The as-as CMMN case model.....	135
Figure 6-5: The as-is HiLLS model in a tree-view format	136
Figure 6-6: The graphical representation of the structure of the as-is HiLLS model	136
Figure 6-7: The partial graphical behavior representation of the as-is HiLLS model	137
.....	
Figure 6-8: The event generator Hills model.....	137
Figure 6-9: The RS_EG HiLLS model’s coupling information.....	138
Figure 6-10: The RS_Suspend atomic HiLLS model	139
Figure 6-11: The complementary information.....	139
Figure 6-12: The event generator.....	140
Figure 6-13: The system behavior of A1Complete.....	140
Figure 6-14: The CMMN to-be case model.....	141
Figure 6-15: The HiLLS to-be model in the tree-view editor.....	141
Figure 6-16: The backlog in the as-is model.....	143
Figure 6-17: The idle time in the as-is model	143
Figure 6-18: The backlog in the to-be model.....	144
Figure 6-19: The idle time in the to-be model	144
Figure 6-20: The comparison of lead-time	145
Figure 6-21: The comparison of WIPs.....	145

List of Abbreviations

ACOM	Artifact-Centered Operational Modeling
ADoc	Adaptive Documents
ATL	ATLAS Transformation Language
AToM ³	A Tool for Multi-formalism and Meta-Modeling
BPI	Business Process Intelligence
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPR	Business Process Re-engineering
CDEVS	Classic DEVS
CMCV	Case Model Conformance Validation
CMD	Case Model Discovery
CMI	Case Model Improvement
CMM	Case Model Management
CMMN	Case Management Model and Notation
CRM	Customer Relationship Management
DDML	DEVS-Driven Modeling Language
DES	Discrete-Event System
DEVS	Discrete-Event System Specification
ECM	Enterprise Content Management
ECT	External Configuration Transition
EF	Experimental Frame
EOC	External Output Coupling
ERP	Enterprise Resource Planning
ETM	Evolutionary Tree Miner
ETMd	Evolutionary Tree Miner discovery
F2P	Finite to Passive
F2T	Finite to Transient
FM	Formal Method
HiLLS	High Level Language for Systems Specification
IC	Internal Coupling
ICT	Internal Configuration Transition
IDEF	Integrated computer-aided manufacturing DEFinition
IEF	Intermediate Experimental Frame
JIT	Just-In-Time
KPI	Key Performance Indicator
LHS	Left-Hand Side
M&S	Modeling and Simulation
MCT	Manual Credit Transfer
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
M2M	Model-To-Model
M2T	Model-To-Text
OE	Operating Expenses
OMG	Object Management Group

P2T	Passive to Passive
PDCA	Plan-Do-Check-Act
PDEVS	Parallel DEVS
QCD	Quality, Cost and Delivery
QVT	Query/View/Transformation
RAD	Role Activity Diagram
RHS	Right-Hand Side
SLA	Service Level Agreement
T2F	Transient to Finite
T2P	Transient to Passive
TA	Throughput Accounting
TOC	Theory of Constraints
TPS	Toyota Production System
UML	Unified Modeling Language
WIP	Work-In-Progress

1. GENERAL INTRODUCTION

1.1 Context

In today's business world, customer requirements change more rapidly than ever before, and new competitors are increasing every minute. To stay competitive in business, organizations put their focuses on managing their business using scientific management approaches. Examples include:

- Customer Relationship Management (CRM), a management approach focusing on the management of the organization's interaction with their customers, and the goal of CRM is to drive sales growth through the improvement of business relationships with customers [BRIEF 2015].
- Enterprise Content Management (ECM), which refers to a set of strategies, methods, and tools developed for capturing, preserving and controlling process-relating information of any form (e.g., a paper document, an email).
- Enterprise Resource Planning (ERP), which aims at increasing the operational efficiency and effectiveness through integrating and sharing business activity data and standardizing business processes from best practices [Seo 2013].
- Business Process Management (BPM), the most important and widely used process management approach in recent years. Instead of focusing on the management of business activity information, BPM aims at improving corporate performance by managing and optimizing organizations business processes [Page 2015], and it has been the most widely studied and implemented process management methodology in business domain. A key feature of BPM is that it targets on automating routine work (and processes within routine work are called structured business processes): activities within such type of processes are completely predictable and repeatable. Everything regarding the process can be predetermined during process-design time, such as activities, the execution sequence of activities, and so on. It leads to a work pattern where BPM workers do their work by strictly following predefined process solutions.

However, in today's enterprises managers realize that the ability of managing knowledge work becomes a crucial factor for them to make more value and stay competitive. Knowledge work refers to the type of work in which the plan is determined and altered by the situation information obtained at execution time [Swenson 2010]. Typical examples of knowledge work include medical work (the procedures of treatment cannot be predicted since judgments from doctors are required based on patients symptoms and their diagnosis results), customer support (it is difficult or even impossible to know which type of service is required in advance due to the fact that many details come late), law enforcement (the course of the

investigation of a crime depends on the details unfold as time goes by, as well as the knowledge and experience of investigators), etc. All these use cases illustrate the unique features of knowledge work: (1) unpredictable, and (2) non-repeated.

We say that routine work is highly predicted since all the necessary activities required for completing that work can be specified beforehand. However, it is difficult to define how a knowledge work will be done (in terms of specifying all the necessary activities and the execution sequence of those activities) since the course of events is determined by many potential factors as the work proceeds. This is the essential nature of knowledge work: its plan changes in accordance with incoming situation information. As knowledge work proceeds, new information will be generated, and this new knowledge sets the next direction towards which the work will carry on. Then more information will be obtained, and the plan set from the last step will be altered based on that new discovered knowledge, and so on. This explains the reason why knowledge work is considered to be unpredictable. For example, a doctor cannot decide the exact treatment procedures unless he receives enough information regarding the patient condition. But such original treatment procedures will not stay unchanged to the end of the treatment. The patient will come for another diagnosis or an examination after a while, and it is his diagnosis result that determines the treatment to be taken for the next step. The same story repeats until the patient is recovered from his illness.

Moreover, in most situations knowledge work does not have the same level of repeatability as routine work has. Once a routine work scenario is standardized, all steps as well as the sequence of them will be specified and stay unchanged, and workers do their work by simply following what have been defined previously, without making any decision. Knowledge work, on the contrary, has many factors that make each instance a unique one. When dealing with knowledge work, the different specifics of each case cause a unique solution to meet its special requirements. Take the example we have mentioned before, due to the fact that patients come under different conditions, the doctor needs to formulate different treatment procedures considering the uniqueness of individual, in order to meet the needs of each particular case.

The first reference of the knowledge work concept was mentioned in [Drucker 1959], a book of Peter F. Drucker's named *Landmarks of Tomorrow*. Afterwards, the importance of knowledge work has been discussed by him and other researchers. Drucker pointed out that the most important task for organizations was "*to make knowledge work productive*" in [Drucker 1969]. Later in 1999 he emphasized that knowledge workers will be the most valuable asset in the 21st century, and their productivity will be the key factor to business success [Drucker 1999]. Davenport

discussed the importance of knowledge work from the perspective of what knowledge work can yield to organizations: they could come with new management strategies (if they are at the management hierarchy), they are able to design and create new products (if they are in the Research and Development department), they are capable of advertising their products and services in ways that attract customers (if they are in charge of selling products), and so on [Davenport 2005]. However, efforts regarding theories and approaches of work management and improvement in business domain have focused largely on repeatable routine work. A large number of process management theories have been proposed, and they have been implemented in the real world to help organizations solve their real business problems. The management of knowledge work did not gain much attention until 2010. Since then people start to consider how to improve the knowledge work existing within their business activities. However, due to its unique unpredictable and non-repeated features, the solutions for the pattern of repeated routine work is not appropriate any more. Therefore, a fundamental different process management methodology is required, which helps knowledge workers adapt changes, control dynamics and unpredictability as they work. This is the approach we will introduce and explain in the next section, i.e., Case Management.

1.1.1 Case Management

As more knowledge work appears in business domain over the past few years, managers start to be aware of the importance of the operational efficiency of unstructured processes with knowledge work, which needs a greater deal of flexibility, adaptability, autonomous decision making and collaboration to achieve organizations business objectives [Zhu et al. 2013]. In today's modern organizations, structured business processes cover around only 30%. Moreover, the following factors from different aspects serve as the driving forces for the research and development interest in a novel support for managing unstructured business processes [Swenson 2010]:

- An increased need to govern the costs and risks rising from the service management (e.g., after-sale service, claims management).
- An increased emphasis on managing (i.e., automating and tracking) ad-hoc processes which are not defined precisely beforehand, where an ad-hoc process refers to a type of process that is not predictable.
- A demand from government agencies of responding massive citizen requests in a rapid manner.
- A need for regulators, auditors, and litigants to quickly react to external regulations.
- An increased demand of managing business with unstructured processes

collaboratively and communicably.

Apparently, traditional process management approaches such as BPM or ECM are not sufficient, in some cases even not able, to handle knowledge work processes due to its unique features. Under this context, the concept of Case Management (also known as Adaptive Case Management, Advanced Case Management, etc.) is proposed. Different from BPM which targets on standardizing and automating structured business processes, case management overcomes the BPM approach limitations and provides an infrastructure for managing unstructured business processes. More specifically, case management deals with changes, dynamics and unpredictability. It proceeds largely depending on evolving circumstances, and decisions are made on the fly. Case management provides knowledge workers sufficient capabilities (flexibility, adaptability, autonomous decision making and collaboration) so that they can achieve positive outcomes.

The term of case management has different meanings in different areas, and this concept was referred only in domains such as legal, social work, healthcare markets and government, in a general sense. For example, in the medical care field, it refers to the planning and cooperation of health care services. In social service, it refers to the process of planning, seeking, and monitoring different social services [Anastas & Clark 2013]. However, in recent years case management has been widely accepted by business organizations, and people render more general definitions of this concept, instead of restricting to one or several specific area(s). [OMG 2014] formally defines the concept of Case as “*a proceeding that involves actions taken regarding a subject in a particular situation to achieve a desired outcome.*” A well accepted generic definition regarding case management is given by Forrester (one of the most powerful research firm that aims at helping clients improve their business results) in [Clair & Moore 2009], where the case management is described as “*a collaborative, dynamic, and information-intensive process driven by outside events requiring incremental and progressive responses from the business domain handling the case.*” [Swenson 2010] also defined case management as “*Systems that are able to support decision making and data capture while providing the freedom for knowledge workers to apply their own understanding and subject matter expertise to respond to unique or changing circumstances within the business environment.*” Many vendors of BPM systems gave their understanding about case management as well, such as Cordys, Global 360, which will be discussed in more detail later in this thesis. All the following statements clearly illustrate the features of case management from the different aspects, and it is crucial to well understand its nature since we can judge which approach will fit and support this kind of work:

- Case management targets on governing unstructured processes occurring in

knowledge work. The exact activities required are unknown beforehand, as well as their execution sequence.

- Case management is information sensitive. The unfolding information influences how a case proceeds.
- Case management empowers knowledge workers. Unlike BPM where employees are asked to only follow what has been specified, knowledge workers in case management have the power and freedom to make decisions.
- Case management has a clear goal to be achieved. The exact path to reach the goal unfolds by yielded information and knowledge workers' decisions, gradually.
- Case management requires communication and collaboration. Information should be shared and assessed by all relating knowledge workers, so that they can make better decisions when responding to ad-hoc events.

In a nutshell, the essential characteristics of case management are unpredictability, information-sensitivity, runtime planning and collaboration. Each Case in case management comes with an explicit goal, and it evolves based on the unfolding circumstances and information yielded from previous steps to achieve that goal. Moreover, it is the data collected and knowledge workers' decisions that determine how a Case proceeds. Therefore, to ensure efficient case management practices it is inevitable for knowledge workers to (at least but not limited to): (1) determine which activities should be involved; (2) decide the execution sequence of selected activities; and (3) cooperate with other colleagues for decision-making (de Man, 2009a).

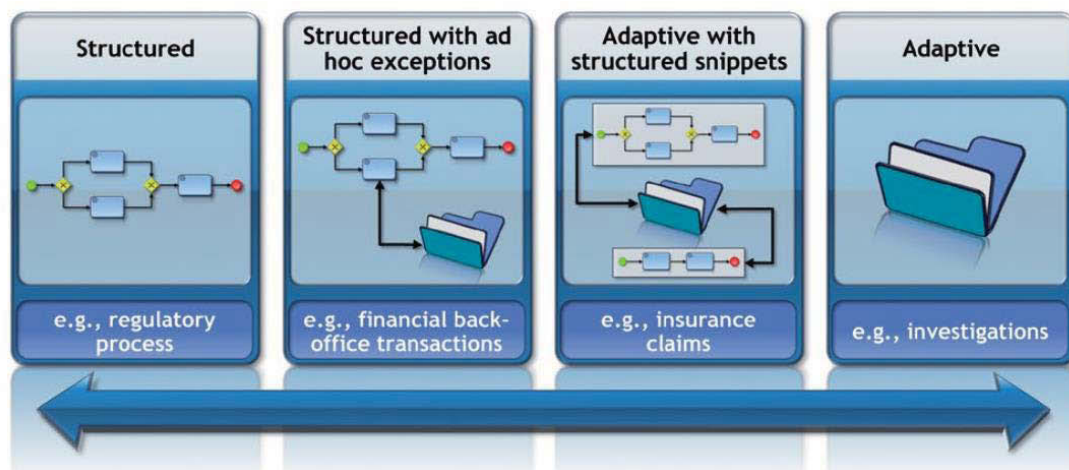


Figure 1-1: The spectrum of business processes [Kemsley 2011]

As illustrated in the process spectrum given in Figure 1-1 [Kemsley 2011], on the left-hand side are highly structured processes where the traditional BPM approach is adopted to improve their operational performance. On the right-hand extreme of the

spectrum are unstructured processes where case management is considered to be a promising solution as a process management approach (and processes positioning in the middle of the spectrum require a mix between BPM and case management approaches). In addition to this, compared with BPM approach case management solutions are fundamentally different for the following aspects:

- Processes in BPM are predictable and repeatable, while case management processes are unpredictable, and have a low level of repeatability.
- BPM is focusing on the process and everything else necessary to complete the process; whereas case management focuses on the data of the Case, as well as other factors needed to achieve its goal.
- In BPM, workers solve their problems by following a well-established process model beforehand: the concept of adaptation of the process does not exist. Knowledge workers accomplish their work by runtime planning and executing as the Case proceeds, and additional activities will be added if necessary.
- BPM rarely requires human participants' decisions, while case management requires case workers involvement to make decisions.

Generally, a Case can be categorized as one of the following three types: *mass cases*, *regular cases*, and *special cases* [Rooze et al. 2007]. *Mass cases* are cases that can be controlled in a completely automated manner. The processes within *mass cases* are highly structured and predicted, and traditional process techniques (such as BPM) are adequate management approaches. *Mass cases* lay in the left-hand extreme of the process spectrum given above. *Regular cases* refer to cases that repeat over time (and they are positioning in the middle area of that process spectrum). However, the solution of each case instance is not exactly the same. As experience grows, common solution patterns (also known as templates) can be identified and formalized. Consequently, similar cases can be managed in a more rigorous and repeatable manner. Knowledge workers control how *regular cases* evolve and complete. In addition, the ways to complete cases are also constrained by other factors (business rules, availability of activities to choose from, etc.) that are defined in the templates. With respect to *special cases* (which are at the right-hand side of the process spectrum), case workers own the whole freedom on deciding how a Case evolves, based on their experience and their evaluation and judgment on incoming information. Solutions to *special cases* are discovered in a complete ad-hoc manner. Generally speaking, organizations start with managing *special cases*, with little suggestion to be referred. As time goes by, similar cases repeat, and recurring patterns can be revealed and summarized as templates by business analysts and modeling experts. As a result, *special cases* become *regular* ones. This becomes the practice of case management. Nowadays, organizations are more interested in exploring solutions on how to assist

Case workers in managing *regular cases*, such as [Man 2009b] [Clair & Moore 2009] [Clair & Miers 2014] and [Zhu et al. 2013].

1.1.2 Challenges

Among all the process management approaches, BPM receives the most attention of the business community in recent years. BPM is based on the scientific management principle proposed by Taylorist, in which the idea is to standardize rigid process models then execute these model for many times. The assumption behind BPM solutions is that the target processes are repeatable and predictable. However, due to its unique features of adaptability, unpredictability and non-repeatability, case management cannot take advantages from existing traditional BPM solutions. Knowledge workers need a type of support that allows them to do their work in an adaptive manner, instead of strictly following what have been defined in advance. Making changes to respond ad-hoc events when a case is being executed, anticipating critical problems, sharing information and making decisions collaboratively, etc., are essential capabilities to knowledge workers. From this point, BPM systems are inadequate and not suitable for governing case management processes.

Case management represents a new generation of process management technologies in terms of effectiveness, efficiency and adaptability. However, it lacks supporting methods and software tools. One essential aspect is case management process modeling. There are lots of benefits from modeling processes in the business domain, such as increasing the understanding of processes, visualizing the detailed components of process, identifying hidden risks, and so forth. Therefore, how to model case processes considering its flexible and adaptive characteristics becomes the starting point for researchers and analysts in the business community. This includes the practice of defining formal case management related concepts (e.g., case, case file, and task), specify the conceptual relationships between case management concepts, and nail down the graphical modeling notations, etc. Over the years, several case management modeling techniques have been proposed. Most noticeable ones include *Business Artifact* [Nigam & Caswell 2003], *Case Handling* [Aalst et al. 2005], *Cordys* [Man 2009b], *Guard-Stage-Milestone* [Hull et al. 2010], *IBM Case Manager* [Zhu et al. 2013], etc. Unfortunately, their proposed solutions do not really fill the gap of case management process modeling since most vendors come with their BPM solutions with only few added features and they claim that they offer suitable systems for case management.

In order to formally standardize the modeling practices, OMG (Object Management Group) offered a case management process modeling specification

(version 1.0) named CMMN (Case Management Model and Notation) in May 2014 [OMG 2014]. CMMN incorporates current case management research contributions (such as the *Business Artifact* concept, the *Case Handling* paradigm, and the *Guard-Stage-Milestone* approach), and defines an executable visual modeling language for case management processes. However, only a few countable discussions regarding CMMN can be found in literature since the modeling specification was released. Some BPM system vendors implement a case model editor on the basis of CMMN within their BPM systems. But with such case model editors knowledge workers are not able to deeply analyze the underlying processes within their work since what they have are merely static CMMN diagrams.

Regardless of the type of processes to be managed, the ultimate goal of any process management approach is to help organizations achieve their financial goal through improving operational performance. Case management is not exceptional at this point. Key process improvement theories such as *Lean*, *Six Sigma*, and *Business Process Re-engineering (BPR)* have illustrated their great profound influence in terms of process improvement in many areas. However, these theories have focused exclusively upon repeatable type of work. How to manage and finally improve knowledge work process performance was largely ignored in the past. This is no wonder since case management did not gain much attention from the business community until recent years. As far as our knowledge the efforts have been done only in the case management process modeling aspect, not to mention the contributions from the perspective of case management process improvement, such as adopting the Model and Simulation (M&S) technique to analyze case models dynamically, using formal methods to do formal analysis to case models, or embedding process discovery technique to explore case models automatically.

1.2 Objectives

To overcome the presented challenges and make contributions in the field of case management, we present here our case management solution in this thesis. The essential objective is to establish a system-theoretic case management framework with which knowledge workers are able to auto-discover and manipulate case models, identify problems (bottleneck, waste, etc.) hidden in processes, and predict operational performance of the system. Ultimately, the original case management process models will be improved. We offer the possibility for knowledge workers (both case managers and case workers) to managing their cases, regardless of the type of the case (i.e., special cases and regular cases). From constructing and analyzing case models to improve those models, our case management framework is expected to provide sufficient supports to manage cases throughout their whole lifecycle, from

starting a case instance to its completion. The detailed objectives are:

1. *Case Model Discovery*
It refers to the automatic establishment of case models from raw data. Instead of constructing case models manually from scratch, case models can be extracted automatically, which eases the modeling effort of knowledge workers, especially when they are not experts in modeling.
2. *Case Modeling*
It refers to the construction and manipulation of case models, where a model represents the real system knowledge workers are interested in. By modeling cases, knowledge workers are able to study and analyze the systems of interest in an abstract manner.
3. *Case Model Analysis*
It refers to the detailed study and examination of case models. Through model analysis, case managers can obtain a deeper view on their work. The analysis practices can be classified into two categories: static model analysis, and dynamic model analysis. The former refers to the analysis performed without executing models, e.g., the derivation of properties before the case is enacted, while the latter refers to the model analysis in a dynamic manner through simulation.
4. *Case Model Improvement*
It refers to a set of approaches and tools used by case managers to enhance their process operational performance. Generally, after understanding the existing case models and obtaining detailed analysis results, constructing a to-be model and then testing if the predicted performance is accepted or not will be the next step.
5. *Case Model Enactment*
It refers to the real time execution of a case scenario, with case workers in the loop.

Different techniques are adopted in our solution for different purposes: *process discovery* is used to automatically construct case models; *BPI techniques*, for instance, BPR, Lean, Theory of Constraints (TOC), etc., are used to analyze and optimize case model. Process Tree [Schunselaar et al. 2014] is selected as our choice to formalize and graphically present discovered case models, and CMMN is considered as the formalism for presenting and constructing case models. In addition, we adopt the HiLLS (High Level Language for Systems Specification) formalism we have defined [Maiga 2015] [Maiga et al. 2015] [Aliyu et al. 2015] to conciliate usability, simulation ability and formal analysis capabilities together in order to analyze case models from different perspectives. Dynamic model analysis is enabled by DEVS (Discrete-Event System Specification) formalism, static model analysis is provided by formal methods,

and model enactment is given by the implementation of an object-oriented specification of the case. To bridge the gaps between different modeling specifications and ensure consistency between them, i.e., Process Tree, CMMN and HiLLS, we use model transformation, a core concept in Model-Driven Engineering (MDE).

1.3 Contributions

The contributions of this thesis can be summarized as follows:

1. A mapping between the de-facto case management modeling standard and a multi-purpose system modeling formalism.
2. An algorithm to automatically generate an experimental frame (EF) for each case model, i.e., a specification of the set of components required to run experimentations on the case model and derive dynamic properties.
3. An algorithm for the dynamic evaluation of process performance metrics from different perspectives.
4. A simulation-based methodology for predicting the implications of changes made in case models.
5. A connection between formal process discovery models and case models conforming to the de-facto case management modeling standard.

1.4 Outline of the thesis

The remainder of this document is structured as follows. In chapter 2, we will introduce and review related works, including process modeling in business domain, case management process modeling, and case management systems. In chapter 3, we will explain our research background, including the introduction of related modeling specifications (Process Tree, CMMN, and HiLLS) as system modeling languages, as well as the techniques and technologies adopted in our work (process discovery, process improvement theories and approaches, model transformation techniques, etc.). Our framework will be presented in chapter 4, where we will explain how case models are discovered, analyzed and improved within this framework. A case study will be given and discussed in Chapter 5 to illustrate how knowledge workers can manage their cases from different perspectives. We will finally make in Chapter 6 a general conclusion followed with research perspectives.

2.RELATED WORKS

2.1 Introduction

In this chapter, we present the state of the art in case management modeling approaches and systems, as well as the improvement efforts made for case management processes. Though case management is a relatively unfamiliar term, it has gained more and more attention steadily over years, and enterprises have shown their own understanding and supporting tools in this new area. Therefore, we will present and discuss all the approaches developed for supporting case management, including the process improvement attempts. Moreover, we will review the work that has been done using process discovery technique in controlling business processes. Regarding the proposed approaches for case management modeling in the literature, we classify them into four categories on the basis of which aspect they offer for knowledge workers to analyze and manage case models:

- Activity-based Modeling Solutions (which focus on the aspect of control-flow modeling).
- Information-based Modeling Solutions (which focus on the aspect of information-flow modeling).
- Communication-based Modeling Solutions (which capture the communications and collaborations between people).
- Hybrid Activity and Information Modeling Solutions (which emphasize on establishing both the process model and information model when modeling cases).

2.2 Case Management Modeling Solutions

Case management emerged as a process management solution for knowledge workers who require a high degree of flexibility in controlling their knowledge work processes. It overcomes the limitations of traditional BPM solutions which restrict their usages in governing and improving highly structured, highly repeated manufacturing processes. In order to gain benefits from using the case management approach, organizations have started to explore novel ideas and approaches that can provide case workers adequate supports for managing cases from both perspectives of process and information, in an adaptive way. Some vendors also proposed their commercial case management applications based on their own understanding of case management. In order to standardize case management modeling practice, the OMG requested a proposal for case management process modeling in 2010, and ten major companies and organizations were involved in the development process (BizAgi Limited, Cordys Nederland BV, etc.). The final response of this collaborative work

was the case management modeling specification entitled “Case Management Model and Notation (CMMN)” [OMG 2014]. Until now this specification is still under development and refinement, but people have reached an agreement to the core concepts, the modeling notations and the execution semantics [Marin et al. 2012]. It has been progressively accepted by the business community. Before explaining the CMMN specification which we adopt for case modeling in this thesis, we will trace and review the efforts made for developing management approaches for knowledge work. Main concerns are the definition of case management related concepts (such as case, case folder) and the attempts of case management modeling, regarding both the information and processes involved.

In the literature, the modeling paradigms are basically categorized as activity-based (in which case related activities are modeled as the primary concern), information-based (in which the flow of information is constructed), and communication-based (in which the interactions of knowledge workers drive the case to progress and are modeled as the primary means) [Wang & Kumar 2005] [Man 2009a]. In addition to these three major case modeling approaches, we add one more hybrid modeling paradigm where activity and information are both described in the process model and information model of a case, respectively (i.e., both the process and information are treated as the key constructs for case modeling). Modeling attempts falling in the fourth category are further divided depending on if the process model is merged with the information model or not.

2.2.1 Activity-based Modeling

[Mayer 1992] defined a family of *IDEF* (Integrated computer-aided manufacturing DEFinition) approaches for the purpose of increasing manufacturing performance through the aide of computer technology. Within that series of techniques, *IDEF1* (also known as the Information Modeling Method) was developed to describe all the information necessary for organizations to accomplish their financial objectives. Generally, an *IDEF1* information model identifies the key information controlled by the organization, as well as issues caused by the lack of well managing the key information. The structure of the information is modeled in *IDEF1* models, in which: a Real-World Object is defined to represent an object in reality; an Entity is used to describe a piece of information; an Attribute is defined to represent a property of an Entity; and a Relation concept is used to describe the associations between Entities.

[Manolescu & Johnson 2001] proposed a new workflow architecture named *Micro-workflow*. In their workflow management solution, they adopted the

object-oriented paradigm so that the gap between workflow systems with different requirements is bridged. In addition, their *Micro-workflow* solution offers a reusable architecture with which users are able to configure a model to the new requirements.

[Kaan et al. 2005] proposed a graphical modeling language for case management. Concepts are embedded into traditional BPM modeling paradigm, where the evaluation of case information has an influence on executing activities. The visualization of the modeling language is inspired by the *Venn-diagram*, a diagram showing the logical relations between different sets.

[Strahonja 2007] suggested to model case procedures (criminal case procedures, for example) through UML state machines in the domain of legislation. It focuses on the dynamics of the legal system, and the limitations of automating the verification and validation of anomalies in legislation models have been discussed as well.

[Hull et al. 2010] informally proposed a business operation modeling approach using business entities and the *Guard-Stage-Milestone* (GSM) approach. It is more declarative than most finite state machine variants in terms of expressing the lifecycles for data-centric business processes. The operational semantics are defined using the ECA rule, which leads the possibilities for formal verification and reasoning.

Due to the fact that case management is information-sensitive and information-driven, the modeling approaches which consider only case activities are not adequate for modeling case management processes. Even in [Kaan et al. 2005] activities are controlled by rules evaluating the case information, the flow of information is not explicitly given. Moreover, such diagrams can be difficult for knowledge workers to read if a large amount of rules is defined. Other approaches do not consider much about the strong linkage between processes and information involved when modeling cases, and the modeling approach given in [Strahonja 2007] is only for the legislation area. They are more suitable for routine work rather than knowledge work. In addition, as modeling approaches they do not have formal definitions in terms of syntax (both abstract and concrete) and semantics.

2.2.2 Information-based Modeling

IDEF3 is another approach from the *IDEF* family, which is entitled as Process Description Capture Method [Mayer et al. 1992]. Unlike *IDEF1* which focuses on the information flow, *IDEF3* captures the knowledge of organizations from the behavior perspective: it was developed specifically for modeling descriptive activities. Within an *IDEF3* model, activities are summarized in the Process Flow Description, and the

lifecycles of objects are defined in the Object State Transition Description.

[Nigam & Caswell 2003] proposed to use the concept of *Business Artifact* (also known as *Business Record*) to capture case data. A *Business Artifact* refers to an entity used to record concrete, identifiable, self-describing and indivisible business information. Their approach was inspired by *IDEF0*, one of the *IDEF* family approaches targeting on system function modeling. The lifecycle of a *Business Artifact* is defined through two constructs called tasks and repositories, where the former defines the place a function applies, and the latter offer means for archiving business information. This artifact-centric thinking in the business area renders a flexible representation of business models, and offers the ability to control changes and system implementation.

The modeling approaches we reviewed in this section focus solely on the information flow of the case model. The evaluation of information captured measures if knowledge workers are on their way to achieve their business targets. However, diagrams modeled by *IDEF3* can be too complex for people to understand. Moreover, these information-based modeling languages lack formal definitions of their abstract syntaxes and formal descriptions of their execution semantics. [Mayer et al. 1992] informally offered the concrete syntax of *IDEF3*; but in [Nigam & Caswell 2003] no noticeable graphical modeling notations are given. In addition, without the control flow of activities it is hard to know which information triggers which task, or which activity yields which information. Thus, the dynamic views of cases are missing: they are more like static analysis tools for knowledge workers.

2.2.3 Communication-based Modeling

[Kumaran et al. 2003] formalized an XML-based programming language for constructing *Adaptive Documents* (ADoc) and *Collaboration* models, where *ADoc* and *Collaborations* are the new modeling artifacts proposed. This new modeling language adopts object-oriented technology, and it mainly focuses on describing the information flow, as well as the collaboration aspect of systems: a set of atomic models function together to achieve organizations business objectives.

[Ould 2005] and [Harrison-Broninski 2005] proposed *Role Activity Diagram* (RAD), a process management technique mainly aimed at modeling human communication and collaboration. *RAD* was developed for enabling communication-based process modeling. It focuses on the “role” of human and how the “role” is defined within related activities. It is considered as the most known process modeling approach aiming at describing communications between people.

There exists a graphical representation for *RAD* models as well.

[Weigand 2005] proposed a communication-based process management application named *DEMO*. It is one of the most prominent applications built on the basis of the Language-Action-Perspective principle, in which business goals are reached by human communications [Kethers & Schoop 2000]. In *DEMO*, the basic modeling unit is a “speech act”, a minimal action of human in terms of communication (e.g., request, accept). The graphical representation is defined, but human tasks involved are not considered in *DEMO* models.

[Swenson 2014] proposed *Cognoscenti*, a free open source experimental collaborative and adaptive platform for experiencing with case management. The main idea of this environment is to establish a *Project Exchange Protocol* for case management. This way, different systems (i.e., BPM systems and case management systems) are able to exchange information when they are integrated to work collaboratively. It offers a set of basic capabilities that case workers require for managing complex, unpredictable work such as tracking documents, exchanging notes, assigning roles.

Collaboration is another key feature of case management, and these approaches we mentioned in this section put their focuses on this important factor. However, as modeling approaches *ADoc*, *RAD* and *DEMO* only have informal graphical representations given in the literature; their abstract syntaxes and how their models execute in reality are not given. The *Cognoscenti* project is an experimental framework providing a means to work collaboratively, but it is not meant to establish models when dealing with cases. Moreover, these communication models do not link to the case information, or the activity control flow. Therefore, the control on data and tasks is not possible to specify in such models. [Weigand 2005] criticized that such communication-based models should be considered as a complementary view when modeling processes, instead of being the primary construct. In addition, the details given in *ADoc*, *RAD* and *DEMO* models are generally not precise enough for knowledge workers to have a deeper view of their work. When it comes with larger scale business problems, the readability of these models lowers due to a large amount of arrows and icons.

2.2.4 Hybrid Activity and Information Modeling

Due to the unique features of case management, it is intuitive to view data as the first-class object when modeling cases. However, in order to automate knowledge work processes the control flow of activities is as important as the information flow:

on one hand, information drives certain tasks to execute by knowledge workers; on the other hand, tasks also yield information which should be evaluated by knowledge workers in order to make decisions. It makes sense to claim that information and activities function together to achieve business goals in case management. In recent years, many business vendors showed their interests in integrating an information flow into their business process products in order to support case management, such as [Aalst et al. 2005], [Wang & Kumar 2005], [Bhattacharya et al. 2007], [Vanderfeesten et al. 2009], [Man 2009b], [Künzle & Reichert 2011], [Ajay 2013], [Newgen 2013], [Pega 2013], and [Zhu et al. 2013]. A noticeable difference between their applications is that [Wang & Kumar 2005] and [Künzle & Reichert 2011] merge the control flow of activities together with the information flow in case models, and other approaches have the opposite solution: they separate information models from process models.

[Aalst et al. 2005] proposed a process management modeling approach called *Case Handling*. It is a new modeling paradigm for process management towards flexible and knowledge-intensive business processes. Both information and processes are considered as the first-class factors in *Case Handling* and they are constructed using one uniform modeling paradigm. Metamodels are given to specify the key constructs used in modeling, at different layers. The system dynamics is defined by finite state transition diagrams combined with the Event Condition Action (ECA) rules. [Vanderfeesten et al. 2009] compared the two *Case Handling* systems: *FLOWer* and *Activity Manager*, and discussed to which extent these systems support a product-based workflow design from the aspects of flexibility and adaptability. They argued that the workflow design should be based on products, rather than activities. In such a workflow, each case is viewed as the “product” being manufactured, and the data elements are considered as “parts” assembled into the case when executing tasks.

[Wang & Kumar 2005] proposed the concept of *Document-driven Workflow*, in which the system has no explicit control flow established. In their workflow framework, the control flow and information flow are mixed in one diagram: processes are defined as a set of business documents, activities, and connectors, and it is the receipt of a document that triggers the execution of related processes. This results in a flexible modeling framework for dealing with ad-hoc tasks since users only need to specify what information a task will receive or produce.

[Bhattacharya et al. 2007] reviewed the *Artifact-Centered Operational Modeling* (ACOM) language, which is known as a language for modeling cases in an artifact-thinking manner. *ACOM* was targeted on the information artifacts flowing within the processes, and business operation models are constructed at different levels. It used the UML state machine technique to capture the behavior of business artifacts.

[Man 2009b] introduced their *Cordys* case management modeling product. New analysis techniques and different notations are adopted in this approach. The information model is constructed using case files with properties, and the behavior model is composed of activities and tasks (which is called activity cluster). The lifecycles of modeling elements are specified by using UML-based state machine with additional rules. It also offers the dynamic planning function so that knowledge workers can manage ad-hoc tasks at run-time.

[Künzle & Reichert 2011] proposed *PHILharmonicFlows*, an object-aware process management framework in which processes and data are strongly merged together. They take both the object behavior and the object interactions into account. In addition, data is integrated with processes so that processes are executed in a data-driven manner. The lifecycle is given by finite state machine, and the processes are modeled at different levels of abstraction (i.e., *Macro* and *Micro* levels).

[Ajay 2013] incorporated a case management add-on in their commercial Oracle BPM Suite. In addition to the control of structured business processes given in their original BPM product, this additional module offers a modeling capability for unpredictable workflows, especially on the flexibility perspective in aspects of case flows, user interfaces, work assignments, and enforcement of business policies. Information is organized in the content management system, and knowledge workers with specific roles can assess to the documents they need when necessary. Case activities and events models are also specified as the control flow in case models. The linkages between process models and the related information models are established as well. The capability of ad-hoc tasks management is enabled.

[Newgen 2013] proposed their commercial case management product for helping organizations control the costs and risks associated with unstructured business processes. The tool is built on a BPM platform, and it mainly focuses on three application areas: investigation, service request, and incident management. It offers knowledge workers capabilities of modeling case information and case tasks, reusing existing case templates, and monitoring executing activities. The lifecycle of a case is informally given by graphs.

[Pega 2013] offered *Pega Dynamic Case Management*, a commercial case management solution to automate knowledge work in order to improve the efficiency of systems in today's fast-changing business environment. It combines together all the people and documents required in one place, and information can be delivered automatically to a right person. It is built for multiple channels (e.g., laptops, tablets, and smartphones) so that people can work with the system from different places. It is considered as one of the most powerful case management tools from the perspective

of design-time and runtime modeling supports it offers [Clair & Miers 2014].

[Zhu et al. 2013] developed their commercial case management tool named *IBM Case Manager*. It is built on top of the *GSM* formalism, and the information model is defined as a content management repository. The actions placed on the files (e.g., create a document, modify a report) within that repository will trigger events, where events are evaluated under various conditions in order to execute tasks defined within stages.

Among all the products given in this section, [Wang & Kumar 2005] and [Künzle & Reichert 2011] mix the activities and information involved in one place. A possible drawback of this type of merge is that when it comes to complex business issues with a large amount of data-dependencies the process models will be too complicated to read. Other commercial case management products have a similar feature except the *Cordys* system mentioned in [Man 2009b]: the business vendors come with their BPM solutions with added flexible and dynamic functionalities and they claim that they offer suitable systems for case management. Each of them provides a means to model case information and case process. However, the concepts and the logical relations between key modeling constructs they defined are unique in their solutions, a common understanding and agreement is missing. In addition, the run-time altering of plans is only noticeable in [Man 2009b] and [Pega 2013], which is another core requirement when developing case management systems. From the modeling perspective, none of them is established on the basis of CMMN which specifies formal modeling syntaxes and semantics for case management modeling. However, they do have contributions for the development of CMMN since CMMN was the result of the collaborative efforts made by many organizations together. The concepts and techniques proposed in [Nigam & Caswell 2003], [Man 2009b], [Zhu et al. 2013], etc., can be found in the CMMN final report. In addition, the capabilities of static/dynamic model analysis, model improvement, model discovery, and enactment for case models are not presented in any of the published case management related literatures.

2.2.5 Summary

An overview and comparison of the literature review regarding the case management modeling approaches and systems considering which requirements they meet for modeling cases is presented in Table 2-1 below.

Table 2-1: Benchmark of case management solutions

Approach		Criteria of Case Management Solutions									
Category	Name	Abstract Syntax	Concrete Syntax	Operational Semantics	Run-time Planning	Information-flow	Control-flow	Simulation	Formal Analysis	Enactment	
Activity-based Solutions	<i>IDEF1</i>		x	x			x				
	<i>Micro-workflow</i>		x	x			x				
	A graphical modeling language on the basis of <i>Venn-diagram</i>		x				x				
	An approach to model case procedures		x	x			x				
	A GSM- and ECA-based modeling approach		x	x			x				
Information-based Solutions	<i>IDEF3</i>		x			x					
	<i>Business Artifact</i>	x			x	x				x	
Communication-based Solutions	<i>ADoc</i>		x	x							
	<i>RAD</i>		x	x							
	<i>DEMO</i>		x	x							
	<i>Cognoscenti</i>				x						
Hybrid Information & Activity	<i>Case Handling</i>	x	x	x		x	x				
	<i>Document-driven Workflow</i>		x	x		x	x				
	<i>ACOM</i>		x	x		x	x				
	<i>Cordys</i>	x	x	x	x	x	x				

Approach		Criteria of Case Management Solutions									
Category	Name	Abstract Syntax	Concrete Syntax	Operational Semantics	Run-time Planning	Information-flow	Control-flow	Simulation	Formal Analysis	Enactment	
Hybrid Information & Activity	<i>PHILharmonicFlows</i>	x	x	x		x	x				
	<i>Oracle BPM Suite</i>	x	x	x		x	x				
	<i>Newgen Case Management Framework</i>	x	x	x		x	x				
	<i>Pega Dynamic Case Management</i>	x	x	x	x	x	x				
	<i>IBM Case Manager</i>	x	x	x		x	x				

2.3 Process Improvement Practice in Knowledge Work

The ultimate goal of a case management process approach is to help organizations achieve their financial goal through improving their knowledge work operational performance. In [May 2005] [Staats & Upton 2011] and [Jones & Bell 2013], authors discussed how *lean* process improvement principles can be adopted in knowledge work theoretically. [Staats et al. 2011] proved the applicability of using *lean* theory in managing knowledge work processes by providing real-life evidence at an Indian software services company. More specifically, [Staats et al. 2011] suggested the following six principles regarding applying *lean* in knowledge work, which are borrowed from the classic *lean* theory: (1) waste should be identified and eliminated; (2) the work should be specified explicitly; (3) communications among workers should be established; (4) problems should be solved quickly and directly, with scientific methods; (5) managers and workers should realize that making their system *lean* is a long journey; and (6) a lean culture should be established through the whole organization. Similar suggestions were also given in [Staats et al. 2011] that are summarized from a real-world case study: problems should be identified in an early stage, and problems and solutions should be considered together.

Process discovery is the core application aspect in *process mining* [Castellanos et al. 2009]. The primary objective of process discovery is to help business analysts extract, understand and improve their business processes from facts (i.e., real data recorded at run-time). It is a novel discipline proposed years ago. However, organizations have shown a great deal of interests in this topic due to its abilities of automatically constructing process models from raw data and helping analysts gain insight into process potential problems. Some work relating to process discovery has been done in the business domain, and major contributions have been made from the perspective of discovering, analyzing and enhancing business process models constructed by BPMN (Business Process Model and Notation, the de facto standard for modeling traditional structured business processes involved in routine work), in the process mining framework named ProM [Verbeek et al. 2009]. [Aalst et al. 2007] proposed their process discovery algorithms in the business context, and explained the application of their algorithms on a basis of a real-life case study in the Dutch National Public Works Department, from different point of view (i.e., process, organizational, and case). [Kalenkova et al. 2014] illustrated their tooling support in ProM with respect to BPMN process models discovery, analysis, and enhancement. The support of BPMN standard bridges the gap between formal process discovery model formalisms (e.g., petri net, process tree) and the standard business process modeling formalism (i.e., BPMN). The solution they adopted was to establish conversion mechanisms which allowed the transformation from various formal process discovery models to BPMN models. In [Weerdt et al. 2014], the “BPMN Miner” was presented as a ProM plug-in to allow a bi-dimensional discovery of

business process models. It provided the ability to represent discovered BPMN models from both the control-flow point of view and the organizational point of view. This way, improvement can be made from analyzing BPMN models from two different perspectives. Based on this, [Conforti et al. 2016] proposed their approach to automatically discover business process models in a hierarchical manner. In such mined BPMN models, interrupting and non-interrupting boundary events and activity markers are considered.

However, by reviewing the literature we noticed that in terms of process improvement, all conclusions drawn above in [May 2005] [Staats & Upton 2011] [Staats et al. 2011] and [Jones & Bell 2013] are all abstract principles and suggestions: concrete approaches are missing. In other process improvement approaches such as *Six Sigma*, *TOC* and *BPR*, no discussions have been made in the domain of case management since people now are focusing on the standardization of general concepts and modeling notations of case management. Obviously, efforts will be put on case management process improvement very soon. However, currently case management lacks concrete solutions with respect to process improvement. On the other hand, in terms of process discovery we noticed that the contributions that have been made relate merely to structured business process (where BPMN is the de facto modeling standard). No noticeable “process discovery” contribution has been given in terms of unstructured business process (where CMMN is the de facto modeling standard).

2.4 Conclusion

We have presented in this chapter a literature review of the state of the art regarding case management modeling approaches and systems, and process improvement practices in case management. The case management modeling solutions were discussed under four categories: activity-based, information-based, communication-based, and hybrid activity and information based. We further summarized to which extent these approaches and systems support knowledge workers modeling processes and information involved in a case.

We noticed that as modeling languages for case management, most of the approaches come with a graphical representation, but they lack a formal description for their abstract syntax and operational semantics. It is reasonable to consider both the process and information as the first-class citizens when modeling cases, due to case management features, and knowledge workers should be able to cooperate with each other, in a collaborative manner. However, most of the proposed modeling languages focus on only one aspect.

We further observed that as commercial case management systems, most of them offer a case modeling language with syntaxes (both abstract and concrete) and

operational semantics defined. Moreover, they all consider that the information model is as crucial as the process model, even though the information-flow and the control flow are mixed together in [Wang & Kumar 2005] and [Künzle & Reichert 2011], and in other systems they have an explicit separation. However, the feature of collaboration is hardly noticeable in most of them, except in [Swenson 2014]. In addition, the empowerment of knowledge workers and run-time planning is another key requirement for case management supports, and can be found in only a few publications. None of the approaches were developed on the basis of CMMN. In terms of process simulation, process improvement and process discovery, not a lot of noticeable contributions can be found in the literature. Obviously, it is beneficial for knowledge workers and organizations with intensive knowledge work to have supports in all the aspects we mentioned above when conducting case management. This is what we try to provide in this thesis.

In the next chapter, we will present in detail the theoretical and technological background of our proposal. More specifically, we will explain the concepts and techniques we adopted in developing our case management solutions, including CMMN, Elements of System Modeling language, the System Theory, HiLLS, Process Discovery, Process Improvement Approaches, Model transformation and supporting languages.

3. Background

3.1 Introduction

In this chapter, we introduce the concepts and techniques we adopted in developing our case management framework. Essentially, the contributions of this thesis involve the management, analysis and improvement of case management processes, which relate to the transformation between different system modeling languages, as well as the usage of process management techniques in case management. This chapter presents all the concerning system modeling formalisms and process management techniques. In detail, we start with introducing the elements of modeling languages, followed by relevant system modeling formalism (CMMN in section 3.3, HiLLS in section 3.4 and process tree in section 3.6.1). Sections 3.5 and 3.6 present the process management techniques that are considered as the basis for managing and improving processes. Section 3.7 presents the technique used to manipulate the translations between different modeling languages, i.e., model transformation.

3.2 Definition of A Modeling Language

A modeling language helps people build models of systems under study. Basically, a model is a representation of the reality with certain relevant aspects [Seidewitz 2003]. The key reason of modeling systems is that, it is easier and less expensive to analyze the system of interest at the model layer (due to the fact that in reality it is usually too expensive, impractical, or even impossible to study the system of interest) [Maria 1997]. As given in Figure 3-1, a formal specification of a modeling language consists of an abstract syntax, one or a set of concrete syntaxes and semantic domains, the mappings between the abstract syntax and the concrete one(s) (each of which is denoted as M_{AC}), as well as the mappings between the abstract syntax and the semantic domain(s) (each of which is denoted as M_{AS}) [Kleppe 2008]. Generally, most languages specify only one concrete syntax and one semantic domain, as well as the mappings M_{AC} and M_{AS} . However, it is possible to have multiple definitions of them, each of which targets on one specific purpose. A general language specification can be expressed as a tuple, $L = \langle A, C, M_{AC}, S, M_{AS} \rangle$, such that:

- A refers to the abstract syntax.
- C refers to the concrete syntax.
- M_{AC} refers to the syntactic mapping from A to C .
- S refers to the semantic domain of the language.
- M_{AS} refers to the semantic mapping from A to S .

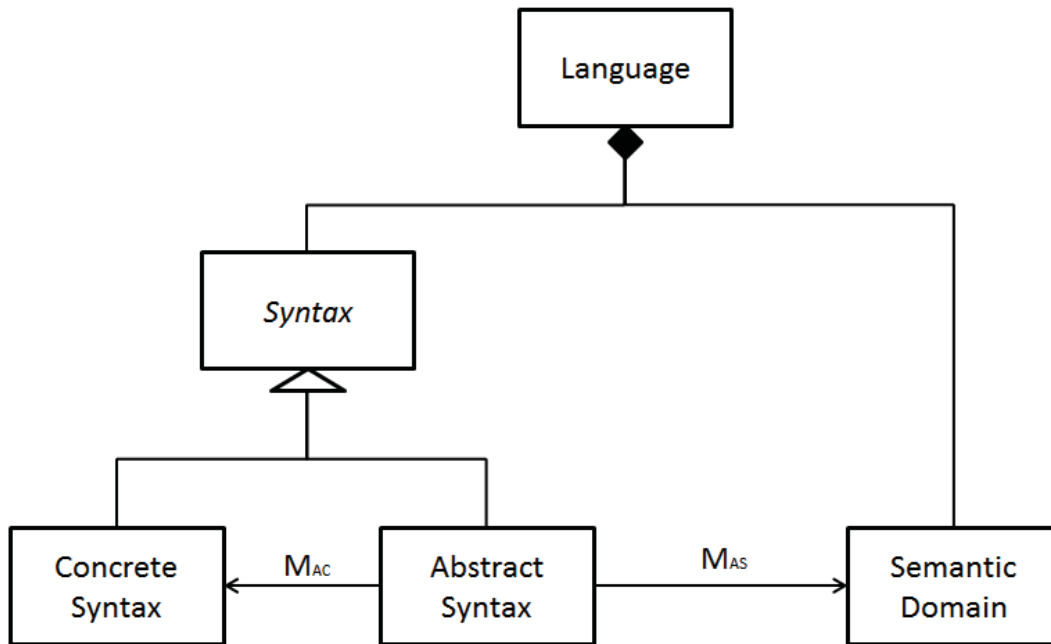


Figure 3-1: The definition of a modeling language specification

The abstract syntax of a language refers to a set of rules specifying the well-formed expressions of the abstract modeling constructs. It consists of the definition of modeling elements and the relations between them [Selic 2009]. In MDE, an abstract syntax of a modeling language is often specified by a metamodel: a model of a modeling language that defines the essential properties and features of modeling elements [Mellor 2004] [Clark et al. 2008]. Basically, a metamodel and an abstract syntax of a language have the same interpretation in MDE. A model is a representation of the system under study, and it always conforms to its metamodel.

Unlike the abstract syntax, the concrete syntax of a language specifies the notations used to describe the entities and their relations defined in the abstract syntax. It is used to offer human-readable presentations of models [Selic 2009]. The presentation of modeling elements can be textual, graphical, or a mixture of both. Regardless of the type of model presentation, the information rendered by the model does not change [Seidewitz 2003].

A syntax mapping, denoted as M_{AC} in Figure 3-1, assigns the concrete presentation of a modeling element with its definition defined in the abstract syntax. It bridges the abstract syntax with its concrete notations together, in an unambiguous manner. Therefore, when people see a symbol, for example, they will know what this symbol represents thanks to the syntax mapping. If a language has a group of concrete syntax defined for various purposes, a group of syntax mappings are required as well, each of which maps the abstract syntax to a concrete one.

The meaning of each modeling element is given by the semantics of the language. In particular, the operational semantics of a modeling language specifies the way the execution of models expressed in that language is carried on [Slonneger & Kurtz 1995]. It formalizes how the behavior traces will be generated by the model, at execution time. Moreover, the meaning should originate from a well-defined and well-understood area [Harel & Rumpe 2004]. This is called a semantic domain. A semantic domain offers the context under which a model is interpreted. The semantic mapping, as denoted by M_{AS} , specifies the relations between the modeling concepts defined in the abstract syntax and the semantic domain. Therefore, each notation of that language has an explicit meaning in a specific domain. For a multi-purpose system modeling language, as the one we adopted in this thesis, HiLLS, there are several semantic domains defined. Similar to the syntax mapping, each semantic domain requires a semantic mapping from the abstract syntax.

3.3 Case Management Model and Notation

In order to standardize case management process modeling practices, OMG released CMMN, a formal case management modeling specification in May 2014 [OMG 2014]. CMMN incorporates the latest case management research contributions proposed by business analysts and vendors. Typical ones include the *business artifact* concept [Nigam & Caswell 2003], the *case handling* paradigm [Aalst et al. 2005], the *GSM (Guard-Stage-Milestone)* approach [Hull et al. 2010], the *finite state machine* concept, etc.

In CMMN, the *case file model* is separated from the *case plan model*, where the former contains case information involved in resolving cases, and the latter depicts related activities to be done in order to achieve business goals. In addition, CMMN distinguishes the *design-time modeling phase* from the *run-time modeling phase*, where the latter is one key requirement in case management, as we have discussed. At the *design-time modeling phase*, an initial plan is established; while dynamic planning (e.g., adding ad-hoc tasks, modifying execution sequence of tasks) can be conducted at the *run-time modeling phase*. CMMN aims at providing case workers the ability to collaboratively manage and control their knowledge work in a flexible and adaptive manner. When modeling cases with CMMN, case workers are capable of constructing the main episode of the case on one hand. On the other hand, during execution with incoming information and gaining experience, they are able to alter the model to some extent [Grudzińska-Kuna 2013].

In the CMMN modeling standard, the abstract syntax (which defines a set of case management process modeling concepts and the relationships among them) is specified using metamodeling, and the concrete syntax (which defines the visible representations of the modeling elements) is defined by graphical notations. The *finite*

state machine concept is used to define the operational semantics of CMMN modeling elements. The completed definition of CMMN is written in [OMG 2014]. In this section, we briefly introduce the key modeling concepts and their relations as defined in the abstract syntax, their graphical notations, as well as their lifecycles defined as the operational semantics.

3.3.1 Abstract Syntax

We present an overview of the core concepts defined in the CMMN abstract syntax in this section, as summarized in Figure 3-2.

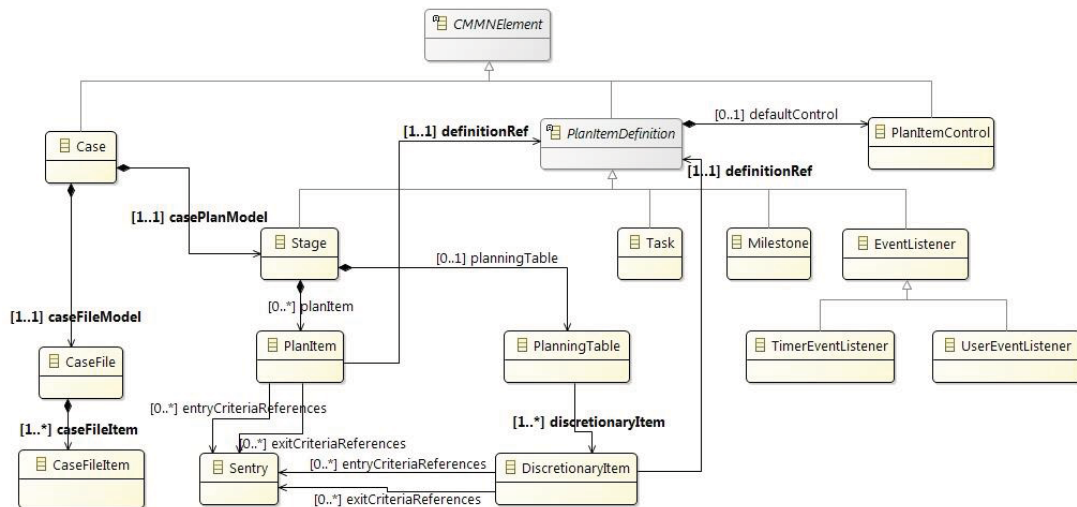


Figure 3-2: The core of the CMMN metamodel

The object *Case* is a top-level concept, in which all other elements constituting that *Case* are defined. Essentially, a *Case* contains a *caseFileModel* and a *casePlanModel*. The *caseFileModel* of a *Case* collects and records all information involved in solving a *Case* problem, and it is defined by a *CaseFile* object. A *CaseFile* object serves as an information container that consists of *CaseFileItems*, each of which represents a piece of information of any nature. The data structure of information ranges from a simple XML to a complex folder hierarchy structure. Information captured in *CaseFileItems* is used as a context for case workers to make decisions, e.g., raising events, evaluating conditions.

The *casePlanModel* of a *Case* consists of elements used for both design-time planning and run-time planning. On one hand, it contains elements that represent the initial plan of the *Case* (i.e., these activities must be done during execution). On the other hand, it also consists of elements that support the further evolution of the initial

plan, at execution time (i.e., these activities will or will not be done at run-time, according to case workers decisions). Essential elements for modeling the case plan are *Stage*, *Task*, *Milestone*, *UserEventListener*, and *TimerEventListener*.

- **Stage:** A *Stage* serves as a container grouping basic modeling elements. In particular, the outermost *Stage* defined within a *Case* model is considered as the *casePlanModel* of that *Case*.
- **Task:** A *Task* represents a unitary piece of work. In addition, a *Task* can be further specified as a *ProcessTask* (which is used to invoke another process), a *CaseTask* (which is defined to trigger the creation of another *Case*), and the most common one, a *HumanTask* (which contains things to be done by case workers).
- **Milestone:** A *Milestone* represents a business goal to be achieved. Different from *Tasks*, no concrete work is associated with *Milestones*; but they can be used to evaluate the progress of a *Case*.
- **UserEventListener:** A *UserEventListener* is defined to capture events that are raised by case workers. Such events are capable of impacting the proceeding of a *Case* directly. Therefore, case workers are able to directly interact with a *Case* through *UserEventListeners*.
- **TimerEventListener:** A *TimerEventListener* is used to catch a certain predefined elapse of time.

The items used for constructing the initial plan of a *Case* are called *PlanItems*, each of which refers to one essential modeling element (which might be a *Stage*, a *Task*, a *Milestone*, a *UserEventListener*, or a *TimerEventListener*). The connections and dependencies between *PlanItems* are expressed by *Sentries*. Each *Sentry* is defined as a guarding condition to activate or terminate a *PlanItem* (the one defined to activate a *PlanItem* is called the *entryCriteriaReference* of that *PlanItem*, while the one defined to terminate a *PlanItem* is named as the *exitCriteriaReference* of that *PlanItem*).

On the contrary, the items used for the further evolution of the initial plan of a *Case* at executing time are called *DiscretionaryItems*. Instances of *DiscretionaryItems* are planned and executed according to the discretion of case workers, and they are defined within the *PlanningTables* of *Stages*. Different from *PlanItems*, each *DiscretionaryItem* can only refer to either a *Stage* or a *Task*. Similar to *PlanItems*, dependencies and linkages between *DiscretionaryItems* are also specified using *Sentries*.

Moreover, each essential modeling element is constrained by the property rules which are specified in *PlanItemControl*. Aspects of control include:













- An element will be manually or automatically activated once its

entryCriteriaReferences are met. This is specified by the *ManualActivationRule*.

- Whether an element is obliged to be completed or not before the containing *Stage* becomes completed. This is captured by the *RequiredRule*.
- An element will be executed only once or many times. This is specified by the *RepetitionRule*.

3.3.2 Concrete Syntax

CMMN defines a set of notations used for depicting the essential modeling constructs. An overview of the concrete syntax of CMMN is summarized in Figure 3-3.

CaseFileItem	CasePlanModel	Stage	Task
			
Milestone	TEL ¹	UEL ²	Sentry
			
AutoComplete	ManualActivation	Required	Repetition
			

1: TEL refers to *TimerEventListener*.

2: UEL refers to *UserEventListener*.

Figure 3-3: An overview of the CMMN concrete syntax

As we can see from this summary table, regarding the case information model CMMN only offers a graphical notation (a “Document” shape) for *CaseFileItems* involved in a *Case*. The case behavior model is totally captured in the outermost *Stage* (which is referred as the *casePlanModel* of the *Case*), which is depicted by a rectangle shape with a small rectangle attached to the upper left indicating the name of the *Case*.

All the planning elements should be defined with the *casePlanModel*.

A *Stage* is depicted as a rectangle shape with angled corners. If that *Stage* is discretionary, i.e., there exists a *DiscretionaryItem* referring to this *Stage*, then that angled-corner rectangle shape is depicted by dashed lines, instead of solid one. A *Task* is depicted by a rectangle with rounded corners. Similarly, the notation for a discretionary *Task* is a rounded-corner rectangle shape with dashed lines. Only *Stage* and *Task* items can be referred by *DiscretionaryItems*. A simple half-rounded-end rectangle shape represents a *Milestone*. As to a *TimerEventListener*, it is depicted by a double-line circle inside which a “Clock” marker centers. If there is a “User” symbol marker centers inside the double-line circle, then it represents a *UserEventListener*. Dependencies between items are defined by *Sentries*. If a *Sentry* is specified as an *entryCriteriaReference* of an item, then it is depicted by a shallow “Diamond” shape. On the contrary, a *Sentry* is depicted by a solid “Diamond” shape if it is used as an *exitCriteriaReference* of an item. A dotted line is defined as the connector to establish the dependencies between certain elements.

In addition, a set of behavior decorators are defined to make the CMMN notations more expressive. The *AutoComplete* decorator is depicted by a small black square. If the *autoComplete* attribute of a *Stage* (it can be either the outermost *Stage* representing the *casePlanModel* of a *Case*, or simply just a *Stage* element) is set to “true,” then that decorator is attached to the bottom of the *Stage* notation. The *ManualActivation* decorator is depicted by a small triangle. If the *ManualActivationRule* attribute of a *Stage* or a *Task* element evaluates to “true,” then this decorator is added to the bottom of the element notation. The *Required* decorator is depicted by an “Exclamation” symbol, and it is added to the bottom of the element notion if the *RequiredRule* attribute of that item (*Stage*, *Task*, or *Milestone*) is set to “true.” The *Repetition* decorator is identical to the ASCII # shape, which is visible if the *RepetitionRule* of a *Stage* or a *Task* item is set to “true.”

3.3.3 Operational Semantics

The operational semantics of CMMN is given by the lifecycles of the essential modeling constructs and a set of their behavior property rules, where the lifecycles are defined using finite state machines. In particular, the majority of the operational semantics consists of the lifecycles of *CaseFileItem*, *CasePlanModel*, *Stage*, *Task*, *Milestone*, *TimerEventListener*, and *UserEventListener*, as summarized in Figure 3-4 below.

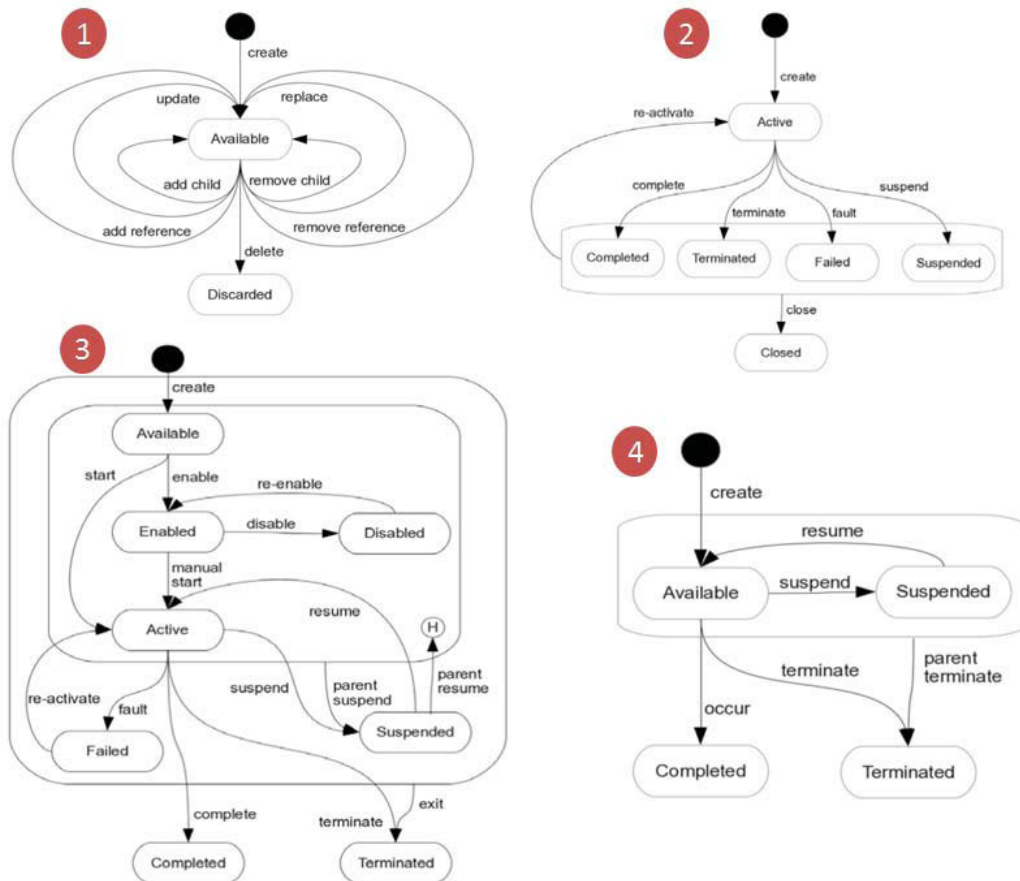


Figure 3-4: An overview of the lifecycles of CMMN essential modeling elements

The state machine with label “1” illustrates the lifecycle of a *CaseFileItem* instance. Basically, a *CaseFileItem* has only two states: Available and Discarded. A *CaseFileItem* in available indicates that the information represented by the *CaseFileItem* is available for case workers to use. If the piece of information is deleted and is not available any more, the corresponding *CaseFileItem* is in the discarded status. Events raised by case workers can influence the state transitions of a *CaseFileItem* instance, including create, addChild, removeChild, addReference, removeReference, update, and replace.

The state machine labeled with number “2” defines the lifecycle of the outermost *Stage* instance used as the *casePlanModel* of a *Case*. In fact, it represents the possible transitions of a *Case* instance. Therefore, its lifecycle is different from other *Stage* instances. A *Case* can be in status of Active (which indicates that the *Case* instance is executing), Suspended (which indicates that the *Case* instance is temporarily suspended by case workers), Completed (which indicates that all the required work defined with the *Case* has been done), Terminated (which indicates that the *Case* instance is terminated by case workers), Failed (which indicates that there is an

exception or software failure), and Closed (which indicates that no action is allowed in the *Case* instance). A *Case* transitions from one state to another when an event is raised by case workers.

A *Stage* and a *Task* share the same lifecycle, as the state machine with label “3.” In addition to the states (except the closed status) owned by a *Case*, a *Stage* or a *Task* instance has additional states including Available (which indicates that the *entryCriteriaReference* of that item is “false”), Enabled (which indicates that the item is waiting for case workers to active or disable it), and Disabled (which indicates that this instance will not be executed unless it transitions back to the Enabled state). All the transitions a *Stage* or a *Task* instance can undergo all illustrated in the figure above.

The last state machine labeled with number “4” specifies the lifecycle of a *Milestone* instance, a *TimerEventListener* instance, or a *UserEventListener* instance. They can transition among states including Available, Suspended, Completed and Terminated, when triggered by events (including create, suspend, terminate, occur, and resume).

3.3.4 Example

In this sub-section, a simple CMMN case model example, which is constructed on the basis of the example illustrated in [OMG 2014], is given in Figure 3-5. The entire *Case* model is named *Claims File Management*, and it consists of the following elements:

- One *CaseFileItem* (which is named as *Request*; each of which defines the information required for processing a *Case* instance).
- Three *Tasks* (which are named as Identify Responsibilities, Create Claims Notification, and Create Claim, respectively; each of the *Task* element refers to a unitary piece of work to be done).
- Two *DiscretionaryTasks* (which are called Change Responsibility and Request Missing Information; these *DiscretionaryTasks* are available and are ready to be executed at run-time).
- Two *Milestones* (which are Responsibilities Identified and Base Information Attached; each of which defines an achievable target that is used to evaluate the progress of the *Case* instance).
- One *UserEventListener* (which is named as Cancellation; it is defined to catch an event raised by case workers).
- One *TimerEventListener* (which is named as Deadline; this element is used to capture a predefined elapse of time).

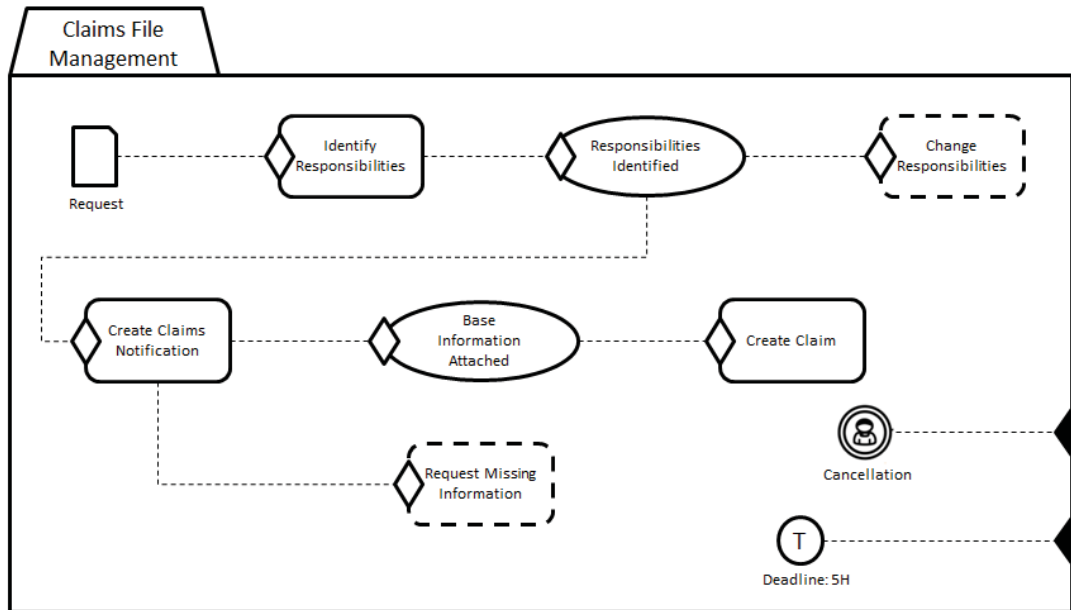


Figure 3-5: A CMMN case model

Each element introduced above has its own lifecycle defined as the operational semantics. The dotted line connectors between elements are used to specify their dependencies. The elements that are depicted by diamond shapes are *Sentries*, in which an entry criterion is depicted by a shallow diamond shape, and an exit criterion is depicted by a solid diamond shape. The *Sentry* associated with the *Milestone* Responsibilities Identified indicates that this *Milestone* cannot be completed until the *Task* Identify Responsibilities is completed. Likewise, the *Task* Identify Responsibilities cannot be in active state unless the *CaseFileItem* Request is available. The exit criterion bonding the *TimerEventListener* Deadline and the whole *Case* Claims File Management implies that once the deadline defined by Deadline is reached (in this example, the deadline is 5 hours after initialization), the whole *Case* will be terminated, as well as all its containing elements.

3.4 High-Level Language for System Specification

HiLLS (High Level Language for Systems Specification) is a system modeling language for constructing multi-analysis system models. It is developed on the basis of DDML (DEVS-Driven Modeling Language), a DEVS-based graphical modeling language for domain experts to facilitate their use of DEVS for the purpose of building system models [Maiga et al. 2012] [Ighoroje et al. 2012]. HiLLS helps domain experts create models of Discrete-Event Systems (DESSs) from various analysis perspectives, where different views of the systems are unified in one single HiLLS model.

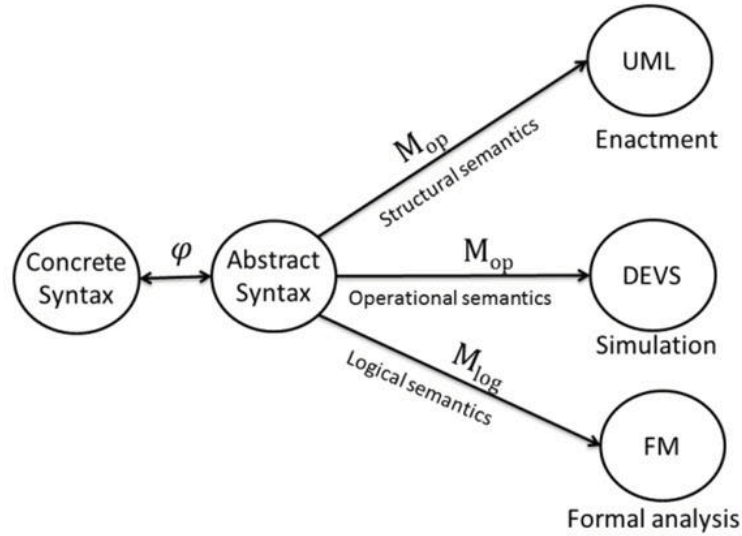


Figure 3-6: An overview of HiLLS

As a system modeling language, HiLLS can be formalized as $HiLLS = \langle A, C, M_{AC}, \{S_i\}, \{M_{AS_i}\} \rangle$, in which A is the abstract syntax, C is the concrete syntax, and M_{AC} is the syntax mapping from A to C , as we have introduced above in section 3.2. However, a special feature of HiLLS is that instead of having one single semantic domain defined, HiLLS specifies a family of semantic domains $\{S_i\}$ and a set of corresponding semantic mappings $\{M_{AS_i}\}$, each of which maps A to S_i (where $S_i \in \{S_i\}$). This way, HiLLS enables its multiple analysis capabilities including simulation, formal analysis, and system enactment. As detailed in Figure 3-6, HiLLS has a unified abstract syntax for specifying DESs models logically, where concepts borrowed from DEVS and Object-Z are integrated. This way, HiLLS offers a system behavior description in addition to the structural and logical system description given by Object-Z. Moreover, a concrete syntax is defined for graphically representing HiLLS models, which makes it easy for domain experts to learn, share and discuss modeled systems thanks to its high expressive power. The visual representations of HiLLS are inspired by UML class diagram, system control oriented transition diagrams, and Z schemas. Furthermore, HiLLS defines its various semantics domains for different purposes: adopting DEVS as its simulation semantics for system simulation, using Formal Method (FM) as its logical semantics for formal analysis and verification of system properties, and making use of UML as its enactment semantics for system enactment [Aliyu et al. 2015] [Maiga et al. 2015] [Maiga 2015]. In this section, we give a brief introduction to HiLLS.

3.4.1 Abstract Syntax

The complete metamodel of HiLLS is formalized and given in [Maiga 2015], and we show it here to illustrate how HiLLS merges and integrates concepts from different domains to provide the ability of analyzing systems from various views. In particular, we focus mainly on the system-theoretic concepts in this sub-section.

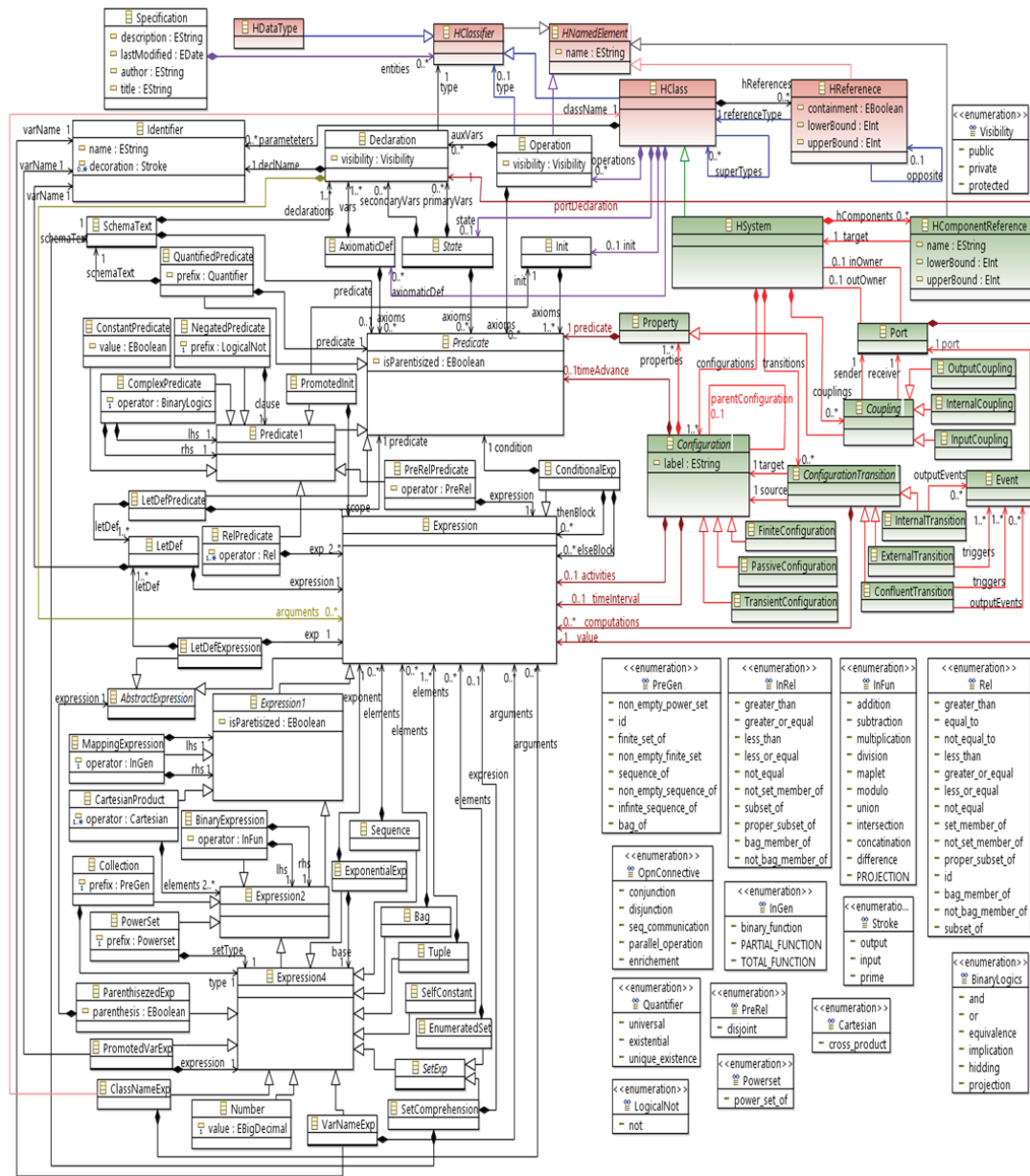


Figure 3-7: The HiLLS metamodel

As we can see in Figure 3-7, from the system theory point of view, the concept of

HSystem is defined (classes with a green background). A *HSystem* instance refers to a HiLLS system describing a DES, and it interacts with others through its input and output ports which are named *inputs* and *outputs*, respectively. The behavior aspect of a *HSystem* is captured by a transition diagram specifying by *Configurations* and *Transitions* between them. A *Configuration* represents a finite or an infinite set of a system's states which can be grouped together and be defined by the same predicate. In addition, each *Configuration* has a *sojournTime* associated, which indicates the maximum elapsed time for the system to be in any state of the *Configuration*. A *Configuration* is said to be transient if its *sojournTime* equals to zero (*sojournTime* = 0). If a *Configuration* is associated with an infinite *sojournTime*, then it is a passive one (*sojournTime* = $+\infty$). Otherwise, a *Configuration* with a finite *sojournTime* is said to be finite ($0 < \textit{sojournTime} < +\infty$).

A *HSystem* has at least one *Configuration* defined. Regarding *ConfigurationTransition*, a *HSystem* may contain zero or many if needed. If a transition occurs due to the expiration of the source *Configuration*'s *sojournTime*, then it is called an *InternalTransition*. If a transition is observed because of at least one input event is received before its *sojournTime* expires, then it is an *ExternalTransition*. However, if a *HSystem* receives at least one input event and its *sojournTime* expires at the same moment, then it is called a *ConfluentTransition*. *Configurations* and *ConfigurationTransitions* are defined together to specify the dynamic aspect of *HSystems*.

In addition, a *HSystem* may contain components, each of which is also a *HSystem* instance. This composition relation is enabled by *HComponentReference*. If a *HSystem* has components defined, then the exchange of information is defined through *couplings*. Three types of couplings are defined: *InputCouplings* (which is defined to let the *HSystem*'s components receive external events), *OutputCouplings* (which is established to allow the *HSystem*'s components to send out events), and *InternalCouplings* (which are created to enable the communications, i.e., the exchanges of events, between components of a *HSystem*).

From the perspective of logical reasoning, concepts (classes with a blank background) from Object-Z are preserved and reused, such as *Predicate*, *Operation*, *StateSchema*, *Declaration*, etc., which are adopted to specify Object-Z expressions and predicates used to define the static section of a *HSystem*.

In addition, basic object-oriented concepts (classes with a red background) are borrowed, such as *HClass* (which is the equivalence of Class in UML), *HAttribute* (which is the equivalence of Attribute in UML), *HReference* (which is the equivalence of Reference in UML), etc., as well as the relations between entities such as composition, inheritance, and so on. This enables the capability of enactment of systems under study.

3.4.2 Concrete Syntax

An overview of the HiLLS concrete syntax is given in Figure 3-8 below.

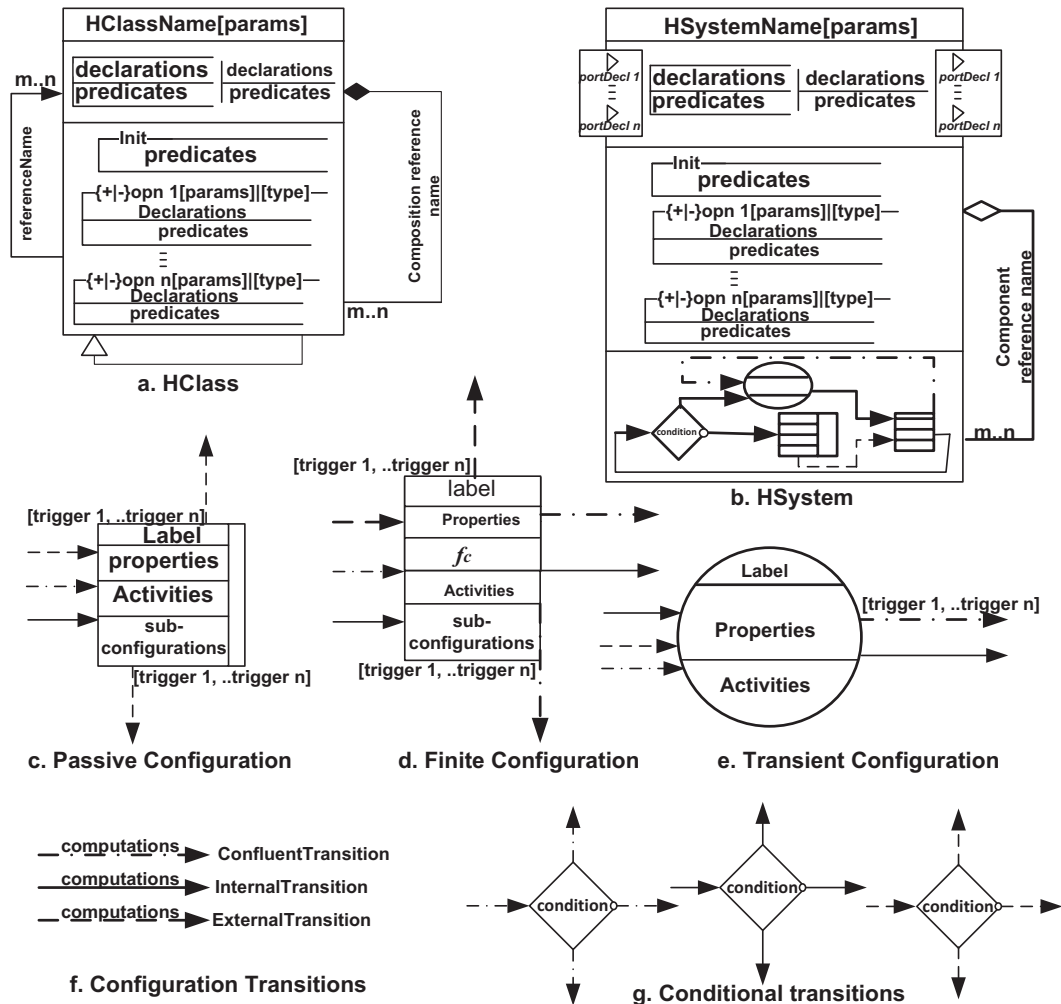


Figure 3-8: An overview of the HiLLS concrete syntax [Maiga 2015]

A *HClass* instance is depicted by a UML class symbol: a rectangle box with three compartments. Similar to the UML class graphical notation, the name of the *HClass* instance is placed in the first compartment, and attributes and operations are specified in the second and third compartment, respectively. The attributes and operations are formalized using the state schema and axiomatic definition borrowed from Object-Z. The references between *HClasses* (inheritance, reference, composition, etc.) are denoted using the same graphical notations as in UML.

The *HSystem* inherits from *HClass*, and its graphical notation extends that of *HClass*. In addition to the three compartments (each of which is reserved for name,

attributes and operations, respectively) like *HClass*, *HSystem* has a fourth one defining the system's behavior through *Configurations* and *ConfigurationTransitions*. Moreover, each of the input and output ports are depicted by a small triangle shape, which is contained by a rectangle representing the input and output interface, respectively. Input and output interfaces are attached to the *HSystem*.

A *FiniteConfiguration* is denoted by a rectangle shape with five compartments, each of which is reserved for its *label* (the name of the *Configuration*), *properties* (the predicates specifying the *Configuration* using Z schema), *sojournTime*, *activities* (the activities to perform), and *sub-configurations*, from top to bottom. Since a *PassiveConfiguration* has an infinite *sojournTime*, it does not require a compartment for representing its *sojournTime*. Therefore, a *PassiveConfiguration* is depicted by a four-compartment box (each of which is reserved for *label*, *properties*, *activities*, and *sub-configurations*) with a vertical stripe attached to the right side indicating its infinite lifespan. Different from these two types of configuration, the *TransientConfiguration* is depicted by a circle with three compartments, each of which is used for specifying *label*, *properties* and *activities*. Its round shape indicates its zero *sojournTime*.

The transitions between configurations are represented by lines with arrows. In particular, the *InternalTransition* is depicted by a solid line with an arrow pointing to the target configuration, and *outputEvents* and *computations* are labeled as well, if there exists any. On the contrary, an *ExternalTransition* is a dashed line with an arrow pointing to the target configuration, and *triggers* and *computations* can be specified, instead of *outputEvents* and *computations*. A *ConfluentTransition* is depicted by a dotted-dashed line with an arrow pointing to the target configuration, and *triggers*, *outputEvents* and *computations* are labeled if there exists any. In cases that if the transitions depends on a guarding condition, then the expression of that condition is specified in a diamond shape. Basically, this diamond shape receives flow, and depends on its evaluation result (true or false) different target configuration is chosen.

3.4.3 Semantics

As given in Figure 3-6 HiLLS maps its abstract syntax to various semantic domains for different purposes: DEVS for the purpose of system simulation, Z for the purpose of formal analysis and verification of system properties, and UML for system enactment. A complete definition of the various mappings is given in [Maiga 2015], and we briefly introduce its semantic domain for system simulation, DEVS, as well as the system theory, in order to ease the understanding of the case study provided later in this these.

Basically, in system theory a system is studied from the perspectives of *system*

structure and *system behavior*. As shown in Figure 3-9, *system structure* refers to a system's states, state transitions, and the mappings between states and outputs; while *system behavior* refers to the relationship between a system's input and output trajectories [Zeigler et al. 2000].

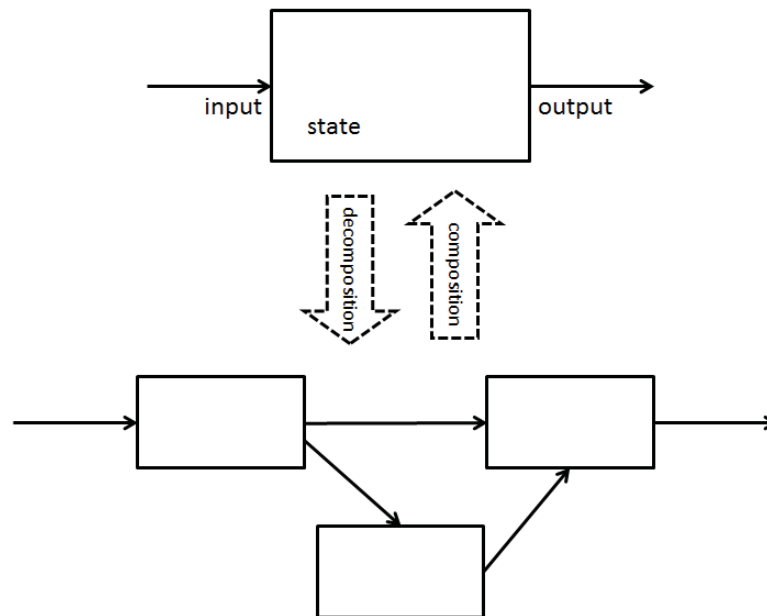


Figure 3-9: Basic system concepts

When specifying the structure of a system, the concept of *decomposition* defines how it can be broken down into several meaningful sub-systems, as indicated by the arrow downwards (with a label “decomposition” inside). On the contrary, *composition* specifies how components (each of which exists as a system) can be grouped together to constitute a larger and more complex system, as illustrated by the arrow upwards (with a label “composition” inside). Moreover, the *closure under coupling* property ensures that a larger system (which consists of many sub-systems) is also a system.

DEVS is the abbreviation for Discrete-EVent System Specification. It is developed on the basis of system theory principles, and it offers a mathematical formalism for modeling and analyzing DESs [Zeigler et al. 2000]. DEVS is originally known as Classic DEVS (CDEVS) which deals with sequential events, and later the Parallel DEVS (PDEVS) is proposed in [Chow & Zeigler 1994] with the ability to cope with parallel events introduced (we refer to PDEVS if DEVS is mentioned in the rest of this thesis). Before diving into DEVS formalism, we introduce the basic concepts regarding system theory which DEVS relies on.

On the basis of system theory, DEVS defines both system structure and system behavior. A DEVS atomic model describes the structure and behavior of a single unit

of a DES through inputs and outputs, a set of states, transitions between states, etc. A DEVS atomic model can be mathematically defined as

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \delta_{conf}, \lambda, ta \rangle, \text{ such that}$$

- $X = \{(p, v) \mid p \in IPort, v \in dom(p)\}$ is the input event set, where
 - $IPort$ refers to the input port set
- $Y = \{(q, v) \mid q \in OPort, v \in dom(q)\}$ is the output event set, where
 - $OPort$ refers to the output port set
- S is the state set
- $\delta_{int}: S \rightarrow S$ is the internal transition function
- $\delta_{ext}: Q \times X \rightarrow S$ is the external transition function, where
 - $Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$ is the total state set
 - e refers to the time elapsed since last state transition
- $\delta_{conf}: S \times X \rightarrow S$ is the confluent transition function (which solves the collision problem when the system is about to send out events and at the same time it receives external events)
- $\lambda: S \rightarrow Y$ is the output function
- $ta: S \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ is the time advance function

A DEVS coupled model describes a complex model that consists of atomic/coupled DEVS models. The hierarchical structure of a DEVS coupled model is defined. Mathematically, a DEVS coupled model can be specified as

$$M = \langle X, Y, D, \{M_d\}_{d \in D}, EIC, EOC, IC \rangle, \text{ where}$$

- X and Y have the same definitions as in the atomic model, respectively
- D is the set of names of components that M consists of
- M_d refers to a component model with a name d ;
- $EIC \subseteq \{((M, ip_M), (d, ip_d)) \mid ip_M \in IPorts_M, ip_d \in IPorts_d\}$ is the set of external input couplings (each of which is a coupling from an input of the coupled model to an input of its components)
- $EOC \subseteq \{((d, op_d), (M, op_M)) \mid op_M \in OPorts_M, op_d \in OPorts_d\}$ is the set of external output couplings (each of which is a coupling from the output of a component to an output of the coupled model)
- $IC \subseteq \{((a, op_a), (b, ip_b)) \mid op_a \in OPorts_a, ip_b \in IPorts_b\}$ is the set of internal couplings (each of which is a coupling from the output of a component to an input of another component)

In addition, simulation algorithms for DEVS models are defined so that any system formalized using DEVS can be simulated in a discrete time manner. Detailed simulation protocol is well defined and explained thoroughly in [Zeigler et al. 2000]. A non-exhaustive list of tools which enable DEVS M&S is maintained and updated in [Wainer 2013]. We make use of *SimStudio*, a platform-independent DEVS-based modeling and simulation environment (Traoré, 2008) (Touraille, Traoré, & Hill, 2011). *SimStudio* aims at conducting the practices of modeling, simulation, analysis and collaboration in a single unitary platform. It relies on the MDE approach, and it consists of various modules for different purposes (e.g., a Modeling module for constructing models, a Simulation module for running simulation, an Automation module for converting models from different formalisms into a unified DEVS representation, etc.). It is implemented as an extensible architecture, and additional modules which bring new functionalities can be integrated as plug-ins.

A generic DEVS-based M&S framework defines both entities and relationships between entities required for conducting the M&S practices [Zeigler et al. 2000]. As given in Figure 3-10, the basic entities with a generic M&S framework include: *source system*, *model*, *simulator*, and *experimental frame* (EF). They are defined to help gain better understandings about M&S studies, and have a better communication and discussion between modeling and/or simulation experts.

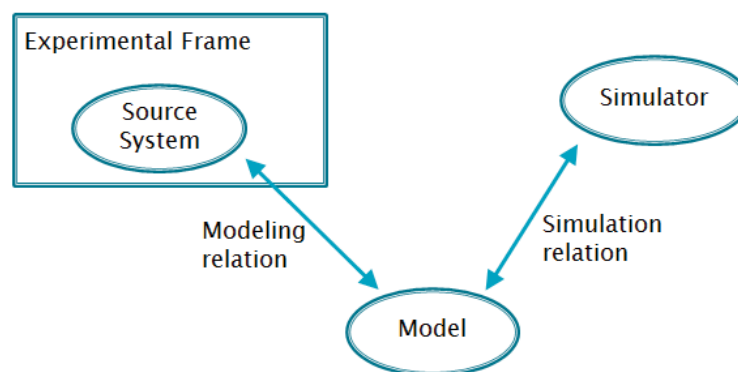


Figure 3-10: The DEVS-based M&S framework [Zeigler et al. 2000]

The *source system* refers to the real or virtual system that users are interested in. It is the source for the observable data, which is stored by the behavior database (where data are gathered either from observing the real system or from conducting system simulation experiments). The *model* commonly refers to a set of instructions, rules, equations or constraints for generating output trajectories from input ones. Generally, a *model* is viewed as a virtual representation of the *source system* we are studying. *Models* help experts and analysts study and understand the structure and behavior of systems from various aspects. The *simulator* is capable of executing a model to generate its behavior in a dynamic manner. It basically can refer to any type of a computation system: a single processor, a processor network, etc.

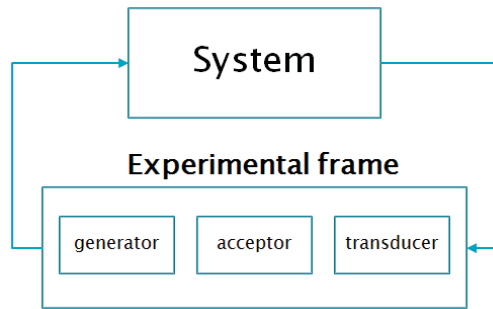


Figure 3-11: General components of an experimental frame [Zeigler et al. 2000]

The *EF* defines the conditions under which a system is experimented with. It is viewed as a system that interacts with the system of interest: it generates data that fed into the system of interest, and it collects and analyzes the results observed. Due to different research interests and objectives, one *EF* can apply to many *models*, and vice versa since *models* are separated from their contexts. In most cases, an *EF* consists of the following components, as shown in Figure 3-11: a *generator* (which generates input segments and feeds input segments into the system), an *acceptor* (which monitors simulation experiments to ensure that specific experimental conditions are met), and a *transducer* (which observes and analyzes output segments generated from the system). However, it is not mandatory to define all the three components when specifying an *EF*: one may need only a transducer in his own case. In other cases, an *acceptor* and/or a *generator* might be necessary. Users have their freedom to define necessary components they need.

3.4.4 Example

We give a taste of the HiLLS formalism through two simple examples presented below, where the first one is an atomic HiLLS model, and the second one is a coupled one.

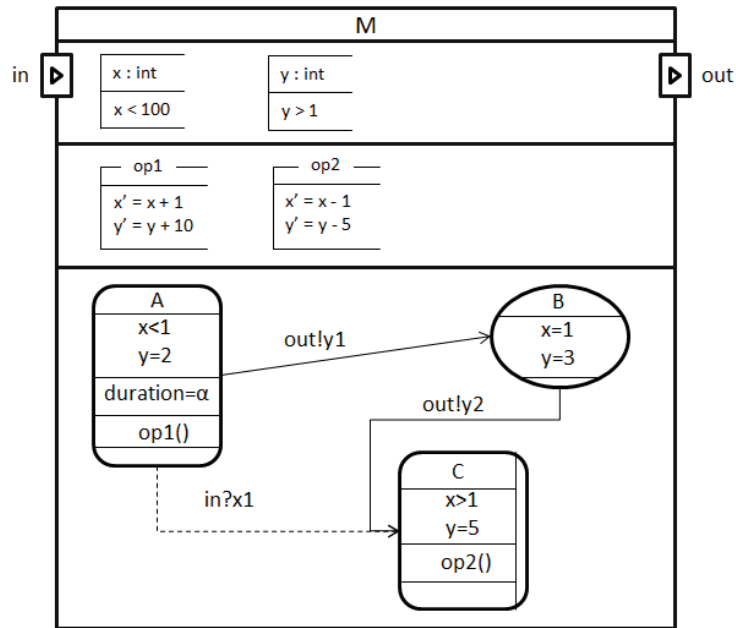


Figure 3-12: An atomic HiLLS model

As we can see in Figure 3-12 which contains a HiLLS atomic model: a *HSystem* named M corresponding to a DEVS atomic model. A *HSystem* is visually similar to a UML class diagram with additional features. It is a box consisting of four compartments, which are reserved respectively for name, attributes, operations and system behavior. Attributes and operations are specified using Z schema. It also has an input and an output *Interfaces* that contain *Ports* for receiving and sending events (which are named in and out). The system behavior is captured by a transition diagram, where nodes (A, B and C) are *Configurations* and edges are *ConfigurationTransitions*. *Configurations* are defined by predicates. For example, the properties of *Configuration A* is specified as $x < 1, y = 2$, and for *Configuration B* it is $x > 1, y = 5$. In addition, according to the shapes of each *Configuration* we can tell that A is a *FiniteConfiguration* with a finite *sojournTime* which is set to α . B is a *TransientConfiguration* with a zero lifespan and C is a *PassiveConfiguration* with an infinite *sojournTime*. When the system is in *Configuration A*, and the elapsed time reaches α , then an *InternalTransition* will be observed (the solid line from A to B): the system sends out an output y_1 through *Port* named out (*out!y1*) first, and then transitions to *Configuration B*. Since B is a *TransientConfiguration* it will transition instantaneously: it sends out y_2 through the same output *Port* (*out!y2*), and transitions to C through an *InternalTransition* (the solid line from B to C). However, if A receives an input event x_1 through its input *Port* in (*in?x*) before its lifespan expires, an *ExternalTransition* occurs (the dotted line from A to C), and the system transitions to *Configuration C*.

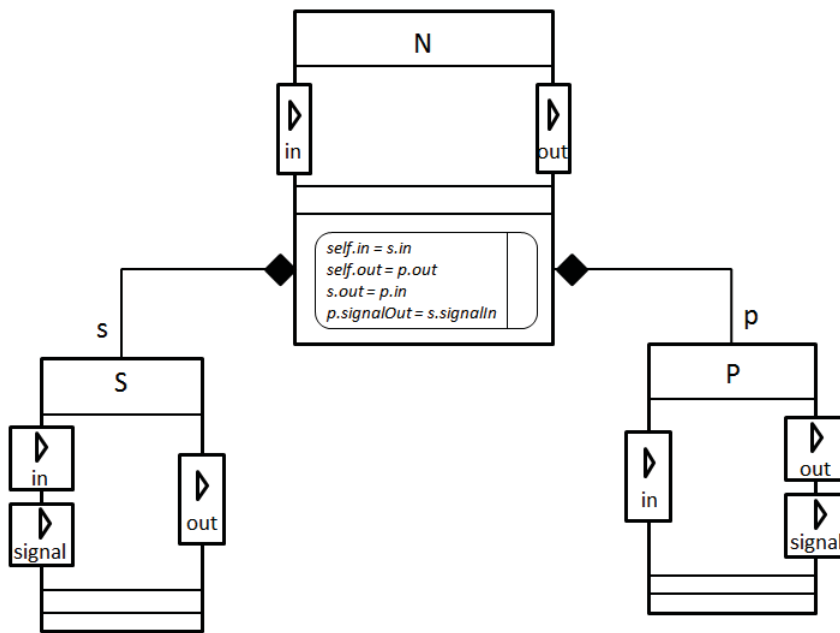


Figure 3-13: A coupled HiLLS model

Figure 3-13 gives another simple example which shows how HiLLS atomic models couple together to form a complex model, in which we omit other details for each atomic model (since we have introduced these above in the example of the HiLLS atomic model). In this example, the coupled HiLLS model is a *HSystem* named *N* with two unitary components: *S* and *P*, each of which is also a *HSystem* as well. We call *S* and *P* are the components of *N*, where the composition relation is specified in the same way as how the composition relation is defined in UML class diagram. Other entity relations can be used as well. As indicated in the metamodel, *HSystems* can interact with others through their ports, and couplings are established to enable such a kind of interaction. There are three types of couplings: *InputCouplings* (which are defined to let the system's components receive external events), *OutputCouplings* (which are established to allow the system's components to send out events), and *InternalCouplings* (which are created to enable the communications between components of the same system). In this example, the predicates specifying couplings are defined in the passive configuration of the *HSystem* *N*. More specifically, the predicate *self.in = s.in* defines the *InputCoupling*, *self.out = p.out* specifies the *OutputCoupling*, and *s.out = p.in* and *p.signal = s.signal* define the *InternalCouplings*.

One can informally see how HiLLS operational semantics can be obtained from DEVS. Regarding a DEVS atomic model, its state set is the set of all states of all

HiLLS *Configurations*. The DEVS input and output sets are those of the HiLLS model. The DEVS external (respectively internal) transition function is derived from the set of all transitions obtained from the dotted (respectively solid) lines of the HiLLS system behavior diagram. The DEVS time advance function is obtained from the *sojournTime* of the HiLLS *Configurations*. The DEVS output function is derived from the schemas carried by the internal *ConfigurationTransitions* of the HiLLS behavior diagram. In term of a coupled DEVS model, the external input coupling, external output coupling, and internal coupling are adapted from HiLLS *InputCouplings*, *OutputCouplings*, and *InternalCouplings*, respectively. A formal semantic mapping between DEVS and HiLLS is given in [Maiga 2015].

The DEVS model that can be derived from the HiLLS examples given in Figure 3-14 and Figure 3-15 are the following, respectively:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta \rangle, \text{ such that}$$

- $X = \{(in, x) \mid x \in dom(in)\}$
- $Y = \{(out, y) \mid y \in dom(out)\}$
- $S = \{A, B, C\}$
- $\delta_{int}(A) = B$ and $\delta_{int}(B) = C$
- $\delta_{ext}((A, e), x1) = C$
- $\lambda(A) = (out, y1)$ and $\lambda(B) = (out, y2)$
- $ta(A) = \alpha$, $ta(B) = 0$ and $ta(C) = +\infty$

$$N = \langle X, Y, D, \{M_d\}_{d \in D}, EIC, EOC, IC \rangle, \text{ such that}$$

- $X = \{(in, x) \mid x \in dom(in)\}$
- $Y = \{(out, y) \mid y \in dom(out)\}$
- $D = \{N, S, P\}$
- $EIC = \{(N.in, S.in)\}$
- $IC = \{(S.signal, P.signal), (S.out, P.in)\}$
- $EOC = \{(P.out, N.out)\}$

3.5 Process Enhancement Approaches

Under the continuous threat of competition, there is always a need for organizations to increase their business process operational performance in order to let them achieve their business goals more efficient, and stay competitive in business. Essentially, the reduction of cost, the elimination of waste, and the improvement of control (and information) flows result in more efficient processes, which eventually lead to achieve organizations' financial goals. Therefore, organizations should

improve their products and services continually by means of enhancing their process performance in terms of efficiency, effectiveness, and flexibility [Clauberg & Thomas 2013]. The large number of buzzwords like BPM (Business Process Management), BPI (Business Process Improvement), Process Analytics, Case Management, etc., shows the interest of organizations to monitor and analyze their business activities. Definitions of these process management approaches vary in publications, but in general all of them relate to one common ultimate goal: the improvement of process performance. We introduce the main process enhancement approaches in this section.

3.5.1 Process Analytics

Process analytics generally refers to a group of approaches and tools used to study information of events, in order to support the decision-making of business managers. Essentially, the results from process analytics offer an insight about the efficiency, effectiveness, and potential risks of processes to managers. From a performance point of view, process analytics helps managers react fast to events, and evaluate immediately the impact of decisions made. While from a compliance perspective, process analytics ensures that governing rules and regulations of processes are met [Muehlen & Shapiro 2010].

To provide actionable information to decision makers, a set of process metrics (e.g., Key Performance Indicator (KPI)) should be defined and used. Such process metrics measure and evaluate how processes proceed towards business goals [Peterson 2006]. Usually, they often use rates and percentages to show the process-related information, instead of using mere raw data. In order to obtain insightful information from process metrics, they should meet certain criteria such as accurate, little-cost to obtain, easy to understand, and actionable (actionable process metrics establish unambiguous relationships between metrics values and decisions). In most cases, the general process metrics relate to the time-stamp: e.g., the change-over time (which refers to the difference between the selection of a work and the actual starting of that work, and in terms of knowledge work it often refers to the mental adjustment of workers when they switch from one task to another), the suspending time (which measures how long time a process suspends), and the gross processing time (which refers to the time spent from the instantiation of a process to its final completion) [Muehlen & Shapiro 2010]. The comparisons between the same time-related metric obtained from different process instances provide a basic view to the process model.

Basically, process analytics can be applied to study process information from three perspectives: understanding what has happened in the past (where the analysis focuses on the historical data collected from completed processes), monitoring what is going on at the moment (where process data recorded at the real time are studied), and

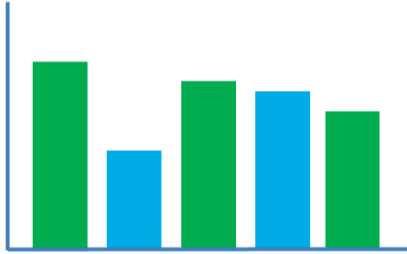
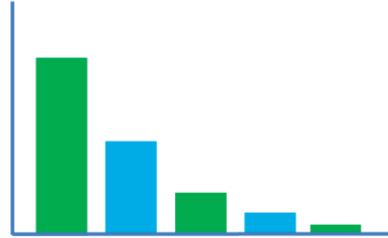
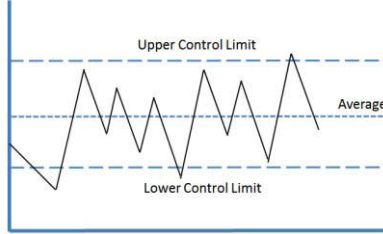
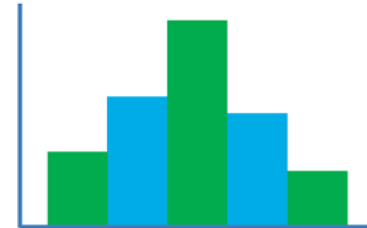
forecasting what will happen in the future (where both the historical and the real time process data are used to predict the future behavior). To ease and visualize the process data, as summarized in the table below (Table 3-1), data visualization tools are usually adopted when analyzing process information.

In a nutshell, analyzing process information helps business managers gain insight into their processes, provides supports for their decision-makings, identifies hidden waste and bottlenecks, and yields potential opportunities for optimizing their processes. Currently, most of the BPM systems are implemented with a process analytics component to collect, analyze, and monitor process events data. Such applications focus on either examining completed business process information, (e.g., process controlling), or monitoring active business processes (e.g., business activity monitoring).

3.5.2 Process Improvement

In order to survive and thrive successfully in today's constant changing environment, organizations realize that it is inevitable for them to adjust themselves to meet new requirements. One major thing is to improve the process performance so that they can be competitive. Process improvement is a systematic process management approach to help organizations archive remarkable changes and enhance their operational performance [Forster 2006]. Essentially, main reasons behind process improvement efforts include the needs for organizations to reduce cost, develop efficiency processes, and respond to regulations [Radnor 2010]. It covers tools and applications for improving process such as process analytics, process monitoring, etc. In general, all process-governing methodologies relate to continuously improvement, identification and elimination of waste or bottlenecks, and establishment of better process models. A basic scenario of process improvement is to establish an as-is model for the current process, analyze and explore potential beneficial changes, and construct a to-be model reflecting the changes, as given in Figure 3-14.

Table 3-1: Common data visualization tools

Tool	Description
<p>Bar Chart</p> 	<ul style="list-style-type: none"> • A <i>bar chart</i> visually displays data in bars. • The height of the bars is used to represent the size or quantity measured.
<p>Pareto Chart</p> 	<ul style="list-style-type: none"> • A <i>Pareto chart</i> represents data through the lengths of bars, in which bars are arranged in descending order from left to right. • It helps identify the problems that have the greatest impact. • It is based on the Pareto principle that 20% of the source causes 80% of a problem.
<p>Control Chart</p> 	<ul style="list-style-type: none"> • A <i>control chart</i> is a statistical tool used to distinguish between common-cause and special-cause variations. • It helps determine which process is out of control.
<p>Histogram</p> 	<ul style="list-style-type: none"> • A <i>histogram</i> summarizes data that has been collected over a period of time and presents its frequency distribution in bar form. • It helps reveal the centering, variation and shape of the collected data.

Once organizations operational performance is improved, then they are able to produce products or provide services with high quality, low cost and on-time delivery (where Quality, Cost and Delivery (QCD) is regarded as the key for organizations to success), and their ultimate goal, making money, will follow. We explain the widely used process improvement approaches in the following, including Process

Reengineering, Lean, and Theory of Constraints (TOC).

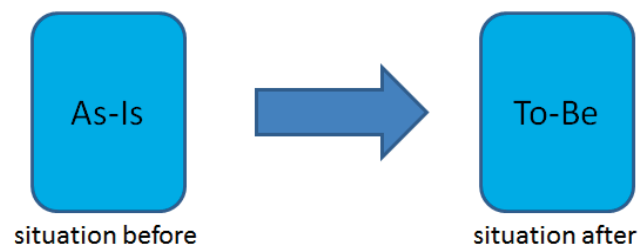


Figure 3-14: A general scenario for process improvement

3.5.2.1 Process Reengineering

[Hindle 2008] pointed out that process reengineering is an approach to rethink and redesign business processes to a radical extent. This approach results in a dramatic improvement in terms of cost, quality and service. In literature there are other organizational change approaches, but what makes process reengineering unique is that it focuses primarily on the business process [Kettinger et al. 1997]. Many companies reported that they had gained significant benefits from reengineering their processes [Cafasso 1993]. In manufacturing area process reengineering has been a successful approach. Some experts claimed that it is also an effective strategy for knowledge work [Davenport & Short 1990]. Consequently, process reengineering has arisen as a major solution for organizations to change and improve their processes.

A generic framework for process reengineering efforts contains six phases [McDonald 2010]. As illustrated in Figure 3-15, it includes *plan*, *analyze*, *re-design*, *acquire resources*, *implement*, and *continually improvement*. Similar framework was also given in [Goksoy et al. 2012] with some additional steps.

1. *Plan*
Select an existing process to improve, where the targeted process contains transparent or potential problems.
2. *Analyze*
Construct the as-is model of the selected process, and examine it for problems.
3. *Re-design*
Explore and determine the changes to make in the to-be model of the targeted model, test the ideas and consider potential implications.
4. *Acquire resources*
Obtain the resources needed to make the changes.

5. *Implement*
Put the process changes into practice.
6. *Continually improve*
Evaluate the effectiveness and efficiency of the to-be model, and make further changes in a continually manner.

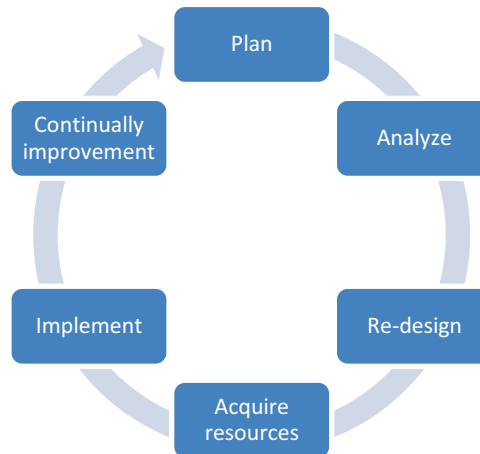


Figure 3-15: The framework for process reengineering

During the last decades of years, similar buzzwords like process reengineering have been proposed, such as process restructuring, process redesign, etc. They all focus on making process changes in order to increase productivity, reduce cost, and improve product (or service) quality and customer satisfaction. The main difference among them is the degree of change: minor, medium, or radical [Cao et al. 2001]. For instance, process reengineering often brings radical process changes, while the degree of changes in process redesign is medium. The “changes” made for processes is the key factor to the success of process improvement. Therefore, the frequency of change often indicates the frequency of improvement, which determines whether a company is competitive or not. However, one major challenge is that in most cases, it is difficult or impossible to know the outcome of changes before putting them into practice when redesigning the process model and testing new ideas, as is shown in the framework in Figure 3-15. Evidence has shown that the failure rate of process changing projects is as high as 70% [Marjanovic 2000] [Cao et al. 2001]. A failed process reengineering practice is a waste of resources (time, money, materials, etc.), which organizations are not willing to pay. However, the payback for a success process reengineering project is so enormous so that organizations strive for improving their processes through this approach.

3.5.2.2 Lean

Lean (also known as lean manufacturing) is a process management philosophy. It was mainly derived from the Toyota Production System (TPS) theory, and was named as lean since the 1990s [Holweg 2007]. Traditionally, lean is applied within manufacturing area, and it aims at identifying and eventually eliminating all sorts of waste in order to deliver products and services with high quality. It helps organizations make more money with less cost. To this end, several lean principles are established to help managers create a lean culture in their organizations. Major ones include *waste elimination*, *Just-In-Time (JIT)*, and *continuous improvement*. As a process analysis and improvement approach, lean has been successfully implemented in numerous companies.

Waste Elimination

Essentially, lean reduces cost by adding *value-added activities* and eliminating all other *non-value-added* activities. Generally, the term "value" often refers to the benefits customers expect to gain from services or products. Therefore, it is the customers who define whether an activity (or a task) provides value to them or not. An activity is said to be a *value-added* activity (i.e., it adds value to customers) if it meets the following criteria [Sayer & Williams 2012]:

1. The customers must be willing to pay for it.
2. The activity must transform the product or service in some way.
3. The activity must be done correctly the first time.

Consequently, an activity that does not meet the criteria listed above is called a *non-value-added* one. From the customer's perspective, *non-value-added* activities are considered as wasted efforts in terms of time, resource, and so on. *Non-value-added* activities are further categorized into three types: *irregularities*, *overdoing* and *waste* (which are also known as *mura*, *muri* and *muda* in Japanese, respectively). *Irregularity* refers to the waste caused by the variation happened in quality, cost, and delivery. Understanding the nature of variation is one key point in Deming's System of Profound Knowledge, a quality analysis philosophy for products and services [Deming 1986]. A smooth workflow will be interrupted by *irregularity*. *Overburdening* refers to the unnecessary or unreasonable demands placed on employees or the equipment. Too many demands will exceed the capacity of employees or the equipment, and it often leads to other problems such as out of service of machines. *Waste* refers to an activity that consumes resources, but does not create any value for customers. Taiichi Ohno (the creator of the TPS theory) defined seven basic types of *waste* that organizations should eliminate, including *transport*,

waiting, overproduction, defects, inventory, motion, and excess processing, as detailed in the following.

1. *Transportation*

The unnecessary movement of products or materials is waste since it requires spaces, causes inventory accumulation, etc. The fact is that no actual value is created or added to the final service or products during transportation. In knowledge work, it often refers to the needless movement of information, e.g., transferring information between different databases (and duplicated information recorded in different data repositories will cause the transportation of information).

2. *Waiting*

Waiting means idle time, and waiting in all forms is waste. It often refers to delays like waiting for instructions, approval, or work to arrive. For instance, a case worker waiting for information needed to start or finish his work is considered as waste. This may happen if a bottleneck exists in the upper stream of the workflow.

3. *Overproduction*

Producing more than customers required is waste since the excess products consume more raw materials, and require additional manpower before they are needed. Generally, managers try to fully utilize their machines and human resources, and this generally results in excess products. If case workers complete more work than required, the storage of the completed work will be so huge that the products or services may become obsolete since new information might be received.

4. *Defects*

A defect may refer to a reject, a design change or an item failing to meet specifications. Any item that is viewed as a defect is waste since it does not add any value to the product. Plus, it requires additional work (e.g., rework) to correct it, where such additional work is also considered as *non-value-added* activities. If the defect is discarded, then a great waste of resources and effort will be observed. Incorrect data or information commonly leads to defects in knowledge work.

5. *Inventory*

Stock of anything (including the final products, semi-final products, raw materials, etc.) anywhere is waste since it takes up space. In addition, it is under the risk of being damaged and obsolete. Moreover, excess items storing in the warehouse as inventory add no actual value to the production process. In knowledge work area, requests accumulated in the backlog can be considered as inventory.

6. *Motion*

Motion here refers to any movement of people that does not add value to the process. For instance, employees may move around to look for tools or

information due to poor layout or design of the workplace. Another typical example is that case workers are trying to find the key information for them to proceed with.

7. *Excess processing*

Excess processing, or extra processing, refers to putting more work or effort into the product or service than necessary. This often happens if employees lack well understanding of the requirements. For example, analyzing and categorizing information when only raw data is needed, or reformatting data when data is formatted already.

Just-In-Time

Traditional manufacturing produces products based on forecast: organizations predict the amount of products customers will require, and start to work before they receive orders. This way, final products will stay as inventory until customers place their orders. This type of manufacturing strategy is known as a push system. Organizations operating as push systems ensure that they will have sufficient products to meet customers' demands. However, a large number of final products and Work-In-Processes (WIPs) will stay in the warehouse as inventories, which are considered as waste in lean. In addition, in most cases this kind of prediction is inaccurate since today's business environment varies from time to time rapidly.

On the contrary, the so called a JIT system (or a pull system) starts with customer orders. It is a manufacturing management philosophy aiming at producing the items demanded by customers just at the right time, with the required quality and quantity [AIDT 2006]. No products will be produced until an order is received from downstream. This way, no excess items (e.g., raw materials, WIPs, final products, etc.) are needed to be stored as inventory. Therefore, the level of inventory will be reduced, as well as the cost on storing and transporting them. Moreover, investment in working spaces will be lower, the *lead time* (which refers to the amount of time used from receiving an order to delivering to customers) will be shorter, and eventually the quality will be improved [Javadian Kootanaee et al. 2013].

To produce at a rate that meets customer orders, JIT focuses on bringing the *cycle time* to the *takt time* as close as possible. The *cycle time* refers to the amount of time used from the start to the end of a process, while the *takt time* refers to the rate (pace) of customer demand. The ideal situation is to produce one product, then to deliver one, and no inventory exists. This is one of the goals JIT strives to achieve. In addition, *kanban* cards are used to indicate the need for materials, where a *kanban* card refers to a visual indicator or symbol that contains order information from downstream to upstream. Moreover, the *continuous flow* principle is adopted when establishing working procedures, where *continuous flow* refers to a system in which work units are

moved through operations from step-to-step with no WIPs and delays in between. To this end, the workplace is often organized into a *U-shape cell* (work cells are arranged in a U shape). All necessary equipment will be rearranged in order to ensure the efficiency and effectiveness of employees.

Continuous Improvement

Continuous improvement, also known as Kaizen, refers to the efforts to improve products, services, or processes in a continuous manner. It is a constant improvement practice, and it involves everyone in the organization, from top managers to employees. Continuous improvement focuses on small and subtle improvement, and dramatic changes can be observed over time.

In reality, waste elimination is the core ground rule for continuous improvement. Moreover, the actual improvement practice often involves the adoption of different tools. A major one is the *PDCA* (Plan-Do-Check-Act) cycle, as is given in Figure 3-16: an iterative four-step management methodology used to maintain and improve system performance, which is very similar to the process reengineering framework showing in Figure 3-16.

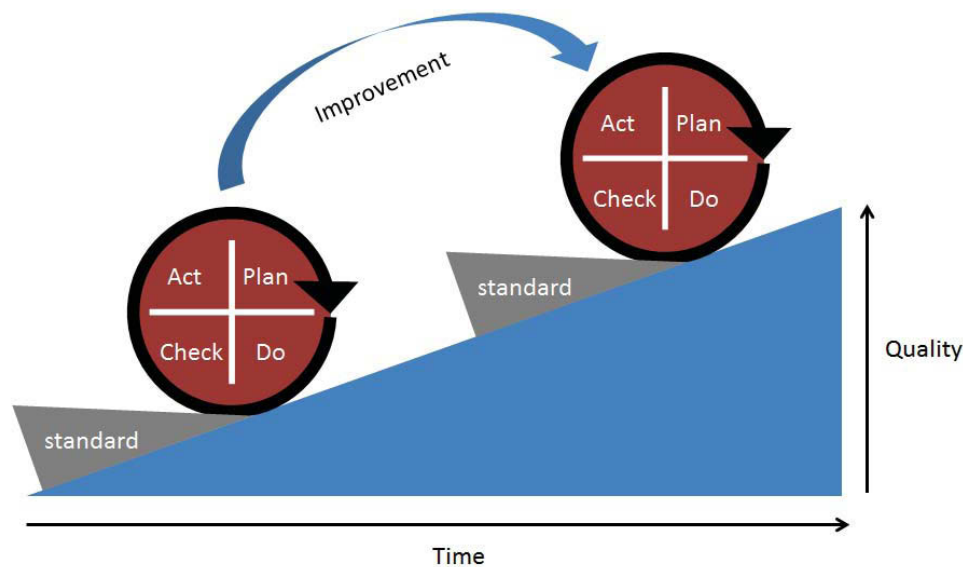


Figure 3-16: The PDCA cycle

Plan refers to establish an effective solution to one or more problems within a process of interest, and determine the target for the improvement practice. *Do* refers to implement the plan established on a small scale, i.e., put the initiatives into practice experimentally. While *Check* refers to monitor the value of performance metrics and evaluate the result of the changes. If the expected enhancement of system performance is not shown, managers need to go back to the first step (i.e., *Plan*) to

modify the original proposals. Once they obtain their expected results, they will performance the *Act* procedure: to standardize and perform all the changes. At this point, a performance-improved system is established. Managers will then target on other problems, and solve them by following more the *PDCA* cycles. Once the system performance is maintained at a certain level, carrying out another *PDCA* cycle will bring the system performance to a higher level, and the standard established from the previous round will stop the performance from moving down. Each *PDCA* cycle will solve one single small issue; and major jump in terms of performance will show over time, after several rounds of *PDCA* project.

3.5.2.3 Theory of Constraints

The Theory of Constraints (TOC) is a system management philosophy originally proposed by Eli Goldratt in his book “*The Goal*” [Goldratt et al. 1992]. It aims at improving system performance by identifying and eliminating *constraints* (which are also called *bottlenecks*), where a *constraint* refers to the resource (a machine, or an employee) whose capacity is equal or less than the demand. On the contrary, a *non-constraint* refers to the resource whose capacity is greater than the demand. By now, TOC has been applied to various domains, such as aerospace, automotive, health care, and manufacturing. The companies involved are the world’s most renowned ones, including Boeing, General Motors, Intel, and so on [ACCA 2011].

TOC is developed on the basis of the fact that any system can be seen as a chain, where its activities (or processes) are linked and connected together as a whole. The entire system's performance will be determined by its weakest link, i.e., the *constraint*. Therefore, in order to improve the system’s performance organizations need to make the full usage of *constraints*. However, it is not the same case with regard to *non-constraints*: if non-constraint resources work at their full capacity, then they will produce extra items to downstream, if a *constraint* exists in the downstream who cannot consume what it receives from upstream. Additional WIPs and semi-final products will be observed (i.e., the inventory will increase). Moreover, running every machine at 100% can increase the operational expense for both machines and employees since machines lose lifetime hours and employees consume energy and will become exhausted very quickly. Therefore, due to such inter-dependencies and variations existed between activities (or processes) a system's optimum performance will not simply equal to the sum of all the local optima. To improve performance, the system should be considered as a whole instead of a group of isolated individual, and *constraints* should be well governed since they limit the actual performance of systems [Goldratt et al. 1992]. To this end, Eli Goldratt proposed the *five focusing steps* approach, which targets on identifying and optimizing *constraints*.

1. *Identify the system's constraints.*
Constraints are not always obvious. However, two measurements can be used here to help identify them: the amount of work items waiting in the queue, and the idle time employees from subsequent step spend in waiting for work items. In addition, managers should decide which *constraint* is the most important one once they have identified all possible candidates.
2. *Decide how to exploit the system's constraints.*
 This step involves making the constraint resource to be used as much as possible (from the perspective of utilization), and to produce work items as many as possible (from the perspective of productivity).
3. *Adjust everything else to the decisions made in Step 2.*
 At this step the *constraint* works at its maximum capacity. To pace with the *constraint* in an efficient manner, all other resources should adjust their working speed. The key principle is to let all resources work in a rhythm so that the work load of the constraint incoming work load reduces, and simultaneously all other employees do not have plant of waiting time.
4. *Elevate the system's constraints.*
 If with the improved *constraint* the requirements still cannot be met, then the *constraint* should be elevated. It means that larger scale changes regarding the whole process are required through approaches such as process restructuring, resource reallocation, etc. Since the plan of changes may increase the inventory and operational expense, managers should analyze the changes well enough before putting them into practice.
5. *If a new constraint is broken in Step 4, go back to Step 1, but do not let inertia become the system's new bottleneck.*
 Once the *constraint* is elevated, another *constraint* might appear to catch attention. Managers should strive for looking for and removing *constraints* by following these steps continuously.

Throughput Accounting

In contrast with traditional cost accounting method, Goldratt also proposed the Throughput Accounting (TA) methodology that is used to monitor, manage and analyze organizations operational performance. Traditional cost accounting method focuses on making profit through reducing and cutting cost. It views the profit-increased equals to the cost-saved, where the total cost is the summation of the cost of each component, and its goal is to lower each component's cost by maximizing the usage of each (i.e., the local optimum). However, TA is a management accounting methodology that takes the system's constraints into consideration, and it seeks to achieve balance all components in order to achieve the whole system's optimum. It emphasizes on improving the operational performance and maximizing profits by increasing *throughput*, while reducing *investment* and *operating expenses*

simultaneously [Goldratt et al. 1992].

From the perspective of TOC, *throughput* refers to the rate at which the entire system generates money through sales. In most cases, it is used to refer to the added value through sales, $throughput = sales - total\ variable\ costs$, where *variable cost* refers to the costs that change along with the volume of production such as direct materials, production supplies, and so on [Garrison et al. 2003]. To increase *throughput*, it is necessary to well manage the usage of constraints. This way, the amount of items delivered will be increased, and this will result in the increase of total sales supposed that the unit price stays. *Investment* often refers to the money tied up in the system. Basically, it includes the investment the organization has made (tools, capital equipment, furnishings, etc.), as well as the physical inventory (WIPs, finished products, etc.). In order to decrease *investment*, organizations should primarily reduce their inventory level. They could also seek to reduce the investment they have made for their business, but in most cases the reduction of things like tools or capital equipment will lead to other potential problems such as low productivity (due to the fact that employees have plenty of idle time in waiting as they do not have the tools they need). *Operating expenses* refer to the money organizations spend in turning *investment* into *throughput*. Expenditures such as salaries of employees, bills of supplies are all considered as *operating expenses* of organizations, and they should be reduced as well in order to make profits.

These three measurements mentioned in TA are adopted to support the decision making of business managers in the tactical management level. Compared with traditional cost accounting measurements, they provide more valuable and actionable information. Managers are able to identify where the problem hides within their systems. In addition, they also could know which aspect they should pay more attention to. In addition, on the basis of TA measures, financial performance measures (which are at the strategic management level) such as *net profit* and *return on investment* (ROI) can be directly obtained, where the former indicates the actual profit organizations have made (where $net\ profit = throughput - operating\ expenses$), and the latter evaluates the efficiency of an investment (where $ROI = \frac{throughput - operating\ expenses}{investment}$). We could see that in order to increase *net profit*, *throughput* should be increased, and *operating expenses* should be decreased simultaneously. To make an investment to be more efficient, *investment* should be decreased as well. Moreover, the efficiency of employees can also be measured on the basis of TA using *productivity*, $productivity = \frac{throughput}{operating\ expenses}$, and the increase of *throughput* and the decrease of *operating expenses* will results in a high efficiency of employees [IMA 1999]. A summary of the performance metrics is given in the table below.

Table 3-2: The summary of performance metrics

	Name	Formula
Tactical Level	throughput	$throughput = sales - total\ variable\ costs$
	investment	the investment the organization has made
	operating expense	the money organizations spend in turning <i>investment</i> into <i>throughput</i>
Strategic Level	net profit	$net\ profit = throughput - operating\ expenses$
	ROI	$ROI = \frac{throughput - operating\ expenses}{investment}$
	productivity	$productivity = \frac{throughput}{operating\ expenses}$

3.5.2.4 Simulation in Process Improvement

As we introduced above, regardless of which process improvement approach organizations adopt, the major factor is to construct a new model of the process (i.e., a to-be model) through redesigning the existing model (i.e., the as-is model) with incremental or even large scale changes proposed. However, managers are challenged with the question of whether or not putting the proposed changes into practice. The fact is that, it is difficult for organizations to make decisions if they do not have sufficient supporting data. Their hesitations arise from the uncertainty of obtaining a positive result [Clauberg & Thomas 2013]. A common question for them to answer is: how can we be confident to claim that the to-be model with modifications will be better than the as-is one?

Simulation of business processes is considered as a cost-effective means to predict the potential impacts of changes proposed. In addition, process simulation also helps examine and compare all proposed improvement alternatives without actually change the existing processes. Moreover, through simulating processes potential bottlenecks can be discovered, and waste can be easily identified as well, in both the as-is and to-be situations. The simulation results can be considered as a quantitative data support when comparing potential process improvement scenarios. This way, managers can confidently choose an improvement proposal among others since they ensure that the to-be model will be better than the current one on the basis of the simulation results. Simulation offers business analysts a way to test and verify suggested process improvement changes [Hlupic 2003].

To ensure the quality of process simulation results, the process models should be established correctly and accurately. Otherwise, inaccurate process models lead to inaccurate simulation results, which further offer unreliable and un-meaningful data supports to managers when making decisions. Many vendors have implemented a simulation module in their process management applications, such as [Nissen &

Levitt 2002] [Hlupic 2003] [Barnett 2003] [April et al. 2006] [Peinl & Maier 2011] [Rust 2011] [Clauberg & Thomas 2013]. However, for unstructured processes with case management little contribute is found in the literature.

3.6 Process Discovery

As we have mentioned, a process model specifies how the activities involved in that process will be performed at run-time, and it is generally a preferred starting point for further process monitoring and analytics. However, establishing process models manually is a difficult and sometimes even error-prone task for both business analysts and modeling experts. Especially when there are many activities involved and their dependencies are complicated and unobvious. Moreover, IT-based solutions are widely used in organizations in recent years, in order to govern and control their business processes [Yan et al. 2017]. Relevant process information is recorded in their information systems (e.g., the Workflow Management Systems), instead of in paper files. Such information systems record historical business event data in the so called event logs. An event log basically records the process information in terms of the start and end time of each activity. In some situations additional process-relevant data may be recorded as well, such as the resources that are executing the activities, the cost of performing each activity, and so on.

The data collected in event logs constitute the basis for process discovery. Process discovery is a process approach that uses the event logs as its starting point and aims at exploring and constructing business process models by merely analyzing raw data stored in event logs [Castellanos et al. 2009]. Process discovery extracts insight knowledge from them. Moreover, it helps construct process models on the basis of the historical data. In situations where organizations have their business processes without formal (or even informal) process models, process discovery is considered as a dominant approach since with this technique managers are able to explore and obtain process models on the basis of the historical records automatically, instead of constructing process models manually from scratch. Process discovery translates raw process data into process information with which organizations are able to gain deep insight into their business processes. In addition, organizations can acquire additional supports to decision-making if they adopt process discovery with other process management approaches (e.g., process analytics, process improvement).

Process discovery techniques ease the process modeling practices. Over the past years, researchers have developed many mature process discovery algorithms, each of which guarantees one or many quality characteristics of discovered process models (which are replay fitness, simplicity, generalization, and precision) [Leemans et al. 2013]. Major categories are deterministic mining algorithms (which produce well-defined and reproducible models), heuristic mining algorithms (which

incorporate frequencies of events when constructing models), and genetic mining algorithms (which describe the evolution of processes). A detailed introduction regarding process discovery algorithms with the discussion about their advantages and disadvantages can be found in [Leemans et al. 2013]. In addition, a bunch of commercial/non-commercial tools are developed as well, such as Disco, ProcessGold, and ProM. ProM is the most widely used one among others. It is a java implemented extensible (through plugins) framework that supports various process mining techniques [Verbeek et al. 2009]. Many ProM plugins have been implemented for different purposes. Thanks to the various algorithms provided with those plugins, sound and robust process models can be established through studying the knowledge hidden in the process execution logs. In this thesis, we select to use the Inductive Miner process discovery plug-in due to the fact that algorithm implemented in the plug-in guarantees to generate sound (i.e., free of deadlocks) and fitting (i.e., all the traces recorded in given event logs can be replayed) process models [Leemans et al. 2013], where the models are constructed using the process tree formalism.

3.6.1 Process Tree

Process tree is a formal modeling formalism for describing models obtained from process discovery. It depicts block-structured processes using tree notations. A detailed introduction to the process tree formalism with respect to its syntaxes and semantics is given in [Buijs 2014]. In this section, we give a taste of the process tree formalism through introducing its syntax and semantics summarized from [Buijs 2014]. In addition, a simple process tree model example is given and explained later.

The metamodel of process tree describing its abstract syntax is given in Figure 3-17.

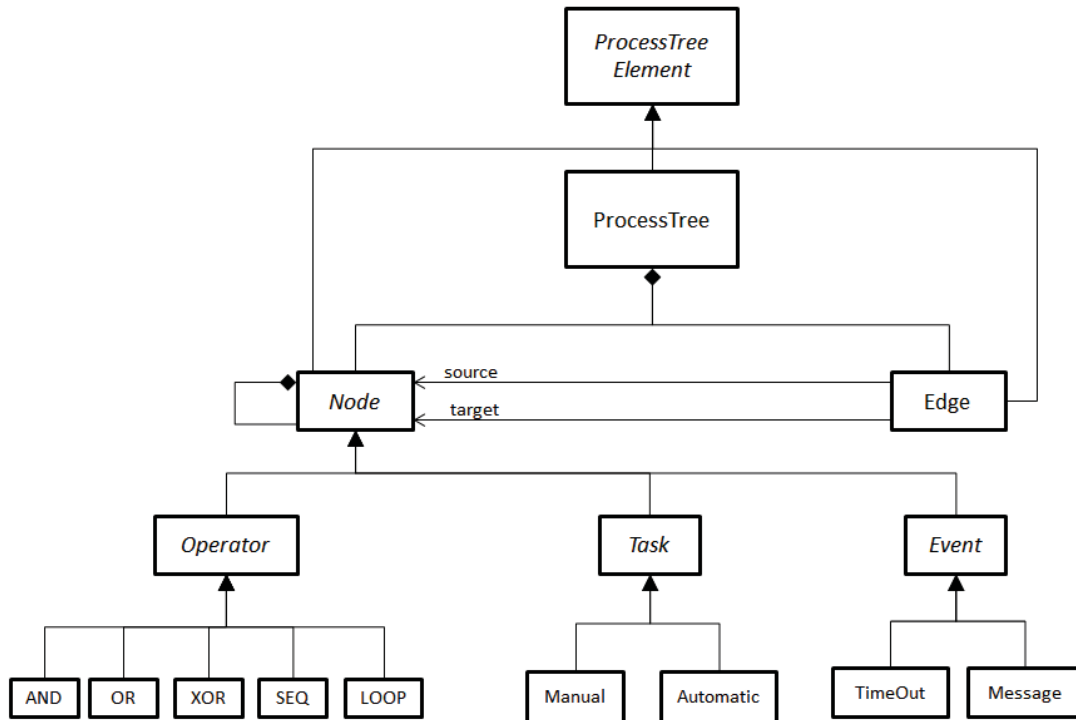






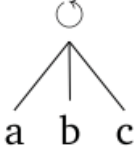
Figure 3-17: The metamodel of process tree

Generally, a process tree model consists of *Nodes* and *Edges*, where *Nodes* are further categorized into *Tasks*, *Events*, and *Operators*. *Tasks* refer to unitary activities to be done by employees, where a manual *Task* is done manually, and an automatic *Task* is completed automatically. An *Event* refers to either a time out event occurred, or a piece of message received. *Operators* describe the causal relationship between the *Nodes* it contains. In detail, there are five relationships among *Nodes*: *AND*, *OR*, *XOR*, *SEQ*, and *LOOP*, which will be explained later. *Edges* are defined to depict the hierarchical relationship between *Nodes* [Schunselaar et al. 2014].

Basically, in terms of the concrete syntax, a *Task* or an *Event* is represented by their names. The graphical notations for *Operators* are as follows: *SEQ* (\rightarrow), *XOR* (\times), *AND* (\wedge), *OR* (\vee) and *LOOP* (\odot). Each *Edge* linking two *Nodes* is a solid line. The operational semantics of process tree models is defined by the execution sequence of *Nodes* contained by *Operators*, as is given in Table 3-3 below (in which we use $Trace(Node)$ to represent all possible traces of a particular *Node*).

Figure 3-18 shows a process tree model as an example. It consists of eight *Nodes* (which are *Tasks* a, b, c, d, and e, and *Operators*: *SEQ* (\rightarrow), *AND* (\wedge), and *XOR* (\times)) and seven *Edges* connecting *Nodes*.

Table 3-3: The semantics of process tree

	<ul style="list-style-type: none"> • Alternatively, it can be written as $\rightarrow \langle a, b \rangle$. • A sequence operator (SEQ) specifies that all its children will be executed sequentially from left to right at run-time (i.e., one child can be executed only if its previous one is completed). • $Trace(\rightarrow \langle a, b \rangle) = \{\langle a, b \rangle\}$
	<ul style="list-style-type: none"> • Alternatively, it can be written as $\wedge \langle a, b \rangle$. • An AND operator indicates that there is no specific order for executing its children. • $Trace(\wedge \langle a, b \rangle) = \{\langle a, b \rangle, \langle b, a \rangle\}$
	<ul style="list-style-type: none"> • Alternatively, it can be written as $\vee \langle a, b \rangle$. • An OR operator specifies that at least one of its children will be executed at run-time. • $Trace(\vee \langle a, b \rangle) = \{\langle a \rangle, \langle b \rangle, \langle a, b \rangle, \langle b, a \rangle\}$
	<ul style="list-style-type: none"> • Alternatively, it can be written as $\times \langle a, b \rangle$. • An XOR operator specifies that ONLY one of its children will be executed at run-time. • $Trace(\times \langle a, b \rangle) = \{\langle a \rangle, \langle b \rangle\}$
	<ul style="list-style-type: none"> • Alternatively, it can be written as $\cup \langle a, b, c \rangle$. • A LOOP operator always has three children defined: a <i>do</i> part (a), a <i>redo</i> part (b), and an <i>exit</i> part (c). • A LOOP operator indicates that after the first execution of a task (the <i>do</i> part), a condition will be evaluated which decides whether the <i>redo</i> part will be executed (which will trigger the <i>do</i> part) or the <i>exit</i> part will be executed. • $Trace(\cup \langle a, b, c \rangle) = \{\langle a, c \rangle, \langle a, b, a, c \rangle, \dots\}$

Alternatively, this model can be expressed as $\rightarrow \langle \wedge \langle a, b \rangle, c, \times \langle d, e \rangle \rangle$, such that

- $Trace(\wedge \langle a, b \rangle) = \{\langle a, b \rangle, \langle b, a \rangle\}$: The AND operator indicates that there is no specific order for executing *Task a* and *Task b*.
- $Trace(\times \langle d, e \rangle) = \{\langle d \rangle, \langle e \rangle\}$: The XOR operator specifies that either *Task d* or *Task e* will be executed.
- $Trace(\rightarrow \langle \wedge \langle a, b \rangle, c, \times \langle d, e \rangle \rangle) = \{\langle a, b, c, d \rangle, \langle a, b, c, e \rangle, \langle b, a, c, d \rangle, \langle b, a, c, e \rangle\}$: The SEQ operator specifies that all its children will be executed sequentially from left to right at run-time (i.e., one child can be executed only if its previous one is completed).

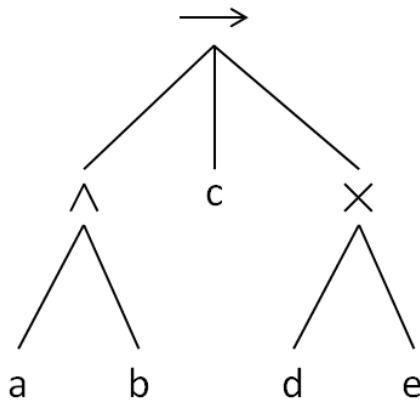


Figure 3-18: A simple process tree model

3.7 Model Transformation

In MDE, models play a key role throughout the whole system development lifecycle. Models represent the systems of interest in an abstract manner: they capture the information regarding different system development steps, such as system design, implementation, analysis, optimization, etc. In addition, models depict systems from different perspectives for various purposes, at different levels of abstraction [Biehl 2010]. Models help organizations gain a better understanding of the systems of interest [Hassan et al. 2016].

In the context of MDE, model transformation is defined as the “*automatic generation of a target model from a source model, according to a transformation description,*” [Kleppe et al. 2003] in which the source and target models describe the same system. In some cases, the input of a model transformation can be more than one source model, and the output can be more than one target model as well. In other words, model transformation takes one or more source models as its input, and automatically generate one or more target models, according to the transformation rules specified [Mens et al. 2006]. Model transformation offers a mechanism for the reuse of system information captured. On the basis of this point, various practices in MDE (such as model creation, modification, refinement, and so on) can be done automatically. Generally, model transformation is applied to convert platform independent models (PIM) to platform specific models (PSM) in MDA, where PIM refers to models that are independent from the technical space in terms of implementation, and PSM are models that require specific technological tool or platform to implement.

As shown in Figure 3-19, the basic components for a model transformation are a

source model, a target model, and a transformation engine used to executing a set of transformation rules. The source model refers to the model to be converted, while the target model refers to the model generated (as the output) through the transformation. Both the source and target models conform to their own metamodels, respectively (e.g., as given in Figure 3-19 the source model A conforms to its metamodel MMA, and the target model conforms to its metamodel MMB). Each transformation rule specifies how an element of the source model will be converted into an element of the target model. The transformation engine will interpret and then execute all the transformation rules. For each element in the source model, it generates the element of the target model by following the transformation rules. The well-defined languages used to implement transformation rules are called model transformation languages.

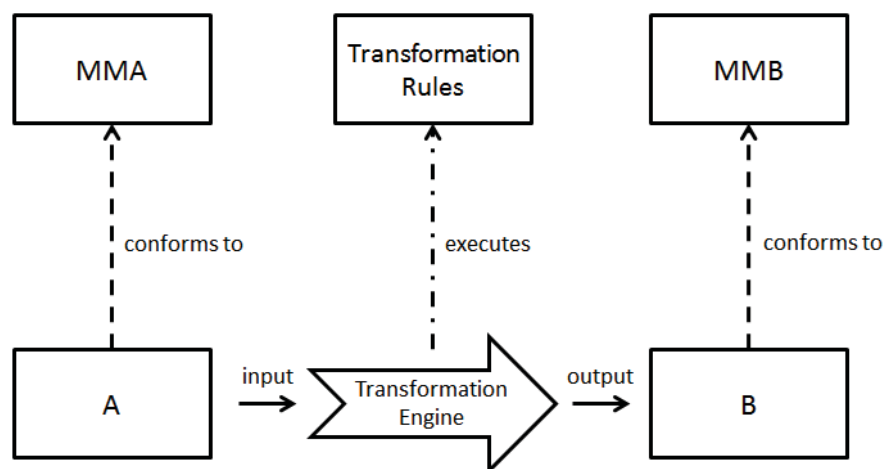


Figure 3-19: Basic components of a model transformation

The metamodel of the source model might or might not be the same as of the target one. If the source and the target models share the same metamodel, then this type of model transformation is called *endogenous*. Typical examples of *endogenous* transformations are model optimization (which transforms models in order to improve the quality of models), model refactoring (which changes the internal structure of systems), etc. Otherwise, models involved in *exogenous* transformations are specified using different languages, such as model synthesis (which converts a more abstract model to a more concrete one), model migration (in which models are at the same abstraction level). In addition, the level of abstraction of the source and the target models might change in model transformation, where the level of abstraction measures the amount of information captured in models. If the target model brings more (or less) information than the source model, then it is a *vertical* model transformation. On the contrary, a *horizontal* model transformation only changes the representation of models, and keeps the same amount of details.

Generally, a system can be described as diagram-based models (e.g., using UML

classes), or text-based models (e.g., using Java, C++). With respect to the type of target models, a model transformation can be classified as Model-To-Model (M2M) or Model-To-Text (M2T). Diagram-based models can be obtained in M2M transformations, and text-based models such as system implementation codes can be generated in M2T transformations. Essential purposes of model transformation include the following aspects, and a non-exhaustive list of model transformation intents and their properties is completely summarized in [Lúcio et al. 2016].

- *Refinement*, which aims at producing models with more precise details.
- *Abstraction*, which aims at generating simplified models with specific information.
- *Semantic Definition*, which aims at specifying the semantics of languages.
- *Language Translation*, which translates one modeling language to another.
- *Constraint Satisfaction*, which outputs models satisfying certain conditions.
- *Analysis*, which implements various algorithms for analyzing different aspects of models.
- *Editing*, which aims at manipulating models.
- *Model Visualization*, which aims at visually projecting the behavior or render the concrete representation of models.
- *Model Composition*, which aims at merging isolated models into one.

To enable model transformations, different techniques have been developed over years. Essentially, transformation rules can be specified as declarative, imperative, and hybrid. We introduce these different approaches in this sub-section, and present some model transformation languages we adopted in this these. More information regarding model transformation paradigms and languages can be found in [Huber 2008] [Dehayni et al. 2009] [Ferhat et al. 2015].

3.7.1 Declarative Approach

When defining transformation rules using the declarative approach, it is necessary to specify clearly the relationship between concepts in the source and target metamodels, respectively. In other words, declarative approach focuses on *what* should be converted into *what*. In this context, the transformation rules are often defined as mappings between elements from the source and target metamodels: each element defined in the source metamodel should be mapped onto a given element in the target metamodel. Examples of declarative model transformation languages include QVT (Query/View/Transformation) Core (a model transformation language that supports low-level model transformations, where QVT is the standard of model transformations defined by OMG), QVT Relational (a QVT Core based, high-level declarative model transformation language that supports bidirectional transformations), ModelMorf (a declarative M2M transformation language that

designed as a response to the QVT standard), and so on. Generally, adopting the declarative approach makes the mappings straightforward, concise and easy to understand [Biehl 2010].

Graph transformation is considered as a sub-category of the declarative approach. It has a theoretical foundation, it is built on top of algebraic graph grammars, and it is often used in formal approaches and proofs [Biehl 2010]. The graph transformation treats models as graphs, so that the whole model transformation is a process of manipulating (e.g., matching, replacing, etc.) sub-graphs [Dehayni et al. 2009]. In graph transformation, the left-hand side (LHS) graph will be found first, and then it will be replaced by the right-hand side (RHS) graph, where the former and the latter are the sub-graphs of the source and target graphs, respectively. AToM³ (A Tool for Multi-Formalism and Meta-Modeling) is a well-known application for designing visual system modeling languages. In AToM³, model transformation rules are expressed using graphs [De Lara & Vangheluwe 2002].

3.7.2 Imperative Approach

Instead of centering on *what* should be transformed during model transformation, the imperative approach emphasizes on the *how* perspective: it specifies an explicit control flow to manage how a source model will be converted. To this end, constructs used in general programming languages (e.g., Java, C++) are adopted in the imperative approach, such as the *for loop* statement, the *if* statement, etc. This provides a high level of control for users to explicitly specify how a target model will be generated step by step. Compared with the declarative approach, using the imperative approach will result in more complex but also more powerful transformation rules. QVT Operational (an imperative model transformation language that is built on the base of QVT Relational) is a typical example of model transformation languages that define transformation rules in an imperative manner [Kurtev 2007].

3.7.3 Hybrid Approach

As indicated by its name, the hybrid approach offers both the declarative and the imperative constructs for defining transformation rules. Users decide to choose the hybrid approach to specify transformation rules mostly because of the flexibility it offers: users are able to select different types of constructs when encountering different problems.

```

9 module CMMN2HiLLS;
10 create HiLLS_Model : HiLLS from CMMN_Model : CMMN;
11
12 uses Lib4CMMN;
13 uses Lib4HiLLS;
14
15 -- transform the root Stage to a HSystem
16 rule RootStage2HSystem{
17     from
18         s : CMMN!Stage (s = thisModule.rootStage)
19     using{
20         inputEvent : String = 'input';
21         outputEvent : String = 'output';
22         attributeName_0 : String = 'itemType';
23         attributeValue_0 : String = 'RootStage';
24         attributeName_1 : String = 'isAutoCompleted';
25         attributeValue_1 : String =
26             if thisModule.isAutoCompleted(s)
27             then 'true'
28             else 'false'
29             endif;
30         attributeName_2 : String = 'itemLevel';
31         attributeValue_2 : String = '0';
32     }
33     to
34         t : HiLLS!HSystem (
35             name <- s.name,
36             inputs <- Sequence{eventIn_Port, parameterIn_Port},
37             outputs <- Sequence{statusOut_Port, parameterOut_Port},
38             hReferences <- thisModule.allContainedPDItems(s)
39                             ->union(thisModule.allCFI)
40                             ->union(s.exitCriteriaRefs)
41                             ->collect(i | thisModule.getReference(i))
42         ),

```

Figure 3-20: A fragment of ATL transformation rules

ATL (ATLAS Transformation Language) is a well-known hybrid M2M transformation language developed on the basis of the QVT standard. Originally, it was developed as a component of the ATLAS Model Management Architecture platform [Bézivin et al. 2005]. ATL provides both declarative and imperative constructs to specify transformation rules [Jouault et al. 2008]. The declarative-style construct consists of two parts showing what transforms into what: the left-hand side accesses the source model, and the right-hand side generates the target one. The imperative-style rules with a sequence of actions illustrate how the transformation should be executed in cases of solving complex transformation problems. Source models in ATL transformation are read-only, and target models are write-only. Basically, an ATL transformation consists of a *module* (and maybe more *modules* if the problem is more complex), which is composed of a *header* section, and a set of *helpers* and *transformation rules*, as is given in Figure 3-20. The header section contains basic information for the transformation, such as the name of the transformation, the declaration of the source and the target models. A helper construct can be either an operation or an attribute helper, where the former navigates over the

source model(s), and the latter decorates information of the source model(s). Transformation rules are the main building blocks in ATL, and they are classified into two categories: *matched rules*, and *called rules*. The former is specified in a declarative style, and the latter one is written in an imperative manner. In addition, the ATL tool is implemented as an Eclipse plug-in, which provides an execution model transformation framework consisting of an ATL compiler, an ATL virtual machine, a Model handler Abstraction Layer, Model handlers, and a Model Repository [Jouault & Kurtev 2005].

Instead of enabling M2M transformations, Acceleo aims to the implementation of M2T transformations. Acceleo is a text file generation language that usually adopted to convert a model into corresponding executable codes. It is developed on the basis of the OMG “MOFM2T” Transformation standard [OMG 2008], and offers many advantages such as customization, interoperability, easy kick off, etc. In addition, it also provides both the declarative and imperative constructs. The language and its implementation (which is a plug-in integrated in the Eclipse IDE) are developed and maintained by a French company called Obeo.

3.8 Conclusion

In this chapter, we introduced the theories and techniques used in this these. In section 3.2 we explain the basic components of a system modeling language: an abstract syntax, one or more concrete syntaxes, one or more semantics domains, the syntax mappings between the abstract and concrete syntaxes, and the semantics mappings between the abstract syntax and semantics domains. Then we introduced CMMN, the standard modeling language for case management, including the abstract and concrete syntaxes, and its operational semantics. Then in section 3.4 we introduced HiLLS, a system modeling language developed for constructing multi-aspect system models. HiLLS maps its syntax into different semantics domains for various types of analysis: DEVS for simulation, formal methods for formal analysis, and UML for enactment.

Moreover, we explained the concepts and approaches regarding business process management, including process analytics, process improvement, process reengineering, lean, and TOC. Such concepts and approaches are widely adopted in business domain to help organizations govern and improve their process performance. We also reviewed how organizations used simulation as an approach to predict the potential impacts of proposed process changes, as well as examine and compare all proposed improvement alternatives without actually change the exiting processes. In addition, we also introduced process discovery in section 3.6, a process approach to explore and construct business process models by merely analyzing raw data stored in event logs, and the process tree modeling language, a modeling formalism used to

depict models generated from process discovery.

Finally, in section 3.7 we presented model transformation, an essential approach in MDE to automatically generate target model(s) from source model(s) following a set of transformation rules. The different paradigms for model transformation are introduced as well. In particular, the model transformation languages ATL and Aceleo were presented with more details, where the former converts a model to several models, and the latter transforms a model to text (e.g., programming codes). We will use the concepts and techniques described in this chapter in subsequent chapters for the specification of our case management framework.

4. CASE MODEL IMPROVEMENT

4.1 Introduction

We propose in this thesis a systematic case model management (CMM) solution that provides case workers sufficient supports to manage their cases, throughout their full life-cycles (i.e., from creating a case model to closing it), in an adaptive and flexible manner. To explain in detail, our approach contains two main modules, as given in Figure 4-1: Case Model Discovery (CMD) and Case Model Improvement (CMI). The CMD module relates to exploring case models by analyzing event logs, in which data from reality relating to special cases are recorded. CMI takes advantages of the discovered CMMN case models (which are called the as-is models obtained from CMD) as a starting point, analyzes their potential issues and operational performance from different perspectives, and offers case workers help in constructing and analyzing the improved CMMN case models (which are called the to-be models).

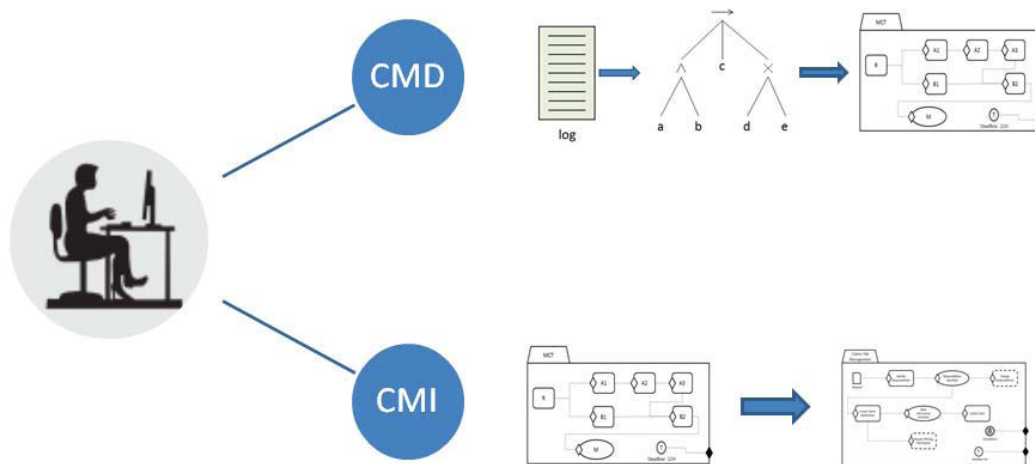


Figure 4-1: A global view of our CMM approach

Figure 4-2 gives the whole workflow of our CMM approach, including the steps and their order in a flow chart, as well as the explanations of what each step does. Generally, case workers start with managing special cases, and historical data such as starting time, completing time, etc. are collected and recorded in an event log. Using the *process discovery* technique, we can obtain a Process Tree model which captures the activities and their logical relations recorded in the event log. Through model transformation we convert the Process Tree model to a CMMN model (which is our as-is model). Case workers now can have a better view on their case models that depicting how activities are done: case models are explored automatically, and displayed in a way that case workers are familiar with. This is what the CMD module (as shown by steps 1 and 2) offers.

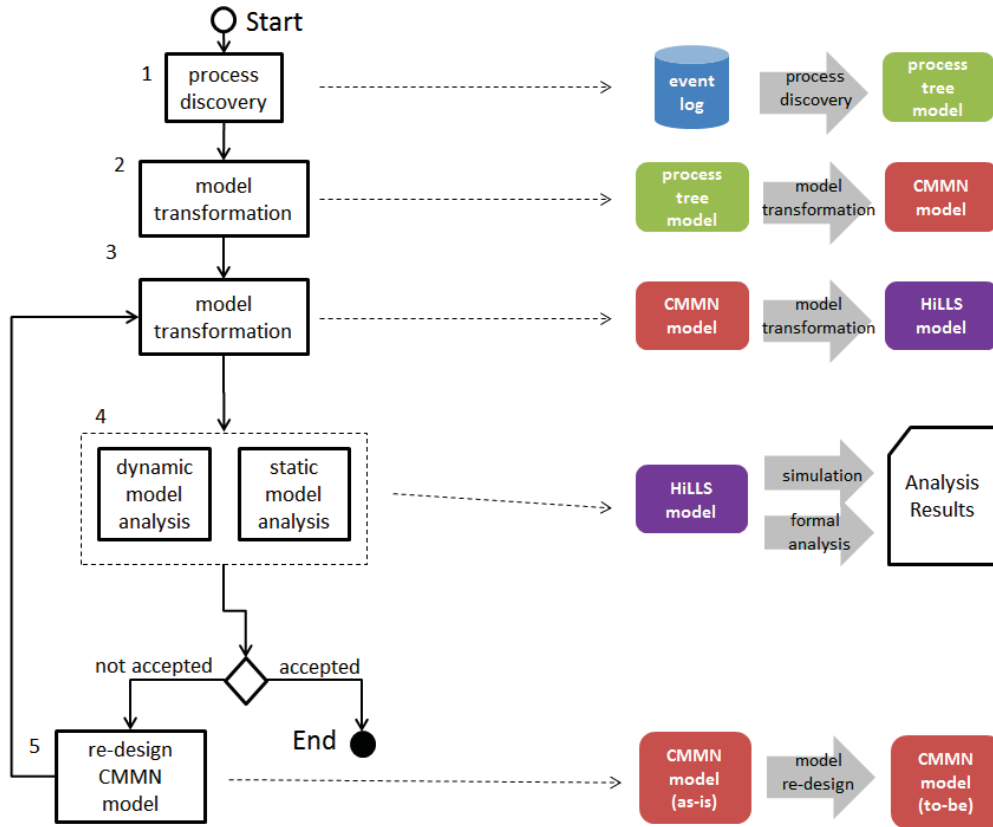


Figure 4-2: The flow chart of our CMM approach

In order to analyze the CMMN models and predict their operational performance, we envision transforming CMMN models into HiLLS models. It is the HiLLS models we analyze in both dynamic and static manners. Generally, the as-is model is the one we put our efforts on, and the to-be model is our target to obtain that has better performance and meets requirements. To this end, case workers can analyze the as-is case models in a dynamic manner (through simulation), or in a static manner (through formal analysis). We have also specified a set of performance metrics from different perspectives, in order to help case workers assess and measure the health and performance of both as-is and to-be case models. Case workers will re-construct their case models on the basis of analyzing results, and then analyze them again to check if the result is accepted or not. If not, the same steps will be followed again (steps 5, 3, and 4), until the expected result is shown. We call this repeated effort CMI, which consists of steps 3, 4 and 5. The whole story is illustrated in Figure 4-3 from another point of view.

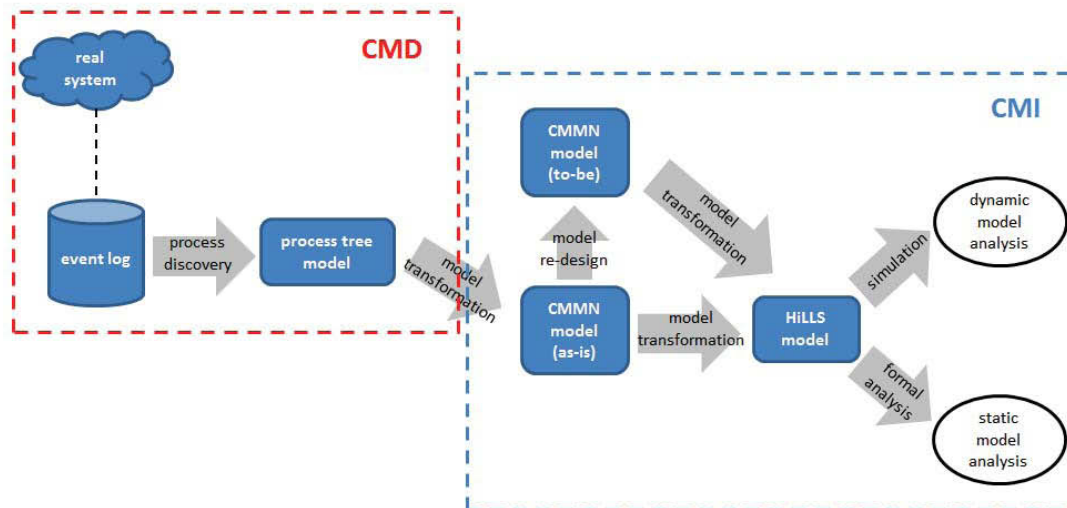


Figure 4-3: The whole picture of the CMM approach

In this chapter, we will begin with explaining the CMI module part first, due to the fact that most of our contributions are done here: we establish an approach to help case workers analyze the as-is case models, assess their performance, explore their hidden problems and provide ideas for designing the to-be models, and predict the performance of the to-be case models. Case models can be finally improved and optimized on a quantitative basis. Then we will introduce the CMD module which we are inspired from the *process discovery* technique in the next chapter.

4.2 Case Model Transformation

As we have mentioned before, there is always a need for organizations to change their as-is process models in order to increase their business process operational performance, stay competitive and make more value in business. A typical challenge is that how to analyze and predict the outcomes of changes made within the to-be models in advance, in order to guarantee a success in a process improvement practice. Process simulation has been proved as an efficient solution to this challenge [Barnett 2003] [Hlupic 2003] [April et al. 2006]. By simulating the to-be models, potential impacts and ridded risks of modifications made can be explored. In addition, if many change plans exist, simulation can help compare all proposals and provide evidence to managers' decision making in terms of whether selecting or rejecting a change initiative [Clauberg & Thomas 2013]. To this end, we select the HiLLS formalism to study CMMN case models, where HiLLS is a multi-purpose high level modeling language aiming at helping domain experts create DES models that can be studied from different perspectives, including simulation (DEVS), formal analysis (FM) and enactment (UML), respectively.

As illustrated in Figure 4-4 and Figure 4-5, to benefit from capabilities provided by HiLLS, we envision transforming CMMN models into HiLLS. A generic way to achieve this goal is to map the CMMN metamodel onto the HiLLS one. This way, any case model can be automatically converted into its HiLLS counterpart. In addition, we also propose a systematic way to generate an Experimental Frame (EF) for case models to conduct simulation experiments, where an EF defines the conditions under which a system is experimented with [Zeigler et al., 2000]. Consequently, case workers are able to observe their current as-is case model, identify and analyze problems, propose changes to make in the to-be model, and predict the performance of changes before implementation.

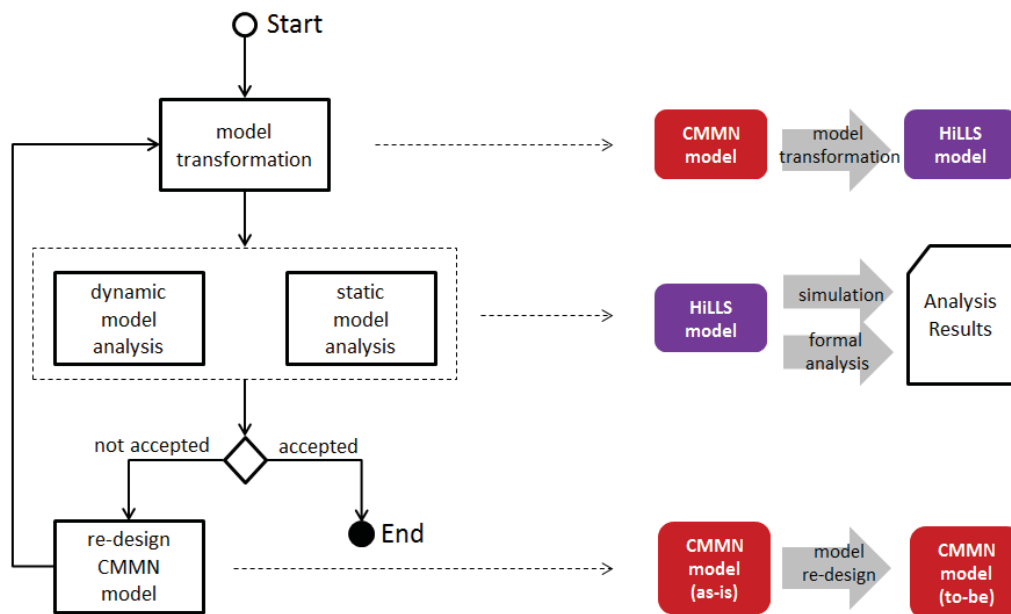


Figure 4-4: The workflow of CMI

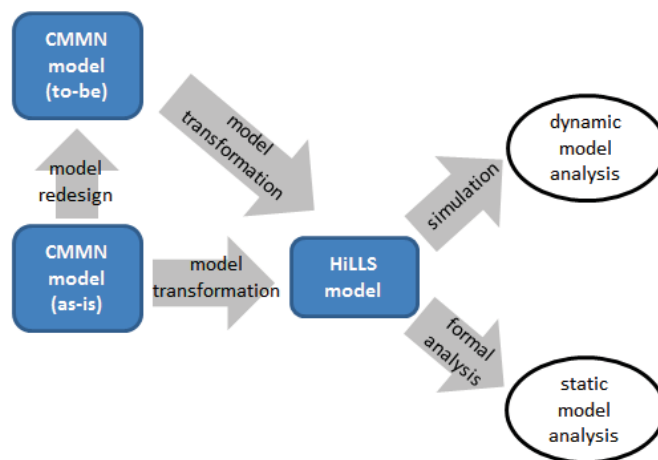


Figure 4-5: The whole picture of CMI

4.2.1 System Structure

As proposed in [Wang & Traoré 2014], the main idea for the model transformation is to turn a CMMN model into a HiLLS model considering both the system structure and the system behavior. In terms of system structure, as given in Table 4-1 below, we consider to convert the *RootStage* item – the *Stage* instance that is defined as the *casePlanModel* of the *Case* – to a *HSystem*. For the rest of the major elements, including *CaseFormItem*, *PlanItem*, and *DiscretionaryItem*, they will be transformed to *HClass* instances (Both *PlanItems* and *DiscretionaryItems* may refer to elements including *Stage*, *Task*, *Milestone* and *EventListener*. The former are defined at the design phase, and the latter are added at the planning phase at run-time. Later in this thesis if we mention a *Stage (Task)*, we mean a *PlanItem* referring to a *Stage (Task)*; likewise, if we say a discretionary *Stage (Task)*, we mean a *DiscretionaryItem* referring to a *Stage (Task)*). Consequently, we have one and only one *HSystem* generated for any *Case* model since a *Case* has one and only one *casePlanModel* defined, and many *HClasses*.

Table 4-1: The mappings of system structure

CMMN	HiLLS
<i>RootStage</i>	<i>HSystem</i>
<i>CaseFormItem</i>	<i>HClass</i>
<i>PlanItem</i>	<i>HClass</i>
<i>DiscretionaryItem</i>	<i>HClass</i>
<i>Sentry</i>	<i>HClass</i>

In addition, the *HSystem* has input and output ports generated, as specified in the following. The **EventIn** input port receives tuples (*element*, *event*), where *element* refers to either the *HSystem* or a *HClass*, and *event* refers to actions that can modify the state of an element (event belongs to the *eventSet*, where *eventSet* = {*close*, *complete*, *create*, *disable*, *enable*, *fault*, *manualStart*, *occur*, *reactivate*, *reenable*, *resume*, *start*, *suspend*, and *terminate*}). The **ParameterIn** input port receives the input parameters of the CMMN case (which are defined as *CaseFileItems* by users). Regarding the output ports, the **StatusOut** output port sends out the current status of the elements defined within the *Case* model (including *Active*, *Enabled*, *Disabled*, *Completed*, *Terminated*, *Failed*, *Suspended*, and *Closed*), and the **ParameterOut** output port sends out the *Case*'s output parameters (which are defined as *CaseFileItems* by users).

$$\text{InputPorts} = \{\text{EventIn}, \text{ParameterIn}\}$$

$$\text{OutputPorts} = \{\text{StatusOut}, \text{ParameterOut}\}$$

The containment relationship between all these CMMN elements will be preserved by the HiLLS composition relationship (which is borrowed from UML Class Diagram) among the *HSystem* and all *HClasses*. As specified in the rules below (we use *source* to refer to the CMMN element from which a HiLLS *HClass* is converted): a *HSystem*'s components will be those *HClasses* such that, each of which has a source defining as a *CaseFileItem*, or a *PlanItem* (*DiscretionaryItem*) that is directly contained by the *RootStage* (which is the source of the generated *HSystem*).

$$\begin{aligned}
 &HSystem.containingHClasses = \\
 &AllHClasses.select \left(i \left| \begin{array}{l} i.source = CaseFileItem \\ \vee i.source \in RootStage.planItems \\ \vee i.source \in RootStage.discretionaryItems \end{array} \right. \right)
 \end{aligned}$$

A *HClass* has components too, if and only if its source is a *PlanItem* (*DiscretionaryItem*) that referring to a *Stage*. Its components are those *HClasses* such that, each of which has a source defining as a *PlanItem* (*DiscretionaryItem*) that is directly contained by that *Stage*.

$$\begin{aligned}
 &HClass.containingHClasses = \\
 &AllHClasses.select \left(i \left| \begin{array}{l} i.source \in HClass.source.planItems \\ \vee i.source \in HClass.source.discretionaryItems \end{array} \right. \right) \leftrightarrow \\
 &HClass.source.definitionRef = Stage
 \end{aligned}$$

Following the rules specified above, Figure 4-6 gives an example of the system structure of a HiLLS model, which is converted from the *Case* model we used in Chapter 2 when introducing CMMN. As we can see, the *HSystem* ClaimsFileManagement (CFM) has two input ports created to receive events, and two output ports generated to send out events, respectively. Moreover, it contains 10 *HClasses* (which are Request (R), ResponsibilitiesIdentified (RI), CreateClaimsNotification (CCN), BaseInformationAttached (BIA), CreateClaims (CC), IdentifyResponsibilities (IR), ChangeResponsibilities (CR), RequestMissingInformation (RMI), Cancellation (C), and Deadline (D)), and each of which is converted from a CMMN basic element. The missing information of the compartment, System Behavior, will be specified in the next section.

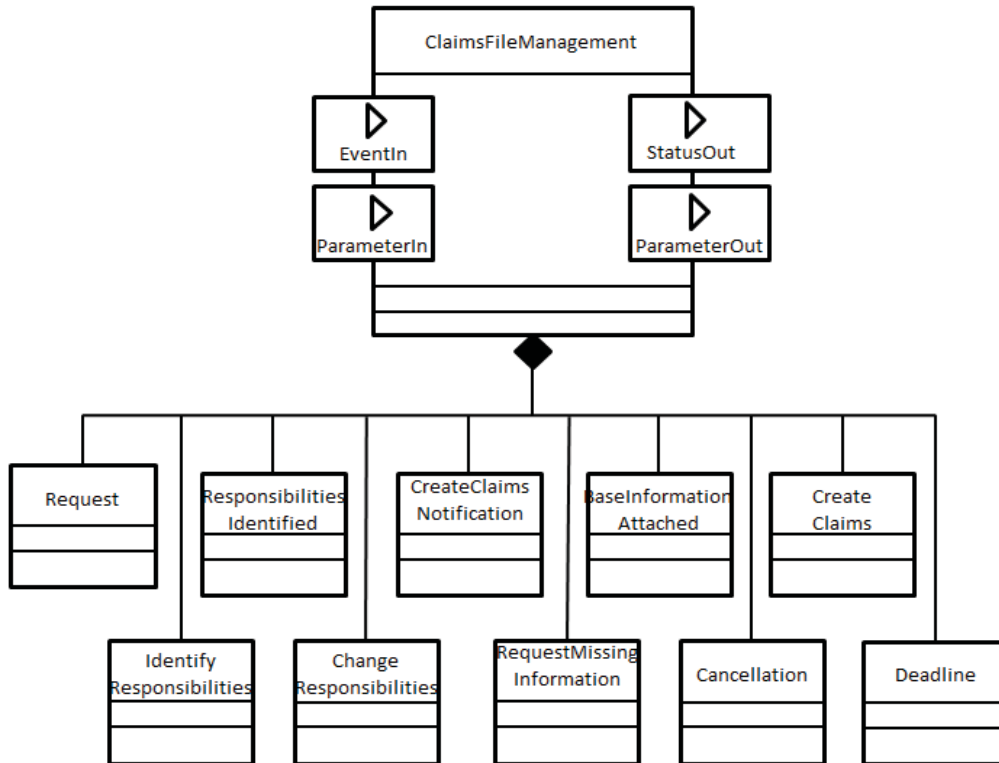


Figure 4-6: The system structure of a HiLLS model

4.2.2 System Behavior

The generation of the system behavior is a challenge in our CMMN to HiLLS model transformation since it is not simply a mapping between different elements. Due to this reason, we use mathematic formulas and tables together to specify how *Configurations* are generated, as well as how they transit from one to another.

4.2.2.1 Configuration

Due to the fact that each CMMN element can be in different states, as depicted in Table 4-2, each HiLLS configuration can be seen as a tuple (v_1, v_2, \dots, v_n) , where each $v_i = (item, state)$ is a state variable that represents the status (which is represented by *state*) of the i^{th} basic HiLLS element (which is represented by *item*) converted from its equivalence defined within the *Case* modeled. For instance, $v_i = (CreateClaims, Suspended)$ indicates that the *CreateClaims* item now is in the *Suspended* status. To ease the writing of the configuration, we use numbers to represent the items states.

Table 4-2: States of CMMN modeling elements

State	Reference Number	CMMN Elements
\emptyset	0	<i>RootStage/Stage/Task/Milestone/EventListener/CaseFormItem</i>
<i>Available</i>	1	<i>Stage/Task/Milestone/EventListener/CaseFormItem</i>
<i>Enabled</i>	2	<i>Stage/Task</i>
<i>Disabled</i>	3	<i>Stage/Task</i>
<i>Active</i>	4	<i>RootStage/Stage/Task</i>
<i>Completed</i>	5	<i>RootStage/Stage/Task/Milestone/EventListener</i>
<i>Terminated</i>	6	<i>RootStage/Stage/Task/Milestone/EventListener</i>
<i>Failed</i>	7	<i>RootStage/Stage/Task</i>
<i>Suspended</i>	8	<i>RootStage/Stage/Task/Milestone/EventListener</i>
<i>Closed</i>	9	<i>RootStage</i>
<i>Discarded</i>	10	<i>CaseFormItem</i>

Consequently, the *HSystem*'s complete configuration set will be defined as following, and it can be obtained by the combinational set of all elements' state sets.

$$ConfigurationSet = \{(item, state) \mid item \in HSystem \cup HClasses, state \in StateSet\} :$$

- $StateSet = \{0,4,5,6,7,8,9\} \leftrightarrow item.source = RootStage$
- $StateSet = \{1,2,3,4,5,6,7,8\} \leftrightarrow item.source \in \{Stage, Task\}$
- $StateSet = \{1,5,6,8\} \leftrightarrow item.source \in \{Milestone, EventListener\}$
- $StateSet = \{0,1,2,3,4,5,6,7,8\} \leftrightarrow item.source \in \left\{ \begin{array}{l} DiscretionaryStage, \\ DiscretionaryTask \end{array} \right\}$
- $StateSet = \{0,1,10\} \leftrightarrow item.source = CaseFormItem$

For example, as given in the case model presented in Figure 2-1, each element's state set is defined as follows:

- $S_{CFM} = \{(CFM, 0), (CFM, 4), (CFM, 5), (CFM, 6), (CFM, 7), (CFM, 8), (CFM, 9)\}$
- $S_R = \{(R, 0), (R, 1), (R, 10)\}$
- $S_{IR} = S_{CCN} = S_{CC} = \{(x, 1), (x, 2), (x, 3), (x, 4), (x, 5), (x, 6), (x, 7), (x, 8)\}$
 - $x \in \{IR, CCN, CC\}$
- $S_{RI} = S_{BIA} = S_C = S_D = \{(x, 1), (x, 5), (x, 6), (x, 8)\}$
 - $x \in \{RI, BIA, C, D\}$
- $S_{CR} = S_{RMI} = \{(x, 0), (x, 1), (x, 2), (x, 3), (x, 4), (x, 5), (x, 6), (x, 7), (x, 8)\}$
 - $x \in \{CR, RMI\}$

Therefore, its full configuration set can be obtained as

$$Configuration Set = S_{CFM} \times S_R \times S_{IR} \times S_{CCN} \times S_{CC} \times S_{RI} \times S_{BIA} \times S_C \times S_D \times S_{CR} \times S_{RMI}$$

As given in the HiLLS metamodel, a *Configuration* can be finite (where its $ta \in \mathbb{R}_{>0} - \{+\infty\}$), or passive (where its $ta = +\infty$). Consequently, we established rules to further categorize each *Configuration* into either finite or passive:

If a *Configuration* meets all the following conditions, then it is a *FiniteConfiguration*; otherwise, if at least one condition is not met, then it is a *PassiveConfiguration*:

- there exists at least one state variable $v_i = (item, state)$, such that
 - the source of this *item* refers to a *TimerEventListener*, AND
 - the source of this *item* is defined as the exit condition of the *RootStage* (i.e., the source of this *item* is one of the items that are defined as the sourceRef of the exit sentries of the *RootStage*), AND
 - the *state* is 1

These rules above can be formalized as following:

$$\forall f \in \text{FiniteConfigurationSet} :$$

$$\exists v_i = (item, state) \in f : (item.source.definitionType = \text{TimerEventListener}) \\ \wedge (item.source \in \text{RootStage.exitConditions}) \wedge (state = 1)$$

$$\forall p \in \text{PassiveConfigurationSet} :$$

$$\nexists v_i = (item, state) \in f : (item.source.definitionType = \text{TimerEventListener}) \\ \wedge (item.source \in \text{RootStage.exitConditions}) \wedge (state = 1)$$

Moreover, in order to observe the status of each item whenever a *Transition* occurs within the *HSystem*, a *TransientConfiguration* (where its $ta = 0$) is added for each *FiniteConfiguration* and *PassiveConfiguration*, where each pair of the transient and finite/passive *Configurations* have the same predicates defined. Assuming that we have a *PassiveConfiguration* A, and it receives a trigger and transits to a *FiniteConfiguration* B. In order to observe the status of items, a *TransientConfiguration* C is added which copies B's predicates. This way, as shown by Figure 4-7, *configuration* A will first transit to *configuration* C when it receives a trigger; then C will send out the output, and transit to the target *Configuration*, B. The formal rule of generating *TransientConfigurations* is specified in below.

$$\forall t \in \text{TransientConfigurationSet} :$$

$$(\exists f \in \text{FiniteConfigurationSet} : t.properties = \\ f.properties) \vee (\exists p \in \text{PassiveConfigurationSet} : t.properties = p.properties)$$

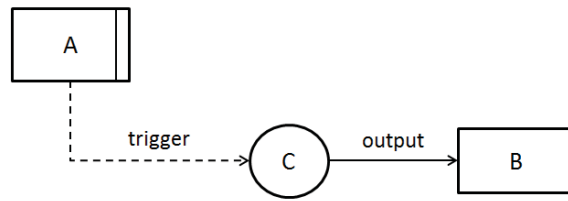


Figure 4-7: An example of configuration transitions

4.2.2.2 Configuration Transition

If the *HSystem* (h) stays in a *Configuration*, and it receives a trigger before its life-cycle (ta , which is defined by the *sojournTime* attribute) is expired, then an external configuration transition (ECT) will be observed, as illustrated in Figure 4-8. In this case, the system will first transit to a transient configuration (through an ECT) that corresponds to the target one; and then the system will output the status of items, and transit to the target (through an internal configuration transition (ICT)). Otherwise, an ICT will be observed: an output will be sent out, and the system will transit from a finite configuration to its target. In both situations, the target configuration to which the system will transit to is determined by the CMMN operational semantics given in [OMG 2014]. We will fully specify how a configuration transits to its target in the following, where we use $c = \{v_1, v_2, \dots, v_n\}$ to refer to the configuration before transitions, and $c' = \{v'_1, v'_2, \dots, v'_n\}$ to refer to the one after transitions.

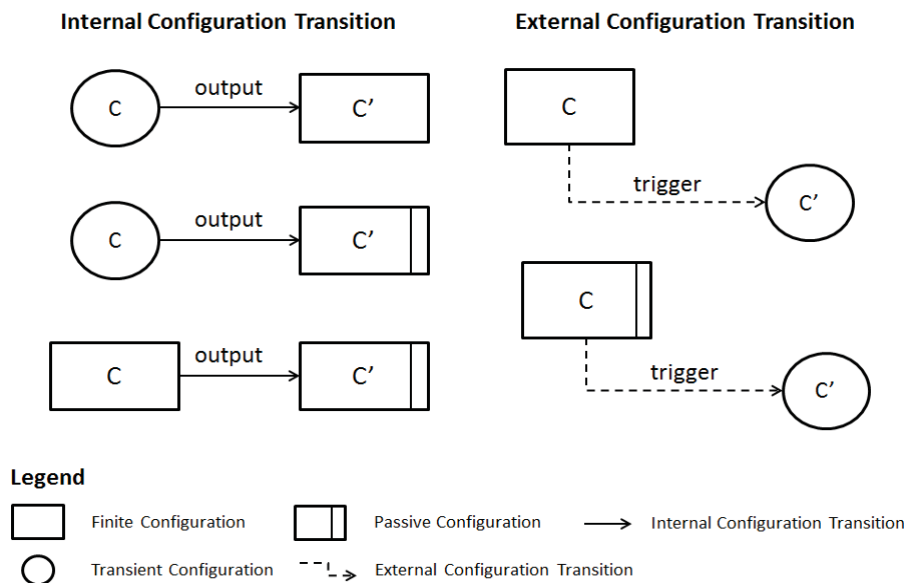


Figure 4-8: Different types of configuration transitions

Internal Configuration Transition

Within the $HSystem(h)$, as we mentioned above, there are three types of possible $ICTs$: $transient \rightarrow finite$ ($t2f$), $transient \rightarrow passive$ ($t2p$), and $finite \rightarrow passive$ ($f2p$). When a $t2f$ or a $t2p$ transition is observed, C and C' will have exactly the same predicates, and the system output the current status of items, as specified in the following rules. Figure 4-9 gives examples of the two types of $ICTs$ that are generated from our CMMN example case model.

$$ICT_{t2f} = c \xrightarrow{status} c' :$$

- ($c \in TransientConfigurationSet, c' \in FiniteConfigurationSet, v_i = v'_i (1 \leq i \leq n), status = c'$)

$$ICT_{t2p} = c \xrightarrow{status} c' :$$

- ($c \in TransientConfigurationSet, c' \in PassiveConfigurationSet, v_i = v'_i (1 \leq i \leq n), status = c'$)

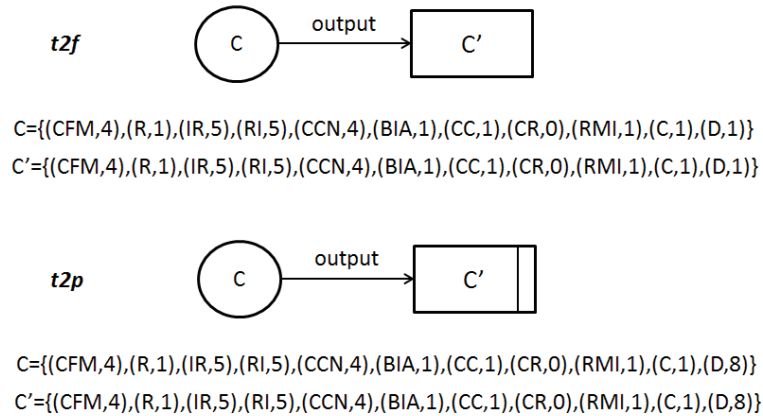


Figure 4-9: Examples of $t2f$ and $t2p$ configuration transitions

An $f2p$ transition will occur if the life-cycle of C is reached. The predicates of C' is defined as following, and Figure 4-10 gives us an example:

$$ICT_{f2p} = c \xrightarrow{status} c' :$$

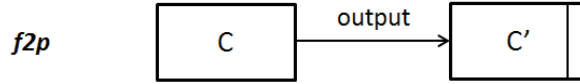
- ($c \in FiniteConfigurationSet, c' \in PassiveConfigurationSet: v'_i = f(v_i)(1 \leq i \leq n), status = c'$) :

- if $v_i.item.source.definitionType \in \{Stage, Task, Milestone, UserEventListener\}$,

$$f(v_i) = \begin{cases} (v_i.item, 5), & v_i.item.state = 5 \\ (v_i.item, 6), & v_i.item.state \neq 5 \end{cases}$$

- if $v_i.item.source.definitionType = TimerEventListener$,

$$f(v_i) = \begin{cases} (v_i.item, 5), & v_i.item.source \in RootStage.exitConditions \\ (v_i.item, 6), & v_i.item.source \notin RootStage.exitConditions \end{cases}$$



$C = \{(CFM,4),(R,1),(IR,5),(RI,5),(CCN,4),(BIA,1),(CC,1),(CR,0),(RMI,1),(C,1),(D,1)\}$

$C' = \{(CFM,6),(R,1),(IR,5),(RI,5),(CCN,6),(BIA,6),(CC,6),(CR,0),(RMI,6),(C,6),(D,5)\}$

Figure 4-10: An example of *f2p* configuration transitions

External Configuration Transition

Basically, an *ECT* will occur if the *HSystem* receives a trigger before the life-cycle of the configuration it is currently stays in is expired. According to different triggers, c' will have a different value. Figure 4-11 gives an example of *ECT*, where C is a passive configuration, the trigger is $(CFM, suspend)$, and the predicates of C' is computed following the rules specified below.

$$ECT = c \xrightarrow{trigger} c' :$$

- $\left(\begin{array}{l} c \in FiniteConfigurationSet \wedge PassiveConfigurationSet, \\ c' \in TransientConfigurationSet, \\ trigger \in \{(item, event) \mid item \in HSystem \cup HClasses, event \in eventSet\}, \\ v_i' = (v_i.item, f(trigger))(1 \leq i \leq n) \end{array} \right) :$

1) *if trigger.item.source = RootStage*

Table 4-3: Transformation rules of ECT – Part 1

$s(v)^1$	trigger	autoC ²	$s(v')$
0	<i>(self</i> ³ , <i>create)</i>	* ⁴	4
4 ⁵	<i>(self, complete)</i>	<i>False</i>	5
	<i>(self, terminate)</i>	*	6
	<i>(self, fault)</i>	*	7
	<i>(self, suspend)</i>	*	8
5/6/7/8	<i>(self, reactivate)</i>	*	4
	<i>(self, close)</i>	*	9

¹ We use $s(v)$ to represent the value of the state parameter of v_i , and $s(v')$ to represent the value of $f(\text{trigger})$.

² We use *autoC* to represent the *autoComplete* attribute of the item.

³ If $v_i.\text{item}=\text{trigger.item}$, then we use *self* to represents this item; otherwise, we use *other*.

⁴ The symbol “*” means that the value of the attribute could be any (e.g., *True*, or *False*).

The symbol “–” means that the value of the attribute is null.

⁵ If at any time, (1) $v_i = 4$, (2) all its required children are in the state of 3/5/6/7, and (3) all its non-required children are not in the state of 4, then $v_i' = 5$.

2) *if trigger.item ∈ {stage, task}*

Table 4-4: Transformation rules of ECT – Part 2

$s(v)$	trigger	P/D	HES ¹	ES ²	Manual ³	autoC	$s(v')$
0	(self, create)	D	True	False	*	*	1
	(self, create)	D	True	True	True	*	2
	(self, create)	D	True	True	False	*	4
	(self, create)	D	False	–	True	*	2
	(self, create)	D	False	–	False	*	4
	(parent ⁴ , create)	P	True	False	*	*	1
	(parent, create)	P	True	True	True	*	2
	(parent, create)	P	True	True	False	*	4
	(parent, create)	P	False	–	True	*	2
	(parent, create)	P	False	–	False	*	4
1	(other, *)	*	True	True	True	*	2
	(other, *)	*	True	True	False	*	4
2	(self, disable)	*	*	*	True	*	3
	(self, manualStart)	*	*	*	True	*	4
	(parent, terminate)	*	*	*	True	*	6
	(parent, suspend)	*	*	*	True	*	8
3	(self, reenable)	*	*	*	True	*	2
	(parent, terminate)	*	*	*	True	*	6
	(parent, suspend)	*	*	*	True	*	8
4	(self, complete)	*	*	*	*	False	5
	(self, terminate)	*	*	*	*	*	6
	(parent, terminate)	*	*	*	*	*	6
	(self, fault)	*	*	*	*	*	7
	(self, suspend)	*	*	*	*	*	8
	(parent, suspend)	*	*	*	*	*	8
7	(self, reactivate)	*	*	*	*	*	4
	(parent, terminate)	*	*	*	*	*	6
8	(self, resume)	*	*	*	*	*	4
	(parent, resume)	*	*	*	*	*	H
	(parent, reactivate)						
	(parent, terminate)	*	*	*	*	*	6
5/6	(parent, reactivate)	P	True	False	*	*	1
	(parent, reactivate)	P	True	True	True	*	2
	(parent, reactivate)	P	True	True	False	*	4
	(parent, reactivate)	P	False	–	True	*	2
	(parent, reactivate)	P	False	–	False	*	4

¹ HES (hasEntrySentry) refers to if the item has any *entry sentry* defined.

² ES (entrySentry) refers to if the condition the item's *entry sentry* specified is met or not.

³ We use *Manual* to represent the *manualActivationRule* attribute of the item.

⁴ We use *parent* to represent the *Stage* item that consists of the source item v_i refers to.

3) if $trigger.item \in \{milestone, eventListener\}$

Table 4-5: Transformation rules of ECT – Part 3

$s(v)$	trigger	HST ¹	ST ²	HES	ES	$s(v')$
0	(parent, create)	True	True	–	–	1
1	(other, any)	–	–	–	–	5
	(self, occur)	–	–	–	–	5
1	(self, suspend)	*	*	True	False	8
	(parent, suspend)	*	*	True	False	8
1	(self, suspend)	*	*	False	–	8
	(parent, suspend)	*	*	False	–	8
1	(self, terminate)	*	*	True	False	6
	(parent, terminate)	*	*	True	False	6
1	(self, terminate)	*	*	False	–	6
	(parent, terminate)	*	*	False	–	6
8	(other, any)	*	*	True	True	5
	(self, resume)	*	*	*	*	1
8	(parent, resume)	*	*	*	*	1
	(self, terminate)	*	*	*	*	6
	(parent, terminate)	*	*	*	*	6

¹ HST (hasStartTrigger) refers to if the item has any *start trigger* defined.

² ST (startTrigger) refers to if the condition the item's *start trigger* specified is met or not.

4) *trigger.item = caseFileItem*

Table 4-6: Transformation rules of ECT – Part 4

<i>s(v)</i>	trigger	<i>s(v')</i>
0	<i>(self, create)</i>	1
1	<i>(self, addChild)</i>	1
	<i>(self, removeChild)</i>	
	<i>(self, update)</i>	
	<i>(self, replace)</i>	
	<i>(self, addReference)</i>	
	<i>(self, removeReference)</i>	
	<i>(self, delete)</i>	10

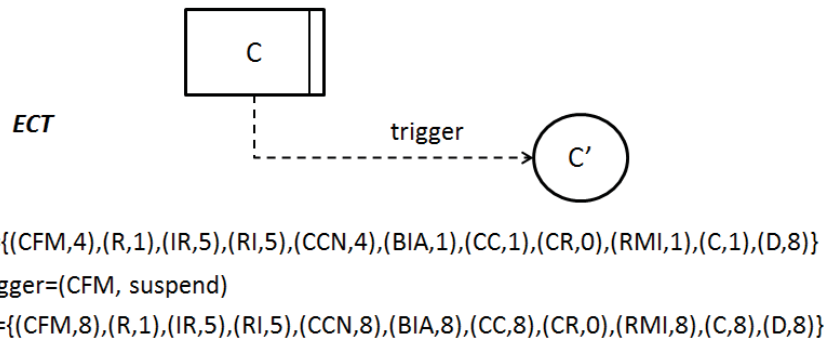


Figure 4-11: An example of *ECT*

As we have explained in the beginning of this chapter before, the translations between a CMMN model and its targeted HiLLS model are not simply n-n mappings between their meta-models. Especially for the configurations and the transitions between them, the rules must take into account several conditions, e.g., each element's state, sentry state, etc. The rules of configuration transitions we have defined above serve as the pre-conditions when executing HiLLS model simulations.

4.3 Experimental Frame

Once a HiLLS case model is obtained, a simulation EF (which is also a HiLLS model) should be established so that case workers are able to run simulation experiments. An EF defines the environment with which a system interacts with, in simulation experiments. Basically, an EF is considered as a system that is coupled with the system of interest. Generally, case workers are domain experts; their work

mainly relates to the construction and manipulation of case models using CMMN. They are not directly dealing with simulated models specified by HiLLS. As a result, it is often difficult and error-prone for them to establish an EF for the purpose of conducting simulation experiments of case models. Moreover, even though they can do this by themselves or receiving help from simulation experts, they are still facing another inconvenience: each time when they have a case model to simulate, or they want to change the simulation environment for a different purpose, they need to create a new EF, manually from scratch.

Due to this fact, we propose here a systematic way to generate EFs, in order to ease the process of simulating case models for case workers. In our approach, the EF contains two parts: one is to generate and feed data into the system of interest (we name this part EventGenerator), the other one collects and analyzes the results observed (we call this part Analyzer), as given in Figure 4-12. On one hand, an EF can be generated in a semi-automatic manner, as shown in Figure 4-13. On the other hand, such an EF is easy to configure so that different simulation environments can be built according to different requirements and research purposes, without having experience from the M&S domain.

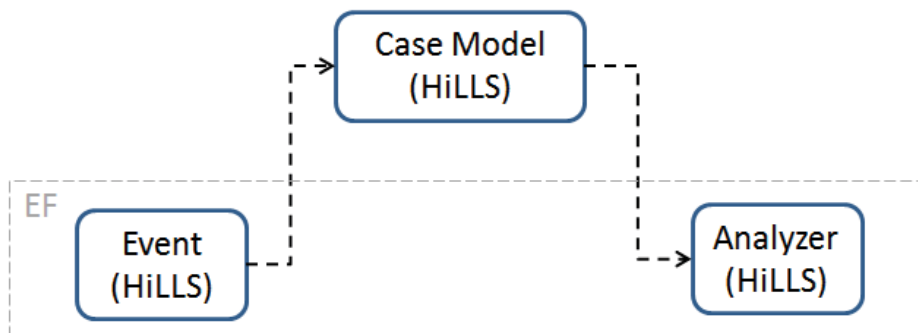


Figure 4-12: The couplings between different HiLLS models

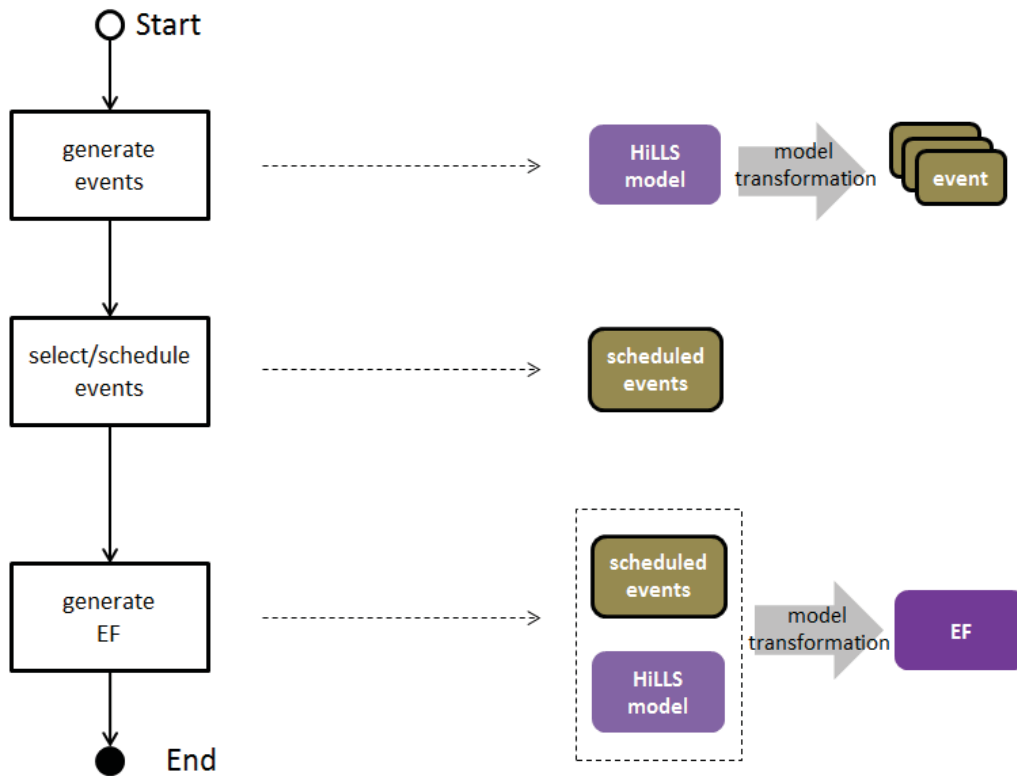


Figure 4-13: The flow chart of generating EFs

Case → EventGenerator

To start with, the whole Case will be transformed to a HiLLS system named EventGenerator. This EventGenerator generates and feeds events to the HiLLS case model. The formal transformation rules are given below.

Port

$$OutputPorts = \{EventOut\}$$

This EventGenerator has only one port: an output port named EventOut, which sends events to the HiLLS case model.

HComponents

$$Components = \{H_1, H_2, \dots, H_n\}$$

Each $H \in hComponents$ is a HSystem, which is converted from the item defined within the Case model and serves as an item event generator. Based on the original CMMN element before transformation, the obtained HSystem has different structure and behavior. We will explicitly define the transformation rules for each type of CMMN element.

4.3.1 RootStage \rightarrow HSystem

4.3.1.1 Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

The HSystem converted from the RootStage has two ports: one for receiving incoming events, and one for sending events to other HSystems.

4.3.1.2 HComponents

$$HComponents = \left\{ \begin{array}{l} H_{create}, H_{complete}, H_{suspend}, H_{terminate}, \\ H_{fault}, H_{reactivate}, H_{close} \end{array} \right\}$$

$$\leftrightarrow Self.autoComplete = false$$

$$HComponents = \left\{ \begin{array}{l} H_{create}, H_{suspend}, H_{terminate}, \\ H_{fault}, H_{reactivate}, H_{close} \end{array} \right\} \leftrightarrow Self.autoComplete = true$$

$\forall h \in hComponents : h$ is a HSystem, and it corresponds to a RootStage event

If the autoComplete attribute of the RootStage is set to False, then it will contain 7 atomic HSystems, each of which represents a RootStage event (i.e., *create*, *complete*, *suspend*, *terminate*, *fault*, *reactivate* and *close*). On the contrary, if the autoComplete attribute of the RootStage is set to true, then this HSystem will have 6 atomic HSystems instead of seven, where the atomic HSystem representing the *complete* event is not considered. The detail of each atomic HSystem is given below.

Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Each atomic RootStage event HSystem has two ports: one for receiving incoming events, and one for sending events to other HSystems.

Configurations

$Configurations = \{passive, active\}$, such that

$$ta_{passive} = +\infty$$

$$ta_{active} = 0 \leftrightarrow self.event = create$$

$$ta_{active} = \sigma \leftrightarrow self.event \neq create$$

Each atomic RootStage HSystem has two configurations: passive and active. Based on the event type of each HSystem, the time advance function is different: if the event is *created* then $ta = 0$; otherwise $ta = \sigma$.

Transitions

$Transitions = \{ICT, ECT\}$, such that

$$ICT: c \xrightarrow{outputEvent} c', \text{ such that}$$

- $c = active, c' = passive, outputEvent = (self, self.event)$

$$ECT: c \xrightarrow{trigger} c' \leftrightarrow self.event \neq create, \text{ such that}$$

- $(c = passive, c' = active, trigger \in \{(self, create), (self, reactivate)\}) \leftrightarrow self.event \in \{complete, suspend, terminate, fault\}$
- $(c = passive, c' = active, trigger \in \{(self, complete), (self, suspend), (self, terminate), (self, fault)\}) \leftrightarrow self.event \in \{reactivate, close\}$

Essentially, if a HSystem stays in the *active* state, and its ta reaches then an ECT will be observed. It will transit to the passive state, and send out an event to other HSystems: the one representing the *create* event will send out $(RootStage, create)$, the one representing the *complete* event will send out $(RootStage, complete)$, the one representing the *terminate* event will send out $(RootStage, terminate)$, etc.

However, if a HSystem stays in the *passive* state, and the event this HSystem

concerning belongs to $\{complete, suspend, terminate, fault\}$ and it receives either a creation or a reactivation event, then an ECT will be observed. On the other hand, if the event his HSystem concerning belongs to $\{create, reactivate\}$ and it receives an event belonging to $\{complete, suspend, terminate, fault\}$, an ECT will also be observed.

4.3.1.3 Configurations

$PassiveConfigurations = EOC \cup IC$, such that

$EOC = \{(s, t) \mid s \in hComponents.outputPorts, t = EventOut\}$

$$IC = \left\{ (s, t) \left| \begin{array}{l} s \in hComponents.select(i \mid i.event \in \{create, reactivate\}), \\ t \in hComponents.select(i \mid i.event \in \{complete, suspend, \\ terminate, fault\}) \\ \vee \\ s \in hComponents.select(i \mid i.event \in \{complete, suspend, \\ terminate, fault\}), \\ t \in hComponents.select(i \mid i.event \in \{reactivate, close\}) \end{array} \right. \right\} \leftrightarrow$$

$RootStage.autoComplete = flase$

$IC =$

$$\left\{ (s, t) \left| \begin{array}{l} s \in hComponents.select(i \mid i.event \in \{create, reactivate\}), \\ t \in hComponents.select(i \mid i.event \in \{suspend, terminate, fault\}) \\ \vee \\ s \in hComponents.select(i \mid i.event \in \{suspend, terminate, fault\}), \\ t \in hComponents.select(i \mid i.event \in \{reactivate, close\}) \end{array} \right. \right\} \leftrightarrow$$

$RootStage.autoComplete = true$

Note: (s : source; t : target)

In terms of EOC, each atomic event HSystem's output port connects with the output port of the EventGenerator translated from the whole case (which we have introduced above). In terms of IC, depending on the Boolean value of the $autoComplete$ attribute of the RootStage, the total number of atomic event HSystems will be different, so is the total number of the internal couplings.

4.3.2 Stage/Task \rightarrow HSystem

Here we will explain how a Stage, or a Task element translates to a coupled HSystem which contains a set of atomic HSystems, each of which generates the Stage- or the Task-related events. Please note that

- *Stage* refers to a *planItem* or a *discretionaryItem* referring to a *Stage*
- *Task* refers to a *planItem* or a *discretionaryItem* referring to a *Task*
- *Item* refers to the *Stage* or the *Task* element
- *mar* refers to the *Item*'s *ManualActivationRule*
- *ac* refers to the *Item*'s *AutoComplete* attribute
 - since a *Task* has no *AutoComplete* attribute, so we consider that *Task.ac=false*
 - if *Item.ac=true*, then *Item=Stage*; otherwise it could be either a *Stage* or a *Task*
- *entry* refers to the *entrySentry* of the *Item*
- *q* refers to the queue of received events
- *trigger* refers to the element the *Item*'s *entrySentry* referring to
 - *trigger.event* can be either *complete* or *terminate*

4.3.2.1 Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Like the HSystem which is converted from the RootStage, each HSystem transformed from a Stage/Task element has also two ports: one for receiving incoming events, and one for sending events to other HSystems.

4.3.2.2 HComponents

According to the different values of related attributes (i.e., item type (*planItem* or *discretionaryItem*), *manualActivationRule*, *autoComplete* and *entrySentry*), the contained atomic event HSystems are various, as specified below:

$$HComponents = \left\{ \begin{array}{l} H_{disable}, H_{reenable}, H_{manualStart}, H_{complete}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow \\ Item.type = planItem \wedge Item.mar = true \wedge Item.ac = false$$

$$HComponents = \left\{ \begin{array}{l} H_{disable}, H_{reenable}, H_{manualStart}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow Item.type = \\ planItem \wedge Item.mar = true \wedge Item.ac = true$$

$$HComponents = \left\{ \begin{array}{l} H_{complete}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow Item.type = \\ planItem \wedge Item.mar = false \wedge Item.ac = false$$

$$HComponents = \left\{ \begin{array}{c} H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow Item.type = \\ planItem \wedge Item.mar = false \wedge Item.ac = true$$

$$HComponents = \left\{ \begin{array}{c} H_{disable}, H_{reenable}, H_{manualStart}, H_{complete}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume}, H_{create} \end{array} \right\} \leftrightarrow \\ Item.type = discretionaryItem \wedge Item.mar = true \wedge Item.ac = false$$

$$HComponents = \left\{ \begin{array}{c} H_{disable}, H_{reenable}, H_{manualStart}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume}, H_{create} \end{array} \right\} \leftrightarrow Item.type = \\ discretionaryItem \wedge Item.mar = true \wedge Item.ac = true$$

$$HComponents = \left\{ \begin{array}{c} H_{create}, H_{complete}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow Item.type = \\ discretionaryItem \wedge Item.mar = false \wedge Item.ac = false$$

$$HComponents = \left\{ \begin{array}{c} H_{create}, H_{suspend}, \\ H_{terminate}, H_{fault}, H_{reactivate}, H_{resume} \end{array} \right\} \leftrightarrow Item.type = \\ discretionaryItem \wedge Item.mar = false \wedge Item.ac = true$$

$\forall h \in hComponents$: h is an atomic $HSystem$, and it corresponds to an $Item$ event ($create$, $complete$, $suspend$, etc.). The detailed specifications of each atomic $HSystem$ are given below.

Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Each atomic event $HSystem$ has two ports: one for receiving incoming events, and one for sending events to other $HSystems$.

Configurations

$Configurations = \{passive, active\}$, such that

$$ta_{passive} = +\infty$$

$$ta_{active} = \sigma$$

Each atomic event $HSystem$ has two essential configurations: $passive$ and $active$.

The *ta* of the *passive* configuration equals to positive infinity, and the *ta* of the *active* configuration will be assigned by case workers later.

Transitions

Here we will detail all the possible configuration transitions of each Stage/Task event related atomic HSystem, considering different values of item attributes we have introduce above (where *c* refers to the source configuration, and *c'* refers to the target configuration).

Transitions = {*ICT*, *ECT*}, such that

ICT: $c \xrightarrow{\text{outputEvent}} c'$, such that

- $(c = \text{active}, c' = \text{passive}, \text{outputEvent} = (\text{self}, \text{self.event})) \leftrightarrow q = \text{empty}$
- $(c = \text{active}, c' = \text{active}, \text{outputEvent} = (\text{self}, \text{self.event})) \leftrightarrow q \neq \text{empty}$

ECT: $c \xrightarrow{\text{trigger}} c'$, such that

- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self.parent}, \text{create}), \\ (\text{self.parent}, \text{manualStart}), \\ (\text{self.parent}, \text{resume}), \\ (\text{self.parent}, \text{reactivate}) \end{array} \right\} \right) \leftrightarrow$
 $\text{Item} = \text{discretionaryItem} \wedge \text{self.event} \in \{\text{create}\} \wedge \text{Item.entry} = \text{null}$
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \left(\begin{array}{l} \text{Item.trigger}, \\ \text{Item.trigger.event} \end{array} \right) \right\} \right) \leftrightarrow \text{Item} =$
 $\text{discretionaryItem} \wedge \text{self.event} \in \{\text{create}\} \wedge \text{Item.entry} \neq \text{null}$
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self.parent}, \text{create}), \\ (\text{self.parent}, \text{manualStart}), \\ (\text{self.parent}, \text{resume}), \\ (\text{self.parent}, \text{reactivate}) \end{array} \right\} \right) \leftrightarrow$
 $\text{Item} = \text{planItem} \wedge \text{self.event} \in \{\text{disable}, \text{manualStart}\} \wedge \text{Item.mar} =$
 $\text{true} \wedge \text{Item.entry} = \text{null}$
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \left(\begin{array}{l} \text{Item.trigger}, \\ \text{Item.trigger.event} \end{array} \right) \right\} \right) \leftrightarrow \text{Item} =$
 $\text{planItem} \wedge \text{self.event} \in$
 $\{\text{disable}, \text{manualStart}\} \wedge \text{Item.mar} = \text{true} \wedge \text{Item.entry} \neq \text{null}$
- $(c = \text{passive}, c' = \text{active}, \text{trigger} \in \{(self, create)\}) \leftrightarrow \text{Item} =$

$discretionaryItem \wedge self.event \in \{disable, manualStart\} \wedge Item.mar = true$

- $(c = passive, c' = active, trigger \in \{(self, disable)\}) \leftrightarrow self.event \in \{reenable\} \wedge Item.mar = true$

- $\left(c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self, manualStart), \\ (self, reactivate), \\ (self, resume) \end{array} \right\} \right) \leftrightarrow$

$Item.type =$

$Stage \wedge self.event \in \{complete, suspend, terminate, fault\} \wedge Item.mar = true \wedge Item.ac = false$

- $\left(c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self, manualStart), \\ (self, reactivate), \\ (self, resume) \end{array} \right\} \right) \leftrightarrow$

$Item.type =$

$Stage \wedge self.event \in \{suspend, terminate, fault\} \wedge Item.mar = true \wedge Item.ac = true$

- $\left(c = passive, c' = active, trigger \in$

$\left. \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, reactivate), (self, resume) \end{array} \right\} \right) \leftrightarrow Item.type = Stage \wedge Item =$

$planItem \wedge self.event \in$

$\{complete, suspend, terminate, fault\} \wedge Item.mar = false \wedge Item.ac = false$

- $\left(c = passive, c' = active, trigger \in$

$\left. \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, reactivate), (self, resume) \end{array} \right\} \right) \leftrightarrow Item.type = Stage \wedge Item =$

$planItem \wedge self.event \in \{suspend, terminate, fault\} \wedge Item.mar = false \wedge Item.ac = true$

- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self}, \text{create}), \\ (\text{self}, \text{reactivate}), \\ (\text{self}, \text{resume}) \end{array} \right\} \right) \leftrightarrow$
Item.type =
Stage \wedge *Item = discretionaryItem* \wedge *self.event* \in
 $\{\text{complete}, \text{suspend}, \text{terminate}, \text{fault}\} \wedge$ *Item.mar = false* \wedge *Item.ac = false*
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self}, \text{create}), \\ (\text{self}, \text{reactivate}), \\ (\text{self}, \text{resume}) \end{array} \right\} \right) \leftrightarrow$
Item.type =
Stage \wedge *Item = discretionaryItem* \wedge *self.event* \in
 $\{\text{suspend}, \text{terminate}, \text{fault}\} \wedge$ *Item.mar = false* \wedge *Item.ac = true*
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self}, \text{manualStart}), \\ (\text{self}, \text{reactivate}), \\ (\text{self}, \text{resume}) \end{array} \right\} \right) \leftrightarrow$
Item.type =
Task \wedge *self.event* $\in \{\text{complete}, \text{suspend}, \text{terminate}, \text{fault}\} \wedge$ *Item.mar = true*
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self.parent}, \text{create}), \\ (\text{self.parent}, \text{manualStart}), \\ (\text{self.parent}, \text{resume}), \\ (\text{self.parent}, \text{reactivate}), \\ (\text{self}, \text{reactivate}), (\text{self}, \text{resume}) \end{array} \right\} \right) \leftrightarrow$ *Item.type = Task* \wedge *Item = planItem* \wedge *self.event* \in
 $\{\text{complete}, \text{suspend}, \text{terminate}, \text{fault}\} \wedge$ *Item.mar = false*
- $\left(c = \text{passive}, c' = \text{active}, \text{trigger} \in \left\{ \begin{array}{l} (\text{self}, \text{create}), \\ (\text{self}, \text{reactivate}), \\ (\text{self}, \text{resume}) \end{array} \right\} \right) \leftrightarrow$
Item.type =
Task \wedge *Item = discretionaryItem* \wedge *self.event* \in
 $\{\text{complete}, \text{suspend}, \text{terminate}, \text{fault}\} \wedge$ *Item.mar = false*
- $(c = \text{passive}, c' = \text{active}, \text{trigger} \in \{(\text{self}, \text{fault})\}) \leftrightarrow$ *self.event* \in
 $\{\text{reactivate}\}$
- $(c = \text{passive}, c' = \text{active}, \text{trigger} \in \{(\text{self}, \text{suspend})\}) \leftrightarrow$ *self.event* \in
 $\{\text{resume}\}$

- $$\bullet \left(c = active, c' = active, trigger \in \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, manualStart), \\ (self, reactivate), (self, resume) \end{array} \right\} \leftrightarrow Item.type = Task \wedge self.event \in \{complete\} \wedge self.mar = true \right)$$
- $$\bullet \left(c = active, c' = active, trigger \in \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, reactivate), \\ (self, resume) \end{array} \right\} \leftrightarrow \right.$$

$$Item.type = Task \wedge Item = planItem \wedge self.event \in \{complete\} \wedge self.mar = false$$
- $$\bullet \left(c = active, c' = active, trigger \in \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, create), \\ (self, reactivate), \\ (self, resume) \end{array} \right\} \leftrightarrow \right.$$

$$Item.type = Task \wedge Item = discretionaryItem \wedge self.event \in \{complete\} \wedge self.mar = false$$

4.3.2.3 Configurations

The configurations of the event generator HSystem converted from a Stage/Task item define the couplings of the atomic event HSystems obtained.

$PassiveConfigurations = EOC \cup EIC \cup IC$, such that

- $EOC = \{(s, t) \mid s \in hComponents.outputPorts, t = EventOut\}$
- $EIC =$

- $$\left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in h\text{Components}.select(i|i.event \in \{\text{disable}, \text{manualStart}\}).inputPorts \end{array} \right. \right\} \leftrightarrow$$

$$\text{Item.type} = \text{planItem} \wedge \text{Item.mar} = \text{true}$$
- $$\text{EIC} =$$

$$\left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in h\text{Components}.select\left(i \left| \begin{array}{l} i.event \in \{\text{complete}, \text{suspend}, \\ \text{terminate}, \text{fault}\} \end{array} \right. \right).inputPorts \end{array} \right. \right\} \leftrightarrow$$

$$\text{Item.type} = \text{planItem} \wedge \text{Item.mar} = \text{false} \wedge \text{Item.ac} = \text{false}$$
- $$\text{EIC} =$$

$$\left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in h\text{Components}.select\left(i \left| \begin{array}{l} \text{suspend}, \\ \text{terminate}, \text{fault} \end{array} \right. \right).inputPorts \end{array} \right. \right\} \leftrightarrow$$

$$\text{Item.type} = \text{planItem} \wedge \text{Item.mar} = \text{false} \wedge \text{Item.ac} = \text{true}$$
- $$\text{EIC} = \left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in h\text{Components}.select(i|i.event = \text{create}).inputPorts \end{array} \right. \right\} \leftrightarrow$$

$$\text{Item.type} = \text{discretionaryItem}$$
- $$\text{IC} =$$

$$\left\{ (s, t) \left| \begin{array}{l} s \in h\text{Components}.select(i|i.event = \text{disable}).outputPorts, \\ t \in h\text{Components}.select(i|i.event = \text{reenable}).inputPorts, \\ \vee \\ s \in h\text{Components}.select(i|i.event = \text{reenable}).outputPorts, \\ t \in h\text{Components}.select\left(i \left| \begin{array}{l} \text{disable}, \\ \text{manualStart} \end{array} \right. \right).inputPorts, \\ \vee \\ s \in h\text{Components}.select\left(i \left| \begin{array}{l} i.event \in \\ \{\text{manualStart}, \\ \text{resume}, \\ \text{reactivate}\} \end{array} \right. \right).outputPorts, \\ t \in h\text{Components}.select\left(i \left| \begin{array}{l} i.event \in \\ \{\text{complete}, \\ \text{suspend}, \\ \text{terminate}, \\ \text{fault}\} \end{array} \right. \right).inputPorts, \\ \vee \\ s \in h\text{Components}.select(i|i.event = \text{suspend}).outputPorts, \\ t \in h\text{Components}.select(i|i.event = \text{resume}).inputPorts, \\ \vee \\ s \in h\text{Components}.select(i|i.event = \text{fault}).outputPorts, \\ t \in h\text{Components}.select(i|i.event = \text{reactivate}).inputPorts \end{array} \right. \right\} \leftrightarrow$$

$$\text{Item.type} = \text{planItem} \wedge \text{Item.mar} = \text{true} \wedge \text{Item.ac} = \text{false}$$
- $$\text{IC} =$$

$$\left\{ (s, t) \left[\begin{array}{l}
s \in hComponents.select(i|i.event = disable).outputPorts, \\
t \in hComponents.select(i|i.event = reenable).inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = reenable).outputPorts, \\
t \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} disable, \\ manualStart \end{array} \right\} \right\}.inputPorts, \\
\vee \\
s \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} manualStart, \\ resume, \\ reactivate \end{array} \right\} \right\}.outputPorts, \\
t \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} suspend, \\ terminate, \\ fault \end{array} \right\} \right\}.inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = suspend).outputPorts, \\
t \in hComponents.select(i|i.event = resume).inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = fault).outputPorts, \\
t \in hComponents.select(i|i.event = reactivate).inputPorts
\end{array} \right. \right\} \leftrightarrow$$

$Item.type = planItem \wedge Item.mar = true \wedge Item.ac = true$

- $IC =$

$$\left\{ (s, t) \left[\begin{array}{l}
s \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} resume, \\ reactivate \end{array} \right\} \right\}.outputPorts, \\
t \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} complete, \\ suspend, \\ terminate, \\ fault \end{array} \right\} \right\}.inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = suspend).outputPorts, \\
t \in hComponents.select(i|i.event = resume).inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = fault).outputPorts, \\
t \in hComponents.select(i|i.event = reactivate).inputPorts
\end{array} \right. \right\} \leftrightarrow$$

$Item.type = planItem \wedge Item.mar = false \wedge Item.ac = false$

- $IC =$

$$\left\{ (s, t) \left[\begin{array}{l}
s \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} resume, \\ reactivate \end{array} \right\} \right\}.outputPorts, \\
t \in hComponents.select\left(i \left| i.event \in \left\{ \begin{array}{l} suspend, \\ terminate, \\ fault \end{array} \right\} \right\}.inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = suspend).outputPorts, \\
t \in hComponents.select(i|i.event = resume).inputPorts, \\
\vee \\
s \in hComponents.select(i|i.event = fault).outputPorts, \\
t \in hComponents.select(i|i.event = reactivate).inputPorts
\end{array} \right. \right\} \leftrightarrow$$

$Item.type = planItem \wedge Item.mar = false \wedge Item.ac = true$

• $IC =$

$$\left((s, t) \left[\begin{array}{l} s \in hComponents.select(i|i.event = disable).outputPorts, \\ t \in hComponents.select(i|i.event = reenable).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event \in \{create, reenable\}).outputPorts, \\ t \in hComponents.select(i|i.event \in \{disable, manualStart\}).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event \in \{manualStart, resume, reactivate\}).outputPorts, \\ t \in hComponents.select(i|i.event \in \{complete, suspend, terminate, fault\}).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event = suspend).outputPorts, \\ t \in hComponents.select(i|i.event = resume).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event = fault).outputPorts, \\ t \in hComponents.select(i|i.event = reactivate).inputPorts \end{array} \right] \right) \leftrightarrow$$

$Item.type = discretionaryItem \wedge Item.mar = true \wedge Item.ac = false$

• $IC =$

$$\left((s, t) \left[\begin{array}{l} s \in hComponents.select(i|i.event = disable).outputPorts, \\ t \in hComponents.select(i|i.event = reenable).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event \in \{create, reenable\}).outputPorts, \\ t \in hComponents.select(i|i.event \in \{disable, manualStart\}).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event \in \{manualStart, resume, reactivate\}).outputPorts, \\ t \in hComponents.select(i|i.event \in \{suspend, terminate, fault\}).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event = suspend).outputPorts, \\ t \in hComponents.select(i|i.event = resume).inputPorts, \\ \vee \\ s \in hComponents.select(i|i.event = fault).outputPorts, \\ t \in hComponents.select(i|i.event = reactivate).inputPorts \end{array} \right] \right) \leftrightarrow$$

$Item.type = discretionaryItem \wedge Item.mar = true \wedge Item.ac = true$

• $IC =$

$$\left\{ (s, t) \left[\begin{array}{l} s \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} create, \\ resume, \\ reactivate \end{array} \right\} \right).outputPorts, \\ t \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} complete, \\ suspend, \\ terminate, \\ fault \end{array} \right\} \right).inputPorts, \\ \vee \\ s \in hComponents.select(i \mid i.event = suspend).outputPorts, \\ t \in hComponents.select(i \mid i.event = resume).inputPorts, \\ \vee \\ s \in hComponents.select(i \mid i.event = fault).outputPorts, \\ t \in hComponents.select(i \mid i.event = reactivate).inputPorts \end{array} \right. \right\} \leftrightarrow$$

Item.type = discretionaryItem \wedge *Item.mar = false* \wedge *Item.ac = false*

- *IC =*

$$\left\{ (s, t) \left[\begin{array}{l} s \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} create, \\ resume, \\ reactivate \end{array} \right\} \right).outputPorts, \\ t \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} suspend, \\ terminate, \\ fault \end{array} \right\} \right).inputPorts, \\ \vee \\ s \in hComponents.select(i \mid i.event = suspend).outputPorts, \\ t \in hComponents.select(i \mid i.event = resume).inputPorts, \\ \vee \\ s \in hComponents.select(i \mid i.event = fault).outputPorts, \\ t \in hComponents.select(i \mid i.event = reactivate).inputPorts \end{array} \right. \right\} \leftrightarrow$$

Item.type = discretionaryItem \wedge *Item.mar = false* \wedge *Item.ac = true*

4.3.3 Milestone/EventListener \rightarrow HSystem

Here we will explain how a Milestone/EventListener element translates to a coupled HSystem which contains a set of atomic HSystems, each of which generates the Milestone- or the EventListener-related events. Please note that

- *Milestone* refers to a *planItem* referring to a *Stage*
- *EventListener* refers to a *planItem* referring to a *EventListener*
 - *EventListener* can be either a *UserEventListener* or a *TimerEventListener*
- *Item* refers to the *Milestone* or the *EventListener* element
- *entry* refers to the *entrySentry* of the *Item*
- *trigger* refers to the element the *Item*'s *entrySentry* referring to
 - *trigger.event* can be either *complete* or *terminate*

4.3.3.1 Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Like the HSystem which is converted from the RootStage, each HSystem transformed from a Miestone/EventListener element has also two ports: one for receiving incoming events, and one for sending events to other HSystems.

4.3.3.2 HComponents

According to the type of the item, the event set is various, as given below:

$$\begin{aligned} HComponents &= \{H_{suspend}, H_{resume}, H_{terminate}, H_{occur}\} \leftrightarrow Item \\ &= UserEventListener \end{aligned}$$

$$HComponents = \{H_{suspend}, H_{resume}, H_{terminate}\} \leftrightarrow Item \neq UserEventListener$$

Ports

Each atomic event HSystem has two ports: one for receiving incoming events, and one for sending events to other HSystems.

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Configurations

Each atomic event HSystem has two essential configurations: *passive* and *active*. The *ta* of the *passive* configuration equals to positive infinity, and the *ta* of the *active* configuration will be assigned by case workers later.

$$Configurations = \{passive, active\}, \text{ such that}$$

$$ta_{passive} = +\infty$$

$$ta_{active} = \sigma$$

Transitions

Here we will detail all the possible configuration transitions of each Milestone-/EventListener-related event atomic HSystem, considering different values of item attributes we have introduced above.

Transitions = {*ICT*, *ECT*}, such that

$$ICT: c \xrightarrow{outputEvent} c', \text{ such that}$$

- ($c = active, c' = passive, outputEvent = (self, self.event)$)

$$ECT: c \xrightarrow{trigger} c', \text{ such that}$$

- ($c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self.trigger, complete), \\ (self, resume) \end{array} \right\}$) \leftrightarrow
 $Item = Milestone \wedge self.event \in \{suspend, terminate\} \wedge self.entry = true \wedge self.trigger.event = complete$
- ($c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self.trigger, terminate), \\ (self, resume) \end{array} \right\}$) \leftrightarrow
 $Item = Milestone \wedge self.event \in \{suspend, terminate\} \wedge self.entry = true \wedge self.trigger.event = terminate$
- ($c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self.parent, create), \\ (self.parent, manualStart), \\ (self.parent, resume), \\ (self.parent, reactivate), \\ (self, resume) \end{array} \right\}$) \leftrightarrow
 $(Item = Milestone \wedge self.event \in \{suspend, terminate\} \wedge self.entry = false) \vee (Item = UserEventListener \wedge self.event \in \{occur, suspend, terminate\}) \vee (Item = TimerEventListener \wedge self.event \in \{suspend, terminate\} \wedge Item.timerStart = null)$
- ($c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (Item.timerStart.item), \\ (Item.timerStart.event) \end{array} \right\}$) \leftrightarrow
 $Item = TimerEventListener \wedge self.event \in \{suspend, terminate\} \wedge Item.timerStart \neq null$

- $(c = \text{passive}, c' = \text{active}, \text{trigger} \in \{\{\text{self}, \text{suspend}\}\}) \leftrightarrow \text{self.event} \in \{\text{resume}\}$

4.3.3.3 Configurations

The configurations of the event generator HSystem converted from a Milestone/EventListener item define the couplings of the atomic event HSystems obtained.

PassiveConfigurations = $EOC \cup EIC \cup IC$, such that

- $EOC = \{(s, t) \mid s \in hComponents.outputPorts, t = \text{EventOut}\}$
- $EIC =$

$$\left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} \text{suspend}, \\ \text{terminate}, \\ \text{occur} \end{array} \right\} \right).inputPorts \end{array} \right. \right\} \leftrightarrow$$

Item = UserEventListener
- $EIC =$

$$\left\{ (s, t) \left| \begin{array}{l} s = \text{EventIn}, \\ t \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} \text{suspend}, \\ \text{terminate} \end{array} \right\} \right).inputPorts \end{array} \right. \right\} \leftrightarrow$$

Item \neq UserEventListener
- $IC =$

$$\left\{ (s, t) \left| \begin{array}{l} s \in hComponents.select(i \mid i.event = \text{suspend}).outputPorts, \\ t \in hComponents.select(i \mid i.event = \text{resume}).inputPorts, \\ \vee \\ s \in hComponents.select(i \mid i.event = \text{resume}).outputPorts, \\ t \in hComponents.select \left(i \mid i.event \in \left\{ \begin{array}{l} \text{disable}, \\ \text{suspend} \end{array} \right\} \right).inputPorts \end{array} \right. \right\}$$

4.3.4 CaseFileItem \rightarrow HSystem

Here we will explain how a CaseFileItem element translates to a coupled HSystem which contains a set of atomic HSystems, each of which generates the CaseFileItem- related events. Please note that

- *Item* refers to the *CaseFileItem* element

4.3.4.1 Ports

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Like the HSystem which is converted from the RootStage, each HSystem transformed from a CaseFileItem element has also two ports: one for receiving incoming events, and one for sending events to other HSystems.

4.3.4.2 HComponents

Unlike other CMMN basic modeling items, the CaseFileItem has no specific attributes. Therefore, all the CaseFileItems will contain the same amount of atomic event HSystems, each of which represents a CaseFileItem-related event:

$$HComponents = \left\{ \begin{array}{l} H_{create}, H_{update}, H_{replace}, H_{addChild}, H_{removeChild}, \\ H_{addReference}, H_{removeReference}, H_{delete} \end{array} \right\}$$

Ports

Each atomic event HSystem has two ports: one for receiving incoming events, and one for sending events to other HSystems.

$$InputPorts = \{EventIn\}$$

$$OutputPorts = \{EventOut\}$$

Configurations

Each atomic event HSystem has two essential configurations: *passive* and *active*. The *ta* of the *passive* configuration equals to positive infinity, and the *ta* of the *active* configuration will be assigned by case workers later.

$$Configurations = \{passive, active\}, \text{ such that}$$

$$ta_{passive} = +\infty$$

$$ta_{active} = \sigma$$

Transitions

Here we will specify all the possible configuration transitions of each CaseFileItem-related event atomic HSystem.

$Transitions = \{ICT, ECT\}$, such that

$ICT: c \xrightarrow{outputEvent} c'$, such that

- $(c = active, c' = passive, outputEvent = (self, self.event)) \leftrightarrow self.event \neq create$
- $(c = active, c' = active, outputEvent = (self, create)) \leftrightarrow self.event = create$

$ECT: c \xrightarrow{trigger} c'$, such that

- $(c = passive, c' = active, trigger \in \{(Case, create), (Case, reactivate)\}) \leftrightarrow self.event \in \{create\}$
- $(c = passive, c' = active, trigger \in \left\{ \begin{array}{l} (self, create), \\ (self, update), \\ (self, replace), \\ (self, addChild), \\ (self, removeChild), \\ (self, addReference), \\ (self, removeReference) \end{array} \right\}) \leftrightarrow self.event \in \{update, replace, addChild, removeChild, addReference, removeReference, delete\}$

4.3.4.3 Configurations

The configurations of the event generator HSystem converted from a CaseFileItem item define the couplings of the atomic event HSystems obtained.

$PassiveConfigurations = EOC \cup EIC \cup IC$, such that

- $EOC = \{(s, t) \mid s \in hComponents.outputPorts, t = EventOut\}$
- $EIC =$

$$\{(s, t) \mid t \in hComponents.select(i \mid i.event \in \{create\}).inputPorts\} \leftrightarrow$$

$Item = UserEventListener$

- $IC =$

$$\left(\begin{array}{l} s \in hComponents.select(i \mid i.event = create).outputPorts, \\ t \in hComponents.select \left(i \left\{ \begin{array}{l} i.event \in \\ update, \\ replace, \\ addChild, \\ removeChild, \\ addReference, \\ removeReference, \\ delete \end{array} \right\} \right).inputPorts, \\ \vee \\ s \in hComponents.select \left(i \left\{ \begin{array}{l} i.event \in \\ update, \\ replace, \\ addChild, \\ removeChild, \\ addReference, \\ removeReference, \\ create \end{array} \right\} \right).outputPorts, \\ t \in hComponents.select(i \mid i.event = delete).inputPorts \end{array} \right)$$

4.3.5 Complementary information

Until now from the original case model we could receive an *IEF* which contains all possible item events and the couplings among them. However, in reality it is not necessary to have all the events for conducting a simulation practice. To this end, case workers have the right to select a set of events which will be used in the simulation practice later. Essentially, the information to complete is as following:

- σ (specified by case workers)
- **triggers** (selected from available generated options)
- **internal couplings** (selected from available generated options)
- **event scheduling** (automatically generated on the basis of the information specified above)

Once all required information is specified, then the final *EF* can be generated.

- Each h (where $h \in hSystems$) has an index number associated that making it unique.
- For each h where $h \in hSystems$ AND $h.hComponents = null$, if $h.ICT.outputEvent \neq (Case, create) \wedge h.ECT.trigger = null$, then this h

will be removed, as well as the *couplings* relating to it.

- For each h where $h \in hSystems$ AND $h.containedHSystem \neq null$, if $h.hComponents = \emptyset$, then this h will be removed, as well as the *couplings* relating to it. For each pair of *couplings* (removed) that coming from h' and going into h'' , a new *coupling* will be generated (which is used to establish a connection between h' and h''), where $h'.outputPort = h''.inputPort$

4.3.6 Example

We will use the CMMN example we introduced in section 3.3 to explain how the EventGenerator of this case model is created. From the HiLLS case model that is generated from the CMMN example, a set of HiLLS events will be generated. It includes events such as $(CFM, terminate, triggers, duration)$, $(IR, complete, triggers, duration)$, $(IR, terminate, triggers, duration)$, and so on, which are generated through a union of the Cartesian product results, as given in the following:

$$eventSet = \{CFM\} \times SetA \cup \{R\} \times SetB \cup \left\{ \begin{matrix} IR, CR, CCN, \\ CC, RMI \end{matrix} \right\} \times SetC \cup \{RI, BIA, C, D\} \times SetD :$$

- $SetA = \{create, complete, terminate, fault, suspend, reactivate, close\}$
- $SetB = \left\{ \begin{matrix} create, disable, reenable, manualStart, complete, \\ terminate, fault, suspend, reactivate, resume \end{matrix} \right\}$
- $SetC = \{create, occur, terminate, suspend, resume\}$
- $SetD = \left\{ \begin{matrix} create, update, replace, addChild, removeChild, \\ addReference, removeReference \end{matrix} \right\}$

Based on their research purpose, case workers will then select a subset of events from the eventSet, and specify the missing values as given in Table 4-7. The value in the column of **Triggers** is the index number representing an event. The value in the column **Duration** is a uniform distributed random number, in which the two numbers in the bracket are the lower and upper bounds from left to right, respectively.

Following the rules defined above, each selected event will convert to a HSystem. Generally, there are three types of configuration transitions within different HSystems, as given in Figure 4-14, Figure 4-15, and Figure 4-16, respectively, where the configuration with a bold outline is the one the system stays at initialization time. Figure 4-14 shows the configuration transition of the HSystem converting from the event $(CFM, create, -, -)$ to create the whole case, since this type of event occurs immediately once simulation starts, with no triggers required. The system stays in the transient configuration Active for zero time units, and then sends out the event to create the CFM case and transits to the passive configuration.

Table 4-7: The complete information of selected events

<i>Index</i>	<i>Item</i>	<i>Action</i>	<i>Triggers</i>	<i>Duration</i>
1	CFM	create	–	–
2	CFM	suspend	1	[100, 180]
3	CFM	reactivate	2	[50, 70]
4	R	create	1	[5, 20]
5	IR	complete	4	[10, 15]
6	CCN	complete	5	[17, 25]
7	CC	complete	6	[10, 18]

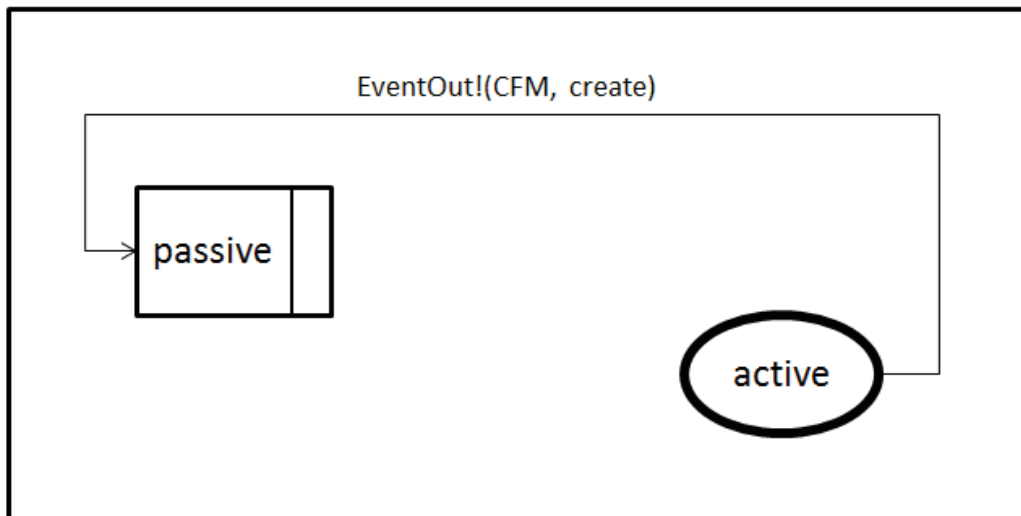


Figure 4-14: The configuration transition of event (*CFM, create, -, -*)

If triggers are required, the system will stay in the passive configuration when simulation initiates. Once the trigger (event (*CFM, create*)) is received, as given in Figure 4-15, an external configuration transition will be observed: the system will transit to the active configuration and stay until the life-cycle is expired. Then, it sends out the event to the case model and transits back to the passive one.

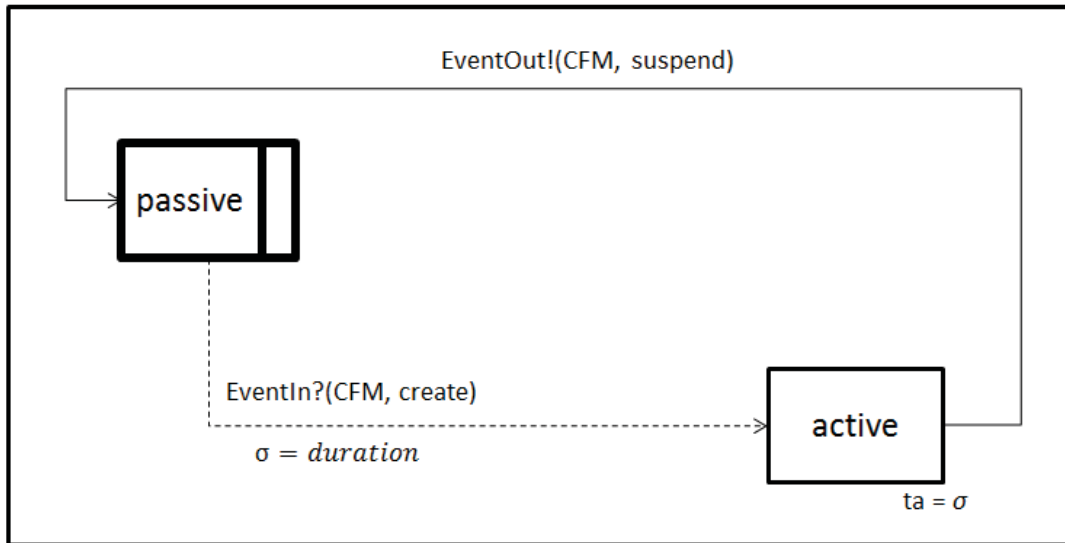


Figure 4-15: The configuration transition of event $(CFM, suspend, 1, [100, 180])$

Figure 4-16 illustrates the situation in which the item of the event refers to a Task or a Milestone with its repetition rule set to True.

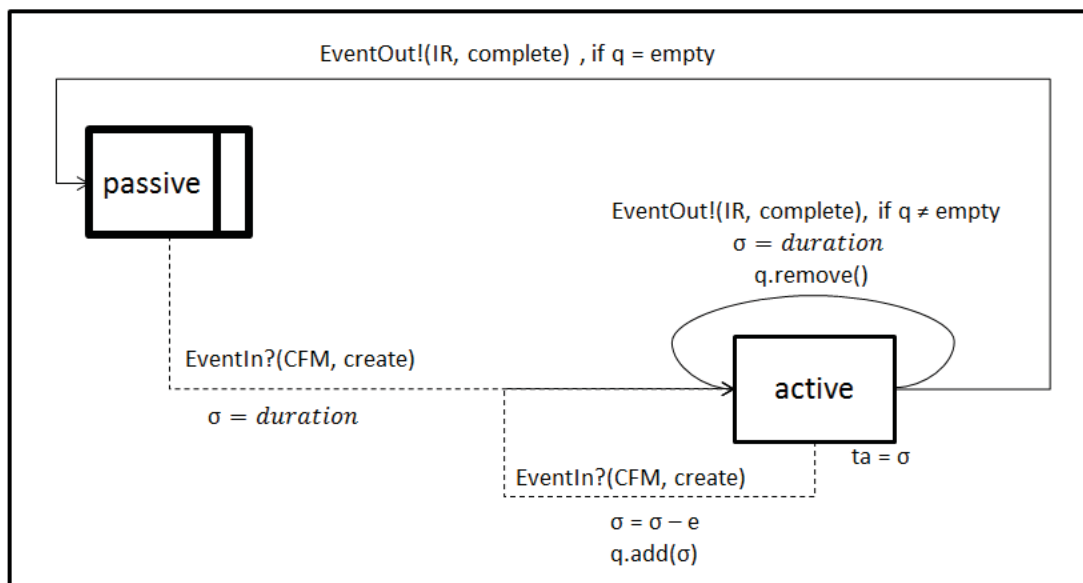


Figure 4-16: The configuration transition of event $(IR, complete, 4, [10, 15])$

All such HSystems are coupled together, as shown in Figure 4-17 and Figure 4-18. The former specifies the external output couplings (EOC) connecting components output ports to the EventGenerator's output port, and the latter defines the internal couplings (IC) connecting components output ports to their input ports.

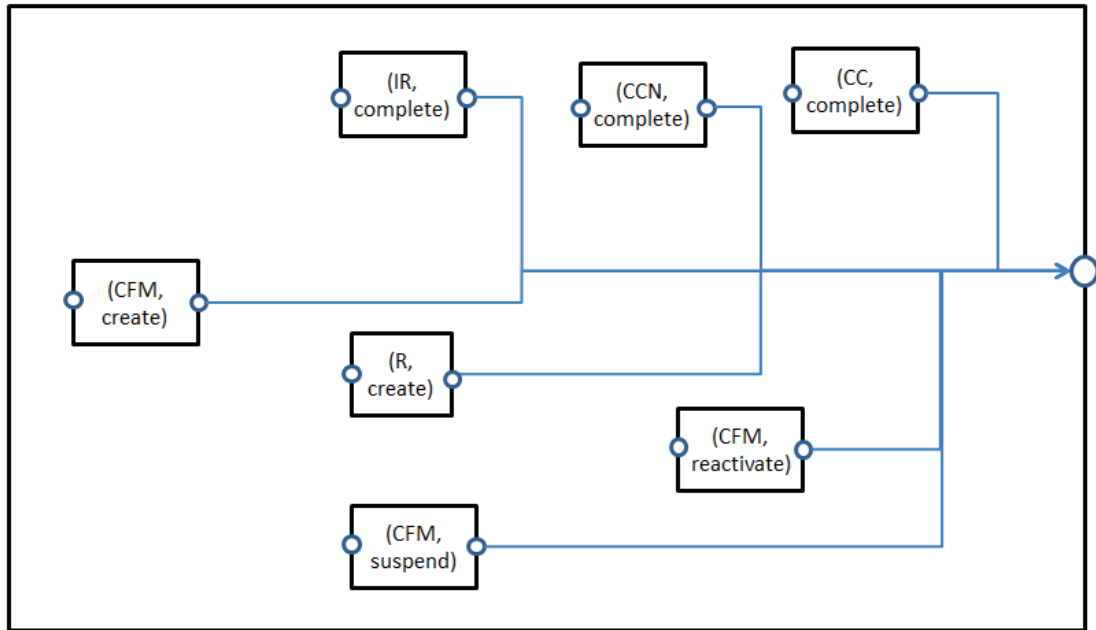


Figure 4-17: The EOC specified in EventGenerator

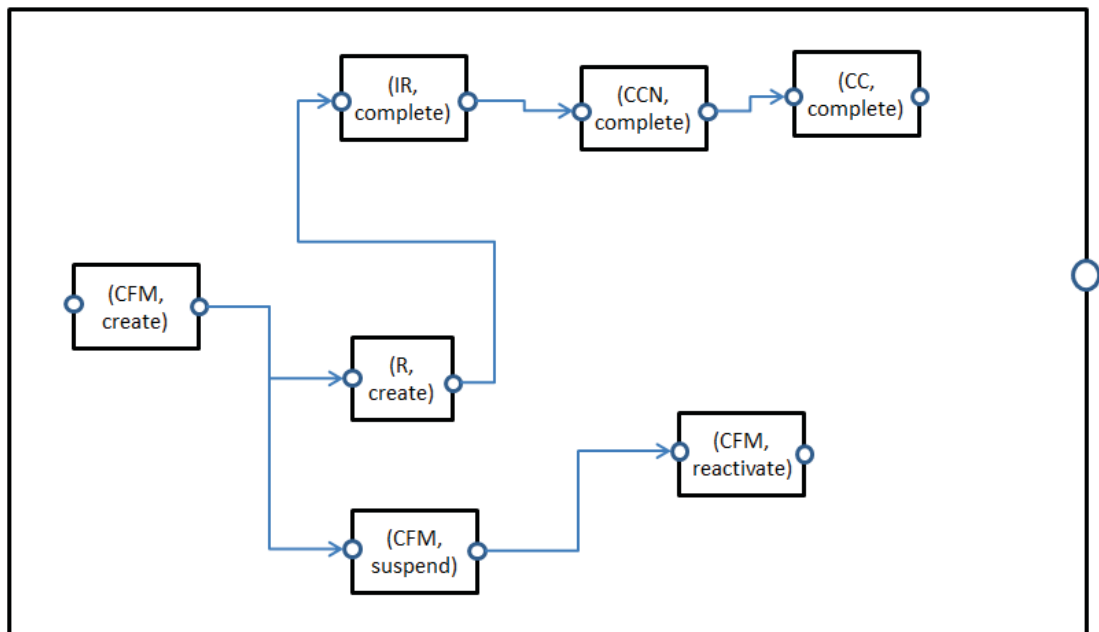


Figure 4-18: The IC specified in EventGenerator

However, we have noticed that with our approach there is a potential limitation relating to the total number of events: if there are many modeling elements in a case model, the total number of events will rise in an exponential growth manner.

Moreover, we establish an Analyzer which is used to collect the output events and analyze the performance of the system of interest. The Analyzer is also a HiLLS HSystem. It has only one input port for receiving item status that is sending out from

the case model, computes the value of performance metrics, and provides the results to case workers when simulation ends. It has only one configuration (passive): each time a trigger comes, it transits back to itself, during which the values of performance metrics are calculated. The specification of the Analyzer is given in the following, and gives its graphical representation. In the next section, we will further introduce what are these performance metrics, and how we link them to case models so that they can analyze and reveal the operational performance of cases under study, from different perspectives.

$$\text{Analyzer} = \langle \text{inputPorts}, \text{configurations}, \text{transitions} \rangle :$$

- $\text{InputPorts} = \{\text{EventIn}\}$
- $\text{configurations} = \{\text{passive}\}$
- $\text{transitions} = c \xrightarrow{\text{trigger}} c' :$
 - $(c = \text{passive}, c' = \text{passive})$

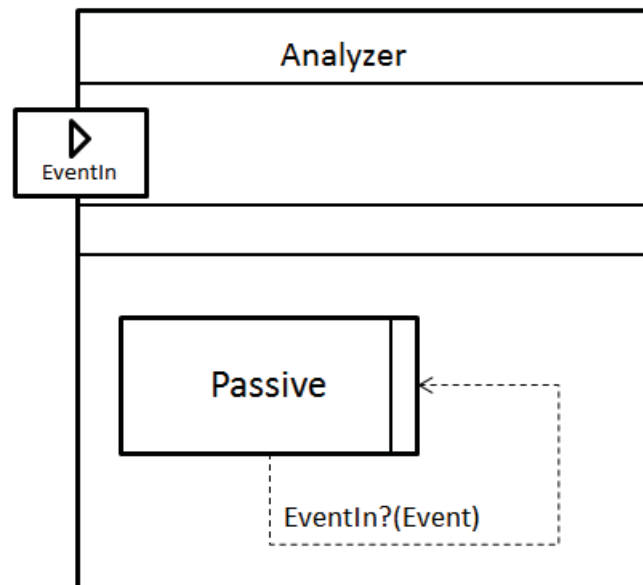


Figure 4-19: The graphical representation of Analyzer

4.4 Performance Metrics

From the literature review in chapter 3 regarding process improvement, we select several performance metrics to help case workers monitor, analyze and evaluate the system's performance from different perspectives (lean and TOC), at different

management levels (operational, tactical, and strategic). Such performance indicators will show a deeper view in both as-is and to-be cases. For example, waste and bottlenecks can be identified hidden in as-is models. Through re-designing as-is models and simulating to-be models, case workers can check if spotted waste and bottlenecks have been removed or not, and investigate if there is any new issue. In addition, an expected target of system performance should be established in advance, so that case workers can compare the simulation results with their target to see where the difference is. We will explain in detail how we link these metrics to case models in this section.

4.4.1 Lean-related Performance Metrics

Lean aims at identifying and eventually eliminating all kinds of waste within processes. The following indicators are defined to evaluate the system performance and reveal the hidden waste within the system, at the operational level. The goal is to anticipate if any waste exists, and assess the performance of the system to see if it is at an expectable level or not.

Cycle Time

The *cycle time* refers to the time elapsed from the beginning of a work process until it is completed [Tapping 2007]. We use *cycle time* to represent how long time a Case or a single Task takes to complete from initialization. Case workers can observe the trend of the cycle time obtained from several simulation experiments, in order to determine if it has been reduced or increased through process changes. In CM, the *cycle time* of a Case or a Task can be calculated by the time stamp when the item is in the completed status, minus the time stamp when the item is in the active status. This indicator will be used to help predict how many employees are required for certain tasks.

$$\text{Cycle Time} = \text{Time}(\text{completed}) - \text{Time}(\text{active})$$

Suspending Time

The *suspending time* refers to the amount of time the work is being suspended. Basically, we use it to indicate for how long time a Case or a single Task suspends, which is viewed as waste. The *suspending time* of a Case or a Task can be obtained by the time stamp when the item transits to the active status from suspended, minus the time stamp when the item becomes to be suspended.

$$\text{Suspending Time} = \text{Time}(\text{active}) - \text{Time}(\text{suspended})$$

Downtime

We use *downtime* to represent the amount of time machines are not being used or case workers are not working, due to technical problems. By observing the trend of downtime, we can see if the technical problems become more serious or less, or being solved totally. The value of the *downtime* of a Case or a Task can be calculated by the time stamp when the item transits to the active status from failed, minus the time stamp when the item becomes to be failed.

$$Downtime = Time(active) - Time(failed)$$

Idle Time

The *idle time* refers to the amount of time case workers spend in waiting work items or information coming from a previous step in the process. The *idle time* is a type of waste from the lean point of view. Moreover, a long *idle time* indicates that the previous step is a bottleneck candidate in the process [Goldratt et al. 1992]. The *idle time* will be observed only in the situation that a Task is repeatable, i.e., the *repetitionRule* of that Task is set to True. In this situation, if case workers just complete the n^{th} instance of that Task, and waiting for receiving information so that they can continue to work on the $n+1^{\text{th}}$ instance, then the *idle time* will be observed. The value of the *idle time* of a repeated Task can be calculated by the time stamp when the $n+1^{\text{th}}$ instance transits to a non-null status from null, minus the time stamp when the n^{th} instance becomes completed.

$$Idle\ Time = Time_{n+1}(non_null) - Time_n(completed), n \geq 1$$

Work-In-Progress (WIP)

WIP is used to represent the amount of Task instances that have been initialized but not yet completed. Knowing the *WIPs* helps organizations improve the flow of value through the whole system, since Tasks cannot add any value to customers unless they are completed. The *WIPs* of Tasks within a Case can be counted by the summation of Task instances that their status are active.

$$WIPs = \sum_{i=1}^n t_i, \text{ where}$$

$$\forall t \in \{t_1, t_2, \dots, t_n\} : Status(t) = active$$

Backlog

The *backlog* refers to the amount of Task instances staying in active and waiting for case workers to complete. The same as *idle time*, the *backlog* will be observed in the situation that a Task is repeatable. In this situation, if case workers just complete

the n^{th} instance of a Task, and waiting for receiving information so that they can continue to work on the $n+1^{\text{th}}$ instance, then the $n+1^{\text{th}}$ instance of that Task is a backlog. By identifying the *backlog* in the system, managers are able to see where they might get stuck within their processes. Moreover, a smaller *backlog* leads to a shorter *idle time* and a shorter total *cycle time*, and a huge *backlog* indicates that the succeeding step in the process is a bottleneck candidate. The *backlog* of repeated Tasks within a Case can be counted by the summation of Task instances that their status are active, and case workers assigning to this Task are waiting for additional information, not working on completing these Task instances.

$$\text{backlog} = \sum_{i=1}^n t_i, \text{ where}$$

$$\forall t \in \{t_1, t_2, \dots, t_n\} : \text{Status}(t) = \text{active} \wedge \text{Status}(\text{case workers}) = \text{idle}$$

Employees Needed

This performance indicator specifies how many case workers are required to complete a Case or a Task instance. If unnecessary case workers are assigned to a Case or a Task, then this is a kind of waste. By comparing the simulation results with different amount of needed employees we can identify that how many case workers assigning to different Tasks will lead to the best performance. Moreover, according to different types of payment of case workers, such cost will also be considered as *fixed cost* (if case workers are paid by fixed salaries, regardless of how many hours they work actually), or *variable cost* (if case workers are paid according to their working hours), which are explained in the *Total Cost* indicator.

Total Cost

The indicator of *total cost* refers to all the costs incurred in producing produces or providing services. Generally, two main types of cost will be observed in business: *fixed cost*, and *variable cost*, where $\text{total cost} = \text{fixed cost} + \text{variable cost}$. The former is the type of cost that remains fixed, irrespective of changes on the level of products produced or services provided. Typical examples of *fixed cost* include salaries, insurance, rent, and so on. The latter one, *variable cost*, refers to the type of cost that changes in proportion to the level of products produced or services provided. For example, the cost of raw materials, billable staff wages (where employees are paid according to their working hours), production supplies such as machinery oil, etc., are all considered as *variable costs* in business domain.

4.4.2 TOC-related Performance Metrics

TOC is a system management principle aiming at improving system performance by identifying and eliminating bottlenecks. To this end, we use two performance indicators (which are *idle time* and *backlog*) to help case workers determine bottlenecks. As we have explained above, a long *idle time* indicates that the previous step is a bottleneck, and a huge *backlog* implies that the succeeding step in the process is a bottleneck candidate. In addition, in order to monitor, manage and analyze system performance from the TOC perspective, we propose to use the *Throughput Accounting* (TA) technique in managing processes in case management, at the tactic level. TA consists of three main concepts, as we have reviewed in section 3: *throughput*, *investment*, and *operating expenses* (OE). The main idea is to increase *throughput*, while reducing *investment* and *OE* simultaneously, in order to balance all components within process to achieve the whole system's optimum.

Throughput

Throughput is the rate at which the entire system generates money through sales. It often refers to the added value through sales, as the formula given in below, where sales refers to the money organizations made through selling products or providing services, and *variable cost* refers to the type of cost that changes in proportion to the level of products produced or services provided. For example, the cost of raw materials, billable staff wages (where employees are paid according to their working hours), production supplies such as machinery oil, etc., are all considered as *variable costs* in business domain.

$$\text{throughput} = \text{sales} - \text{total variable costs}$$

Investment

Investment often refers to the money tied up in the system. Basically, it includes the investment the organization has made (tools, capital equipment, furnishings, etc.), and the physical inventory (e.g., WIPs, finished products). In essence, in order to decrease the level of *investment* organizations should focus on reducing their inventory level.

Operating Expenses

OE refers to the money organizations spend in turning *investment* into *throughput*. Expenditures such as salaries of employees, bills of supplies are all viewed as *OE* of organizations. In order to make profit, *OE* should be reduced as well.

In addition, since some performance indicators at the strategic level are linking

with throughput, investment and OE, once a certain number of cases have been done managers are able to have a global view on how their business is going on, at a relative higher level. Such indicators include net profit (which indicates the actual profit organizations have made), return on investment (which evaluates the efficiency of investments), and productivity (which measures the efficiency of employees).

$$\text{net profit} = \text{throughput} - \text{operating expenses}$$

$$\text{return on investment} = \frac{\text{throughput} - \text{operating expenses}}{\text{investment}}$$

$$\text{productivity} = \frac{\text{throughput}}{\text{operating expenses}}$$

4.5 Conclusion

In this chapter, we have explained the CMI module we have proposed to help case workers better manage their cases and improve their efficiency and effectiveness. Case workers start with case models, which later will be transformed into HiLLS models in order to allow case workers conduct case model simulations. As we have explained in the beginning of this chapter before, the translations between a CMMN model and its targeted HiLLS model are not simply n-n mappings between their meta-models. Especially for the configurations and the transitions between them, the rules must take into account several conditions, e.g., each element's state, sentry state, etc. To this reason, we have adopted several manners to formally specify the CMMN to HiLLS model transformation rules. Regarding to the system structure, we have used tables in which the mappings between CMMN and HiLLS metamodels were defined. In terms of system behavior, which concerns configurations and their transitions, we have used mathematic formulas and tables together to specify how *Configurations* are generated, as well as how they transit from one to another. The rules of configuration transitions serve as the pre-conditions when executing HiLLS model simulations.

In addition, we have also proposed a semi-automatic mechanism to generate an EF for a case model, which concerns generating event models, event generator models, as well as an analyzer model. The translated case model, which conforms to HiLLS, will receive events generated from the EF, and will send outputs to the analyzer. This analyzer which will later compute the values of performance metrics we have defined from different point of view: Lean and TOC.

Case workers now start improving their daily work by analyzing a case model. If this model does not exist, it is necessary for them to create one, on the basis of stored

information. To ease this model creation phase, we will propose another module in the next chapter, CMD, which aims at constructing case models directly from recorded information automatically, instead of manually.

5. CASE MODEL DISCOVERY

5.1 Introduction

The aim of CMD module is to help case workers automatically construct case models from the historical raw data recorded. To this end, we consider to adopt the process discovery approach that extracts process information from event logs. Different process discovering algorithms result in process models conforming to different modeling formalisms: Process Tree, Hidden Markov Models, Yet Another Workflow Language, Event-Driven Process Chains, etc. Apparently, a gap exists between these process discovery modeling specifications and the de-facto modeling standard for CM (i.e., CMMN). To overcome this problem, we envision transforming Process Tree models into CMMN models, as illustrated in Figure 5-1. The reason for us to select Process Tree as the modeling formalism for expressing discovered models among other ones is that Process Tree models are ensured to represent sound models [Eck et al. 2014]. Moreover, the ETM algorithm used to generate Process Tree models guarantees that the discovered models are error-free, and meet the four quality-evaluation dimensions: replay fitness, simplicity, generalization, and precision [Buijs 2014]. In this section, we will first explain how we translate a Process Tree model into a CMMN correspondence. Then we will use a simple example to illustrate our case models discovering approach.

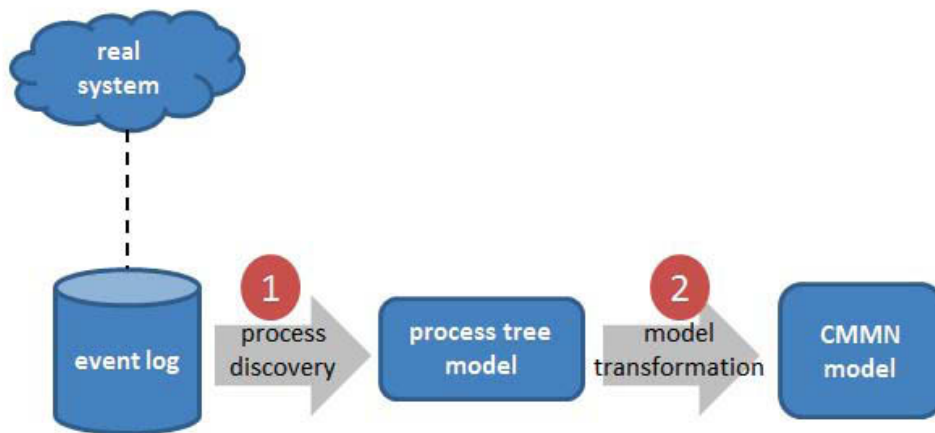


Figure 5-1: The CMD module

5.2 Translations from Process Tree to CMMN

A generic way to transform Process Tree models into CMMN counterparts is to map the Process Tree metamodel onto the CMMN one, where the former is given

before in chapter 3, and the latter is given in [OMG 2014]. Process Tree specifies three types of nodes: Task, Event, and Operator. Each node will convert to a corresponding CMMN element, as illustrated in tables below.


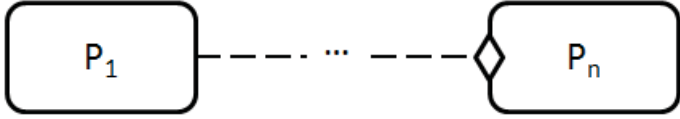

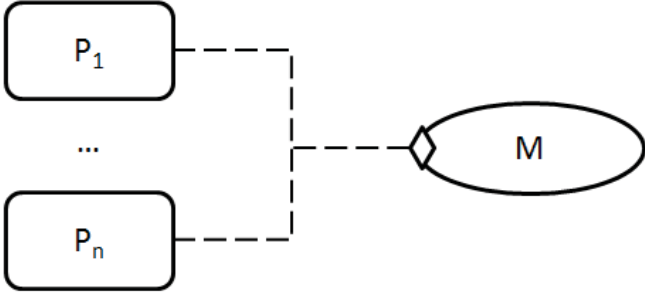
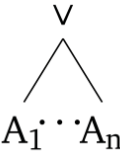
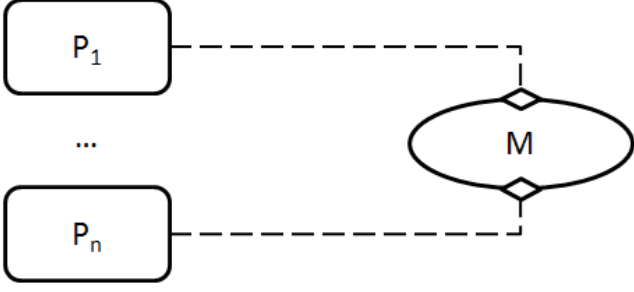

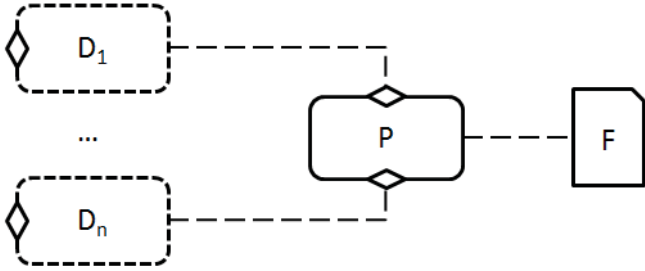
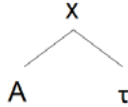
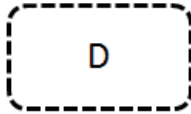
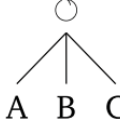
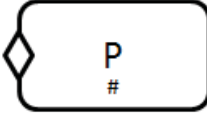
Table 5-1: The mappings between Process Tree and CMMN - 1

Process Tree	CMMN
<i>Manual Task</i>	<i>Task (isBlocking=True)</i>
<i>Automatic Task</i>	<i>Task (isBlocking=False)</i>
<i>Time Out Event</i>	<i>TimerEventListener</i>
<i>Message Event</i>	<i>CaseFileItem</i>
<i>Edge</i>	<i>Sentry</i>

Basically, each Process Tree Task element will convert to a CMMN PlanItem or a DiscretionaryItem referring to a Task (where a CMMN Task element represents an atomic unit of work). The *isBlocking* attribute of obtained CMMN Task elements is set to True (if the Process Tree element is a Manual Task), or False (if the Process Tree element is an Automatic Task). Based on the logical relationships between Tasks, the targeted CMMN elements will be either PlanItems or DiscretionaryItems, as specified in Table 5-2 below. Moreover, the equivalent of a Time Out Event of Process Tree is a PlanItem referring to a TimerEventListener element (where CMMN TimerEventListener elements is adopted to catch predefined elapses of time), and the counterpart of a Message Event of Process Tree is a CaseFileItem (which represents a piece of information necessary for proceeding a case). In addition, each Process Tree Edge element will be translated into a CMMN Sentry element. According to different type of nodes and/or operators an Edge connects, the detailed information of the generated sentries will be different as well, as specified in Table 5-2.

Table 5-2 illustrates the translations from Operators of Process Tree to CMMN elements. As specified above, there are basically five Operators: SEQ, AND, OR, XOR, and LOOP.

Table 5-2: The mappings between Process Tree and CMMN - 2

Process Tree	CMMN
	
	
	
	
	
	

1) SEQ Operator, $\rightarrow \langle A_1, A_2, \dots, A_n \rangle$

If nodes $A_1, A_2 \dots A_n$ are linked by an SEQ operator ($\rightarrow \langle A_1, A_2, \dots, A_n \rangle$), then each node A_i ($1 \leq i \leq n$) converts to a CMMN PlanItem P_i , respectively. P_i refers to different types of CMMN elements, based on the type of the Process Tree node it converts from, as given in Table 5-1: (1) P_i refers to a Task with its `isBlocking` attribute set to True, if A_i is a Manual Task; (2) P_i refers to a Task with its `isBlocking` attribute set to False, if A_i is an Automatic Task; (3) P_i refers to a `TimerEventListener`, if A_i is a Time Out Event; or (4) P_i refers to a `CaseFileItem`, if A_i is a Message Event. Moreover, each P_i (except P_1) has a sentry defined as its guarding condition, and these PlanItems are connected as a chain following the sequence specified in the original Process Tree model: P_2 links with P_1 through P_2 's entry sentry, P_3 links with P_2 through P_3 's entry sentry, and so on, as specified below. This way, once P_1 is completed then P_2 will be triggered to execute, and P_3 will be active once P_2 is completed, and so forth.

$$\forall p_i \in \{P_1, P_2, P_3, \dots, P_n\} :$$

- $p_i.entrySentry = null \leftrightarrow i = 1$
- $(p_i.entrySentry.size = 1) \wedge (p_i.entrySentry.onParts.sourceRef = p_{i-1}) \leftrightarrow i \neq 1$

2) AND Operator, $\wedge \langle A_1, A_2, \dots, A_n \rangle$

If nodes $A_1, A_2 \dots A_n$ are linked using an AND operator ($\wedge \langle A_1, A_2, \dots, A_n \rangle$), then each node A_i ($1 \leq i \leq n$) converts to a CMMN PlanItem P_i , respectively. As defined in the SEQ operator, the type of CMMN element each P_i referring to is different, according to the element type of A_i . The element M is a Milestone used as a constraint to indicate the AND logical relations among all the PlanItems: it will be completed only if each P_i is completed. To this end, M is associated with one and only one entry sentry S , and all PlanItems are connecting with S .

$$\forall p \in \{P_1, P_2, P_3, \dots, P_n\} :$$

- $m.entrySentry.size = 1$
- $p \in m.entrySentry.onParts.sourceRef$

3) OR Operator, $\vee \langle A_1, A_2, \dots, A_n \rangle$

If nodes $A_1, A_2 \dots A_n$ are linked using an OR operator ($\vee \langle A_1, A_2, \dots, A_n \rangle$), then each node A_i ($1 \leq i \leq n$) converts to a CMMN PlanItem P_i , respectively. As defined in the SEQ operator, the type of CMMN element each P_i referring to is different, according to the element type of A_i . Moreover, a Milestone M is defined and is associated with a set of entry sentries $S_1, S_2 \dots S_n$, and each S_i ($1 \leq i \leq n$) links with P_i , respectively. This way, the OR logical relation can be enabled: M will be completed if at least one P_i is

completed.

$$\forall p_i \in \{P_1, P_2, P_3, \dots, P_n\}, \forall s_i \in \{S_1, S_2, S_3, \dots, S_n\}:$$

- $m.entrySentry.size = n$
- $p_i = s_i.onParts.sourceRef$

4) XOR Operator, $\times \langle A_1, A_2, \dots, A_n \rangle$

If nodes $A_1, A_2 \dots A_n$ are linked using an XOR operator ($\times \langle A_1, A_2, \dots, A_n \rangle$), then each node $A_i (1 \leq i \leq n)$ converts to a CMMN DiscretionaryItem D_i , respectively. The same as the situation specified in the SEQ operator, the type of CMMN element each D_i referring to is different, according to the element type of A_i . Moreover, each D_i is associated with an entry sentry S_i that has only an ifPart defined, and the CaseFileItem F generated specifies the content for evaluating the ifPart of each S_i . In addition, a PlanItem P referring to a Task is created to modify the content of F (i.e., F is the output of P), and it links with D_i through its associated entry sentries (S_i). At initial time, all the DiscretionaryItems are at the null status, and their associated entry sentries are set to True. Once a D_i is created, then P will be activated to modify the content of F such that all other sentries of the DiscretionaryItems will be evaluated to False, except the one that has been created. This ensures that one and only one DiscretionaryItem can be executed.

$$\forall d_i \in \{D_1, D_2, D_3, \dots, D_n\} : d_i.entrySentry.size = 1$$

$$p.entrySentry.size = n$$

$$p.outputs.bindingRef = f$$

$$\forall s_i \in \{D_1.entrySentry, D_2.entrySentry, \dots, D_n.entrySentry\} : s_i.contextRef = f$$

$$\forall s'_i \in p.entrySentry : s'_i.onParts.sourceRef = d_i$$

5) XOR Operator, $\times \langle A, \tau \rangle$

However, if a node A is linked with τ (which represents a silent action) using an XOR operator ($\times \langle A, \tau \rangle$), then it indicates that either node A is executed, or nothing will happen. In this case, the node A converts to a CMMN DiscretionaryItem D , and the type of CMMN element D referring to is different, according to the element type of A . D is at the null status when initialization, and is applicable at run time. This way, D can be either created to be executed or still remains at the null status.

6) LOOP Operator, $\circ \langle A, B, C \rangle$

If nodes A , B and C are linked using an LOOP operator ($\odot(A, B, C)$), then a PlanItem P referring to a Task will be created. In addition, P is repeatable: the repetitionRule of P is set to True (which is indicated using a # shape). An entry sentry associated as its guarding condition is created too, so that each time the entry sentry evaluates to True, a new instance of P is created. Otherwise, no new instance of P will be created.

$p.entrySentry.size = 1$

$p.itemControl.repetitionRule = True$

5.3 Model Transformation Algorithm

In addition, we also define an algorithm in order to execute Process Tree to CMMN model transformations effectively and efficiently, which traverse all the operator nodes starting from the lowest level ones to the highest level one, where in our case the root node is at the highest level. More specifically, assuming that the root node is at level 0 , and the lowest level the children operators are at is level n . We start from converting the operators at level n , with all its containing children, and from left to right, as specified in the section above. Each child node will convert to its CMMN equivalence, and they will be joined together in the logical manner the operator node defines (SEQ, AND, OR, XOR, and LOOP). Then we move to the upper level, level $n-1$, and perform the transformation using the CMMN segments we obtained at level n . Then we continue until we reach to level 0 where the root node stays. Figure 5-2 below summarizes this algorithm using pseudo codes.

```

1  for (o in operators at level n)
2      convert o with its children
3  for (o in operators at level n-1)
4      convert o with its children
5  ...
6  for (o in operators at level 1)
7      convert o with its children
8  convert the root node with its children

```

Figure 5-2: The algorithm of the Process Tree to CMMN model transformation

In this case, each node within the Process Tree model will be visited once. Consequently, this algorithm takes linear time to complete a model transformation, i.e., its time complexity is: $T(n) = O(n)$. This will ease the model transformation process if case workers deal with cases with a great amount of process-related recorded

information.

5.4 Conclusion

In this chapter, continued with chapter 4 we have explained the CMD module. This CMD module aims at analyzing event logs and constructing case models on the basis of recorded process information. This module should serve as a case model discovering tool for case workers: case workers start with feeding recorded process information into this module and will obtain a corresponding (as-is) case model; after that, thanks to the CMI module they could begin modifying the as-is case model and forecasting the performance of the to-be models and analyzing the real value of their model modifications.

Due to the fact that a gap exists between these process discovery modeling specifications and the de-facto modeling standard for CM (i.e., CMMN), we proposed in our CMD module the model transformation from Process Tree to CMMN. Essentially, the meta-model of Process Tree concern Nodes and Edges, where the former relate to different types of tasks and the latter refers to the logical relationships between Nodes. Consequently, we have defined the mappings from Nodes to CMMN PlanItems and from Edges to the connections between PlanItems, respectively. Moreover, in order to ensure that the model transformation could be executed effectively when dealing with a massive quantity of process information, we also defined an algorithm with mapped all the nodes and all the edges in a linear time complexity manner.

6. CASE STUDY

6.1 Introduction

In chapter 4 and chapter 5 we have explained our CMD and CMI module, respectively. Moreover, we have showed how the two modules can help case workers improve their work performance. In this chapter, we will use a case study to give a comprehensive understanding about the whole story, from analyzing recorded process information to a successful model improvement.

The case study adapted from [Russo et al. 2013] and used in this section is to show how our CM solution help case works manage their daily work in terms of discovering case models, validating model transformations, and analyzing and improving case models in a quantitative and scientific manner. The transformations are implemented by ATL (for model-to-model transformations) and Aceleo (for model-to-text model transformation), and the HiLLS simulator is a java-implemented version.

6.2 Background

A bank deals with a certain amount of Manual Credit Transfer (MCT) operation in a monthly manner. The MCT operation is triggered by a request specifying the detailed information required, and it has the highest priority among other tasks.

Generally, two units are working together to complete the operation: a Payment unit, and an Accounting unit. The former deals with operations regarding transferring the payment and recording the operation in the payment information system; while the latter deals with checking the funds availability, confirming the operation and updating the corresponding account information in their accounting information system. Once a request is received (R), the basic tasks performed by the two units are listed below:

Payment Unit:

1. register and check the payment orders received (A1);
2. data entry in the payment information system (A2);
3. final check and validation with the manager's signature, and pass the order to the Accounting unit (A3).

Accounting Unit:

1. register and check the payment orders received (B1);
2. check the order received, check funds availability, validate payment and update information in the accounting information system (B2).

At this moment, employees are dealing with MCT cases, and recording their activity information in their data warehouse. In order to stay competitive, the managers are seeking possible ways to manage and eventually improve the operational performance of the MCT process. They want to understand how their process proceeds (e.g., the sequence of activities), and then investigate what the result will be if they implement a change initiative: merging the two units together to accomplish the operation, and reallocating the human resource in accordance. The challenge for them is to know if the change is worthy or not. The change will only be worthy if the benefits after change is larger than the cost which will spend on the change process. Otherwise, there is no sense to change their as-is process model.

6.3 Case Model Discovery

The first task for the case manager is to understand how MCT cases proceed. They can obtain a MCT model using the CMD approach, which merely requires activity-related data recorded in the event log. A segment of information is given in Table 6-1. Note that the value in the column **Timestamp** refers to the timestamp an activity ends.

Table 6-1: The historical event data

Process Instance	Activity	Timestamp
1	R	9-3-2004:15.01
1	A1	9-3-2004:15.18
1	A2	9-3-2004:15.29
2	R	9-3-2004:15.30
1	A3	9-3-2004:15.34
1	B1	9-3-2004:15.35
2	A1	9-3-2004:15.37
3	R	9-3-2004:15.39
2	B1	9-3-2004:15.41
2	A2	9-3-2004:15.48
1	B2	9-3-2004:15.50
...

As given in Table 6-1, it is a segment of process data stored in an event log, where three kinds of basic process information are recorded: the process instance, the name of each activity, and the timestamp indicating the time point an activity ends. The process discovery tool we adopted here is ProM (version 6.6), a java implemented extensible (through plug-ins) framework that supports various process mining techniques [Verbeek et al. 2009]. Many ProM plug-ins have been implemented for different purposes. Thanks to the various algorithms provided with such plug-ins,

sound and robust process models can be established through studying the knowledge hidden in the process execution logs. In this example, we select to use the Inductive Miner process discovery plug-in with its algorithm, due to the fact that the Inductive Miner algorithm guarantees to generate sound (i.e., free of deadlocks) and fitting (i.e., all the traces recorded in given event logs can be replayed) process tree models [Leemans et al. 2013].

From Table 6-1 we can also see that this process have been executed for several times continually, and a set of activities have been executed as well. The resulting Process Tree model representing such an event log using the Inductive Miner algorithm is given in Figure 6-1, from where we can see that tasks A_1 , A_2 and A_3 should be executed in sequence ($\rightarrow \langle A_1, A_2, A_3 \rangle$), and task D (which is a verification task that is executed not often) will or will not be executed ($\times \langle \tau, D \rangle$), where the black node represents a silence action (τ). Moreover, task B_1 is connected with $\rightarrow \langle A_1, A_2, A_3 \rangle$ using the AND operator, $\wedge \langle \rightarrow \langle A_1, A_2, A_3 \rangle, B_1 \rangle$, where in the model the AND operator is graphically shown as a + symbol. This set of tasks will be executed with R, B_2 and $\times \langle \tau, D \rangle$, in a sequence (R, $\wedge \langle \rightarrow \langle A_1, A_2, A_3 \rangle, B_1 \rangle, B_2, \times \langle \tau, D \rangle$). As a consequence, the complete process tree model given in Figure 6-1 can be expressed as $\rightarrow \langle R, \wedge \langle \rightarrow \langle A_1, A_2, A_3 \rangle, B_1 \rangle, B_2, \times \langle \tau, D \rangle \rangle$.

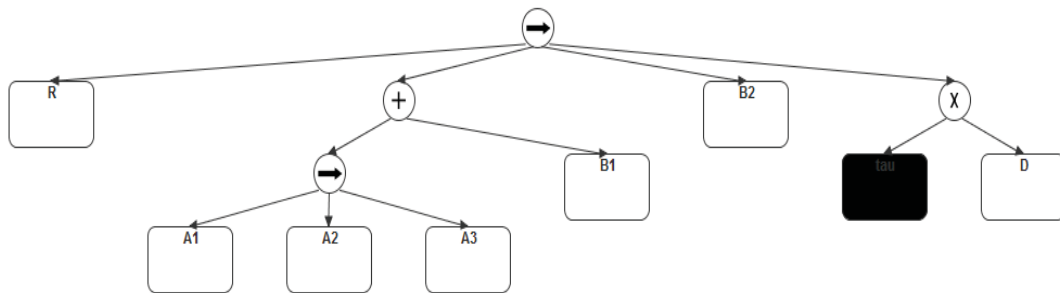


Figure 6-1: The resulting process tree model

To transform this process tree model into a CMMN case model, we start from the Operators at the lowest level, as the transformation algorithm specified above. In this case, it is the SEQ operator with tasks A_1 , A_2 and A_3 : $\rightarrow \langle A_1, A_2, A_3 \rangle$. From the transformation rules we know that each A_i (which belongs to $\{A_1, A_2, A_3\}$) will be converted into a PlanItem referring to a Task, and these Tasks are connected by sentries in a sequential way: A_2 connects with A_1 through an entry sentry, and A_3 links with A_2 through its entry sentry, as given in the upper part of Figure 6-2.

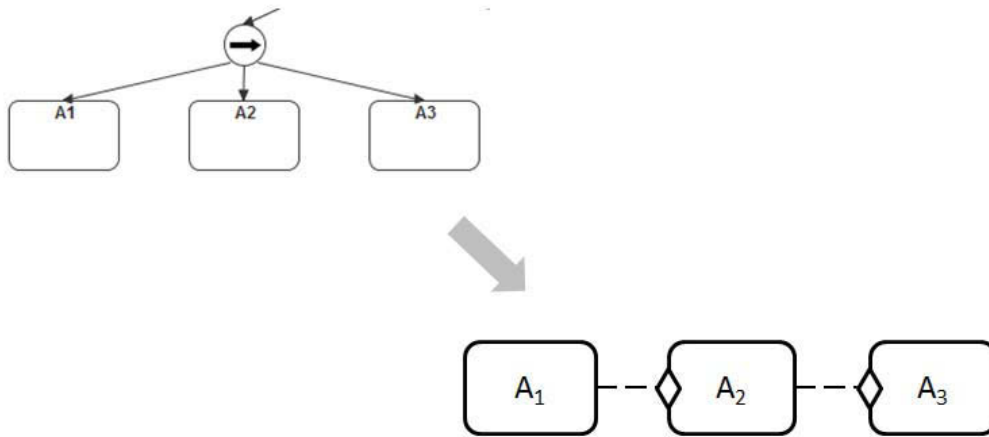


Figure 6-2: The model transformation example – 1

Then we move to the upper level in which we have two operators defined: an AND operator $\wedge(\rightarrow \langle A_1, A_2, A_3 \rangle, B_1)$, and a XOR operator $\times \langle \tau, D \rangle$. To convert the AND operator with all its children, B_1 will convert to a CMMN PlanItem referring to a Task, and this PlanItem will be linked together with the resulting CMMN model segments obtained from the last step transforming $\rightarrow \langle A_1, A_2, A_3 \rangle$, in a logical AND manner, as given in the upper part in Figure 6-3. Moreover, the XOR operator with its children ($\times \langle \tau, D \rangle$) will convert to a CMMN DiscretionaryItem referring to a Task, as given in the lower part in Figure 6-3.

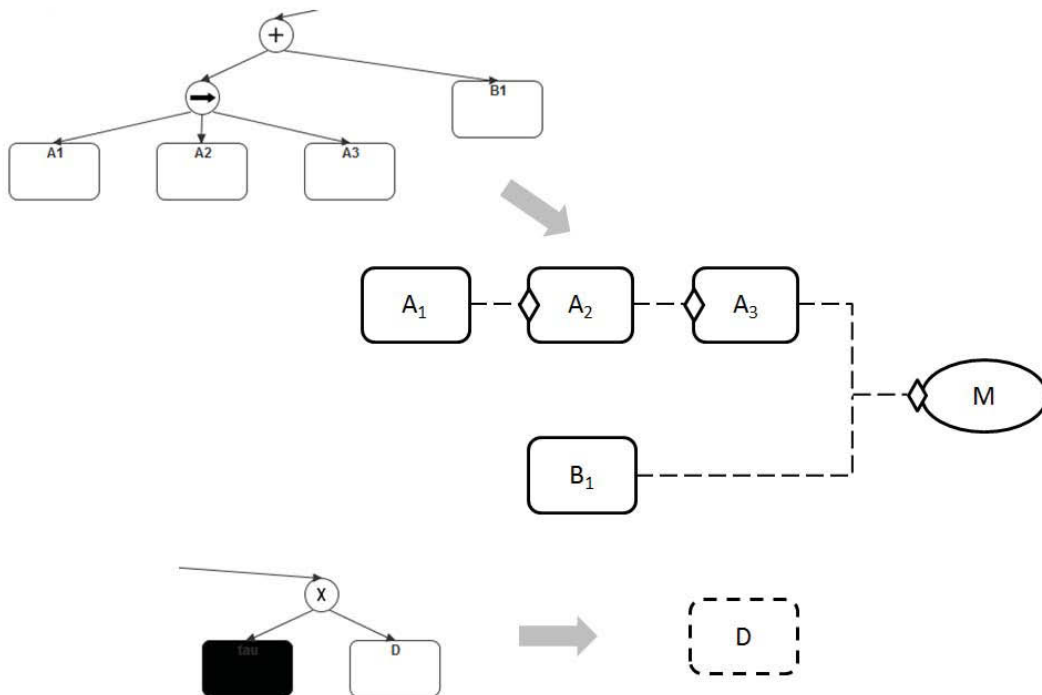


Figure 6-3: The model transformation example - 2

Now we will be at level 0 to transform the root node (which is an SEQ operator) with its children. The node R (as well as the node B_2) will convert to a CMMN PlanItem referring to a Task. In addition, all obtained CMMN model segments from last steps will be linked together in a sequential manner, through certain added sentries: A_1 and B_1 have an entry sentry connecting to R, B_2 has an entry sentry connecting to M, and D has an entry sentry connecting to B_2 , respectively. Figure 6-4 gives the final complete case model obtained from Process Tree to CMMN model transformation.

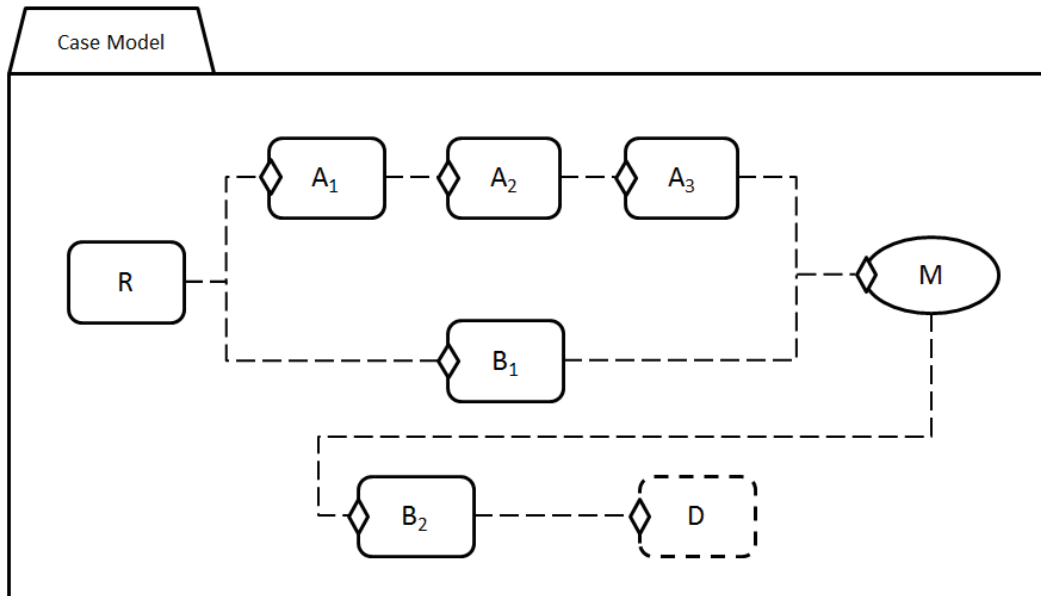


Figure 6-4: The as-as CMMN case model

At run time, this case model will be executed as follows: when the whole case model is initiated, then task R will be executed first. Once R is completed, then A1 and B1 will be triggered to be active, where A1, A2 and A3 will be executed in sequence. B2 will be activated only when M is completed, which indicates that A3 and B1 are both completed. Based on case workers decisions at run time, D will be executed if necessary once B2 is completed, or stay in the null status till the end. Consequently, we can see that all the tasks in both the original Process Tree model and the generated CMMN model will be executed in the same manner. Case workers are able to further analysis and improve their processes on the basis of case models specified using CMMN, a modeling formalism that is more friendly and easy-to-use for them. This is the as-is CMMN case model we start with. Next we will see how to forecast potential performance and predict hidden risks coming with the change initiative.

6.4 Generated HiLLS Model and Experimental Frame

According to the CMMN2HiLLS rules defined in chapter 4, the as-is MCT model's HiLLS counterpart is shown in Figure 6-5 in a tree-view HiLLS model editor. A detailed explanation of the obtained case model is illustrated in [Wang & Traoré 2014], as given in Figure 6-6 and Figure 6-7, where the former presents the system structure and the latter presents the system behavior. As we can see, the whole Case transforms to a HiLLS HSystem, and each of the basic modeling elements within that Case transforms to a HClass.

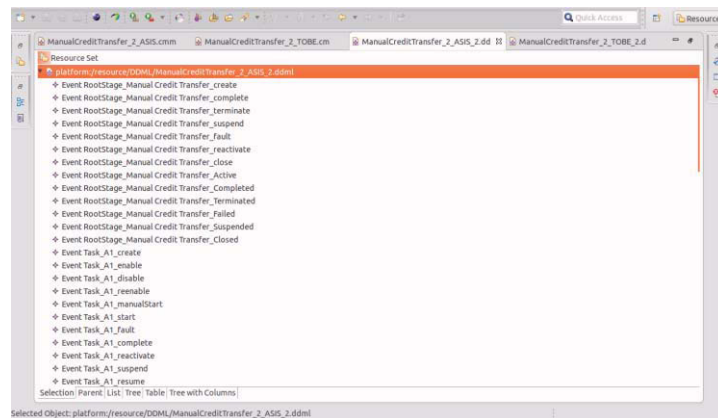


Figure 6-5: The as-is HiLLS model in a tree-view format

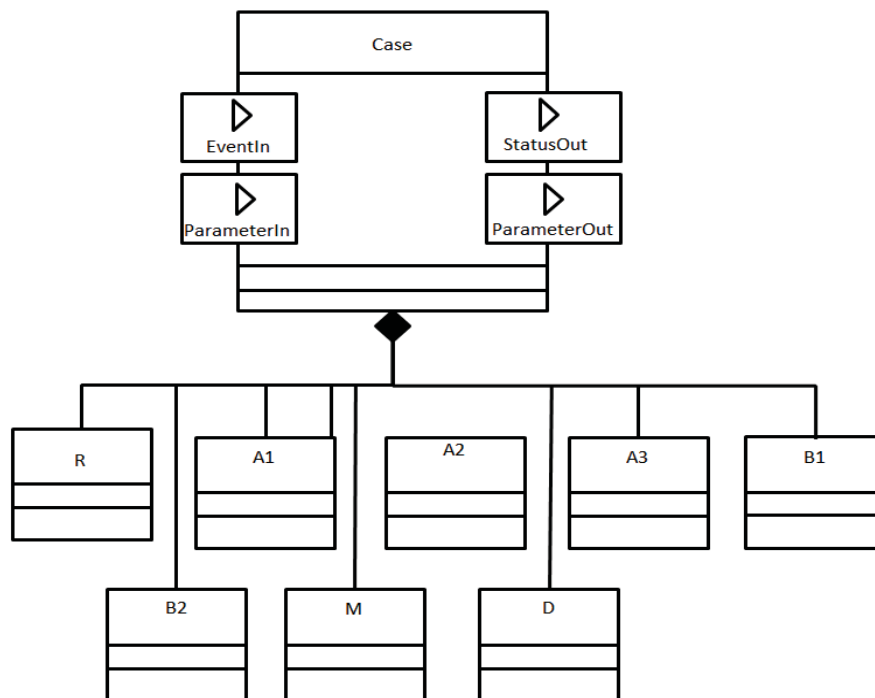


Figure 6-6: The graphical representation of the structure of the as-is HiLLS model

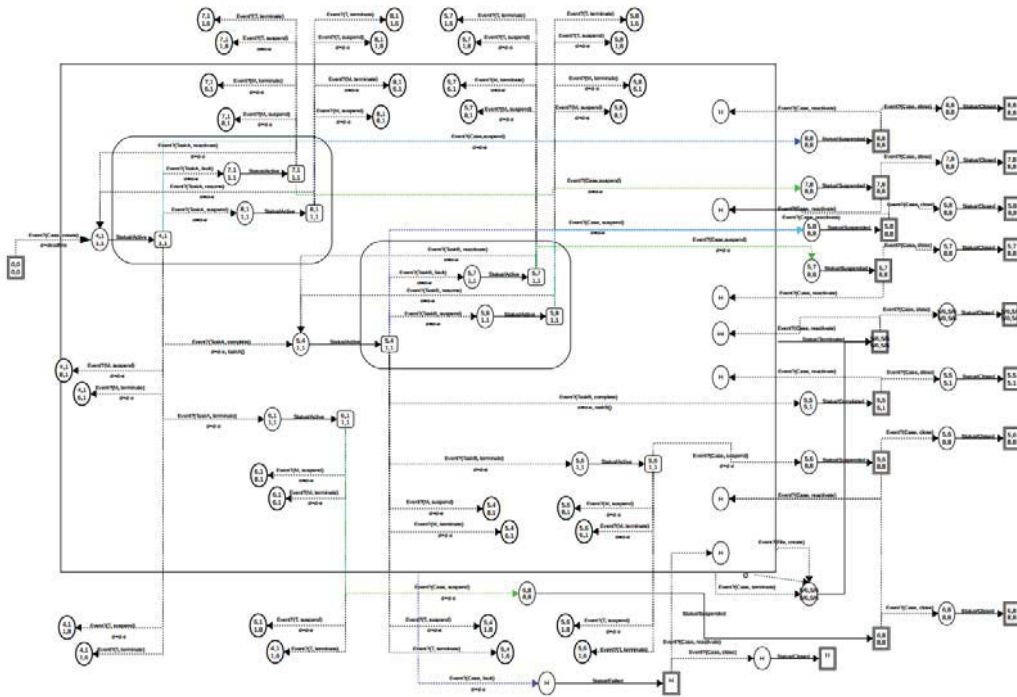


Figure 6-7: The partial graphical behavior representation of the as-is HiLLS model

Moreover, the intermediate EF is generated as well on the basis of the IEF generation rules we have specified in chapter 4. As we explained in the case study in section 4.3.6, the IEF of this case model is generated in the same manner, as given in Figure 6-8 which shows the structure of the EventGenerator.

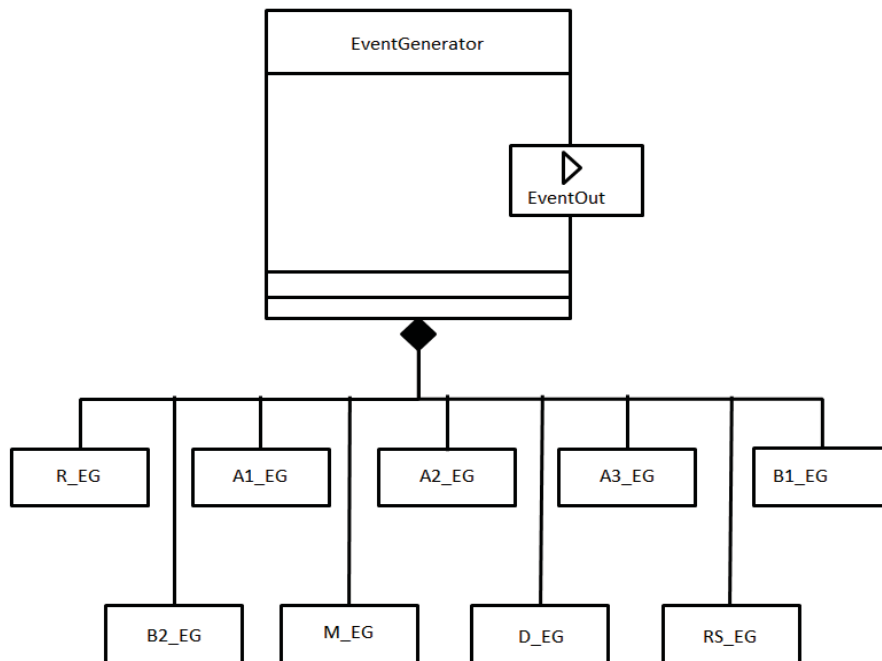


Figure 6-8: The event generator Hills model

The couplings are specified in the configuration of the EventGenerator as following:

- *R_EG.EventOut = RS_EG.EventIn*
- *RS_EG.EventOut = A1_EG.EventIn*
- *RS_EG.EventOut = B1_EG.EventIn*
- *A1_EG.EventOut = A2_EG.EventIn*
- *A2_EG.EventOut = A3_EG.EventIn*
- *B1_EG.EventOut = M_EG.EventIn*
- *A3_EG.EventOut = M_EG.EventIn*
- *M_EG.EventOut = B2_EG.EventIn*
- *B2_EG.EventOut = D_EG.EventIn*
- *R_EG.EventOut = EventGenerator.EventOut*
- *RS_EG.EventOut = EventGenerator.EventOut*
- *A1_EG.EventOut = EventGenerator.EventOut*
- *A2_EG.EventOut = EventGenerator.EventOut*
- *A3_EG.EventOut = EventGenerator.EventOut*
- *B1_EG.EventOut = EventGenerator.EventOut*
- *B2_EG.EventOut = EventGenerator.EventOut*
- *M_EG.EventOut = EventGenerator.EventOut*
- *D_EG.EventOut = EventGenerator.EventOut*

For each item event generator (i.e., R_EG, RS_EG, A1_EG, etc.), the structure and the behavior are defined by our IEF generation rules. In the figure below we give the RS_EG HSystem's coupling information graphically as an example.

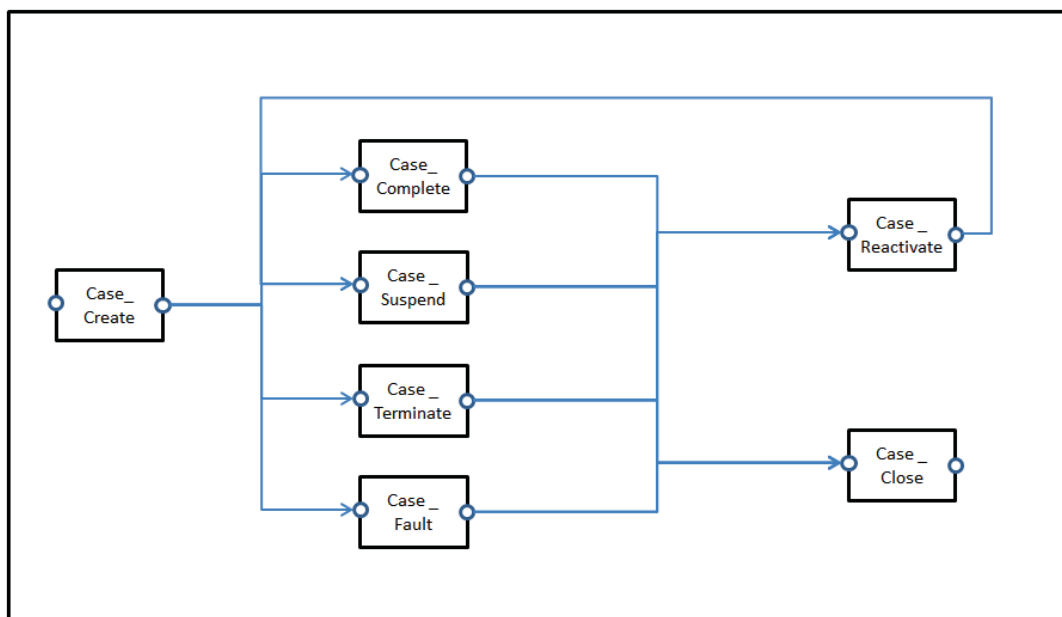


Figure 6-9: The RS_EG HiLLS model's coupling information

The figure above gives the configurations and the configuration transitions of the Case_Suspend atomic HSystem as an example.

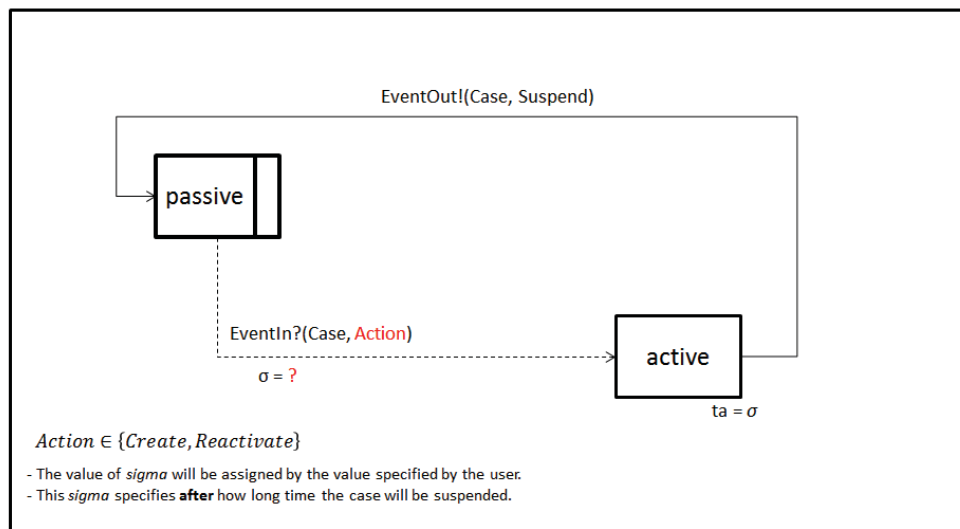


Figure 6-10: The RS_Suspend atomic HiLLS model

As we can see, there are some information missing: it left to the case worker to complete once the IEF is generated. As explained, not all the possible events are necessary for a simulation practice. Once all the complementary information is added, the final EF will be obtained. For example, based on the information stored in their event log, the time used to perform each task and the human resource allocation is listed in the figure below. The value in the column Time is a uniform distributed random number, where the two numbers in the bracket are the lower and upper bounds from left to right, respectively.

Index (to generate)	Event (to select)	Trigger (to specify)	Time (ta) (to specify)
1	(Case, Create)	-	[0, 0]
2	(Request, Create)	(Case, Create)	[1, 20]
3	(A1, Complete)	(Case, Create)	[8, 10]
4	(A2, Complete)	(A1, Complete)	[10, 14]
5	(A3, Complete)	(A2, Complete)	[4, 6]
6	(B1, Complete)	(Case, Create)	[5, 7]
7	(B2, Complete)	(A3, Complete) (B1, Complete)	[12, 18]
8	(Case, Suspend)	(Case, Create)	[1000, 1300]
9	(Case, Reactivate)	(Case, Suspend)	[200, 300]
10	(Case, Suspend)	(Case, Reactivate)	[500, 520]
11	(Case, Reactivate)	(Case, Suspend)	[200, 300]
12	(Case, Close)	(Case, Create)	[4500, 4500]

Figure 6-11: The complementary information

With such information plus the case workers decisions about what events are considered in their experiments, the final EF can be obtained. Figure 6-12 shows what the Event Generator in the EF looks like, and Figure 6-13 gives an example illustrating the system behavior of a HSystem, A1Complete, contained in A1EventGenerator (*A1, complete*). There are two configurations, and the passive one with a wide border indicates that it is the original configuration that the system stays in at the beginning of the simulation. For other atomic HSystems defined, they behave in the same manner, with different assigning value to σ .

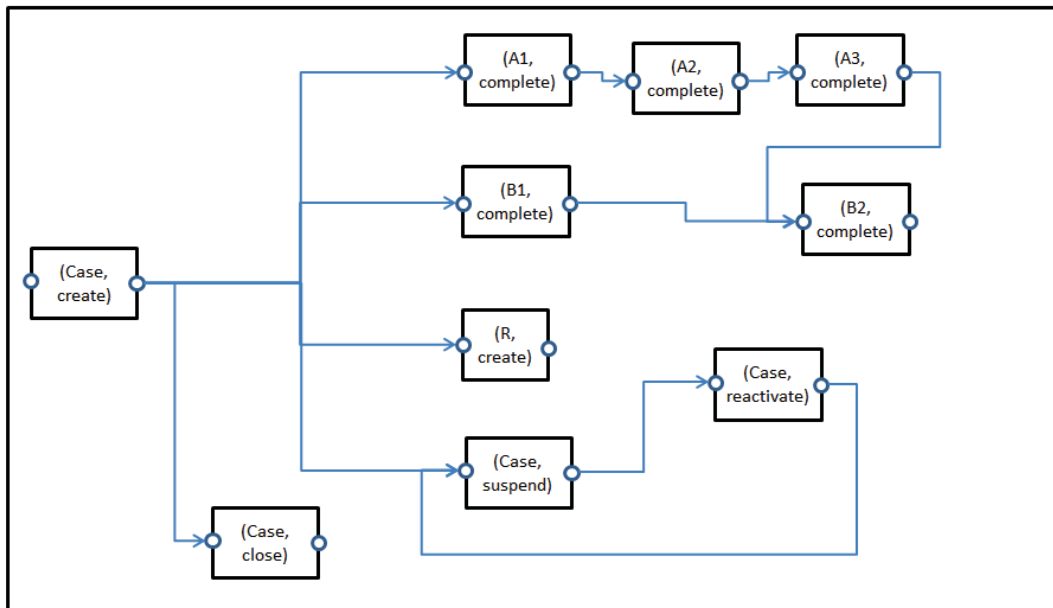


Figure 6-12: The event generator

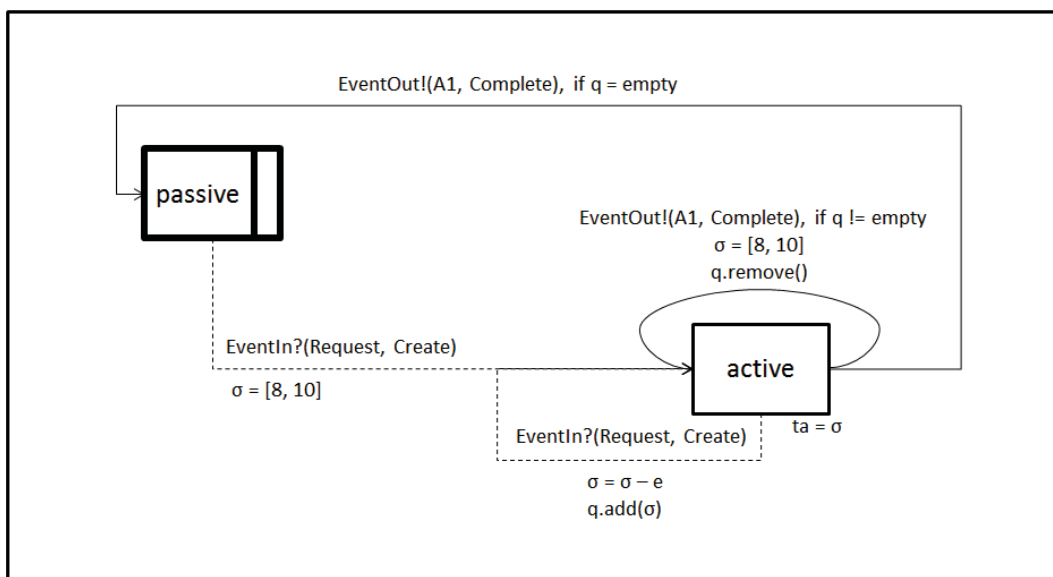


Figure 6-13: The system behavior of A1Complete

6.5 To-be Model

The managers' intention is to merge the two separate units together in order to save cost and improve efficiency and effectiveness. To this end, they remove repetitive tasks performed in both units, and rearrange the human resources. In addition, the case managers add one additional non-activity element in the case model, as given in the figure below: a TimerEventListener (*T*) defining a deadline for completing the case. The equivalent HiLLS model is given in Figure 6-15 in a tree-view HiLLS model editor.

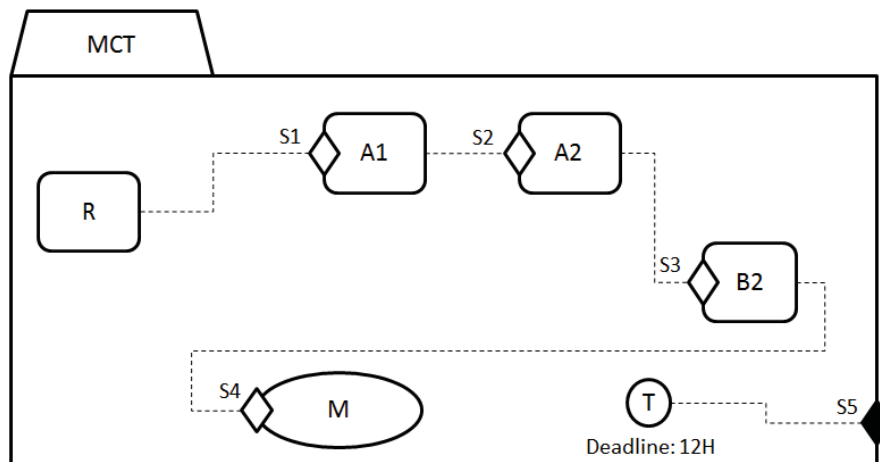


Figure 6-14: The CMMN to-be case model

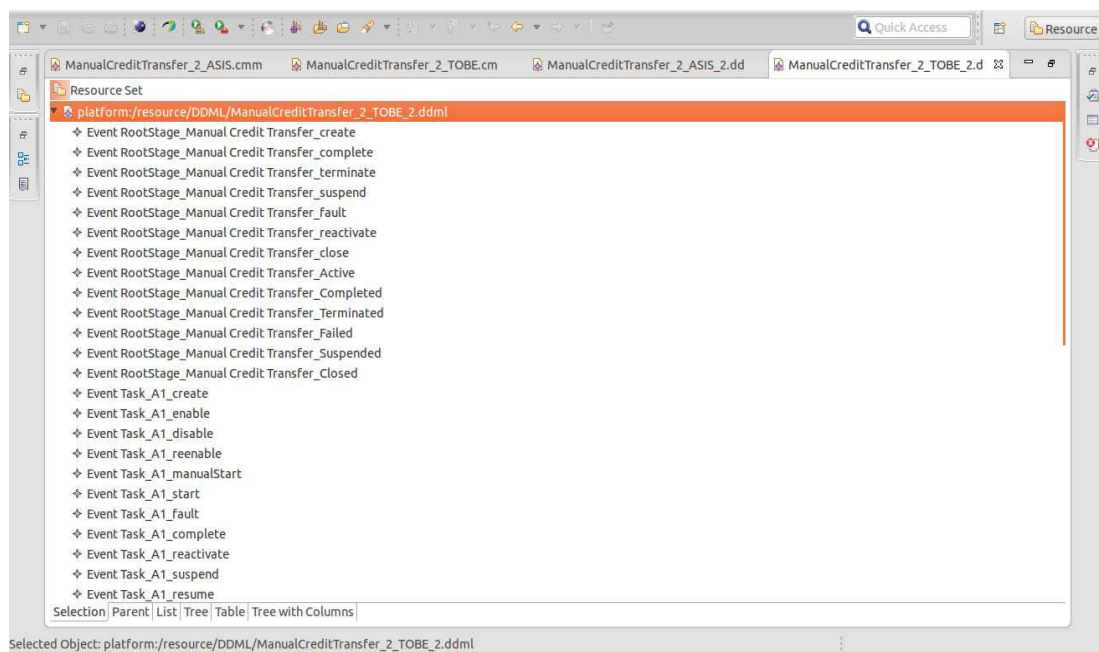


Figure 6-15: The HiLLS to-be model in the tree-view editor

The related information for the to-be situation is updated in accordance with the to-be model, and listed in the table below.

Table 6-2: Information for the to-be model

Task	Employees Required	Time Required per Employee
R	-	[0, 0]
A1	2	[4, 6]
A2	4	[7, 11]
B2	5	[10, 14]

The final EF is similar with the one for the as-is model, except that the *Event Generator* has different atomic event generators compared with the as-is situation (due to the fact that in the to-be situation the number of elements concerned is different). To this reason, we will not repeat the same process here.

6.6 Results Analysis

There are 40 case requests in the simulation practice, and the goal is to anticipate if the change of units merging is worthy or not.

The results from the simulation presented here are interpreted mainly from two perspectives: *Lean* and *TOC*. From the former perspective managers can have a global view on how effective and efficient their operation proceeds, and identify if there exists any waste within their processes. Related performance metrics include *lead-time* (the amount of time a product takes to flow completely through the process), *work-in-progress* (WIP, the number of work items being started but not yet finished), *backlog* (the number of work items waiting between different work steps), *idle-time* (the non-productive time of employees), *handoffs* (the number of times the information/product passes from one employee to another), etc, as we have explained in section 4.4. From the *TOC* point of view, it is possible to identify if any bottleneck exists within their processes. The metrics are *backlog* and *idle-time* due to the fact that if a work step has a huge *backlog*, or the one downstream has a long *idle-time*, then this work step is a bottlenecks [Goldratt et al. 1992].

Figure 6-16 shows the *backlog* in the as-is model simulation, from which we can identify that the work step having the most *backlog* is *A1*. However, from the *idle-time* point of view in Figure 6-17, *A2* is also a bottleneck due to the fact that employees at step *A3* have plenty of time for waiting. Consequently, *A1* and *A2* are the bottlenecks that slow other steps down.

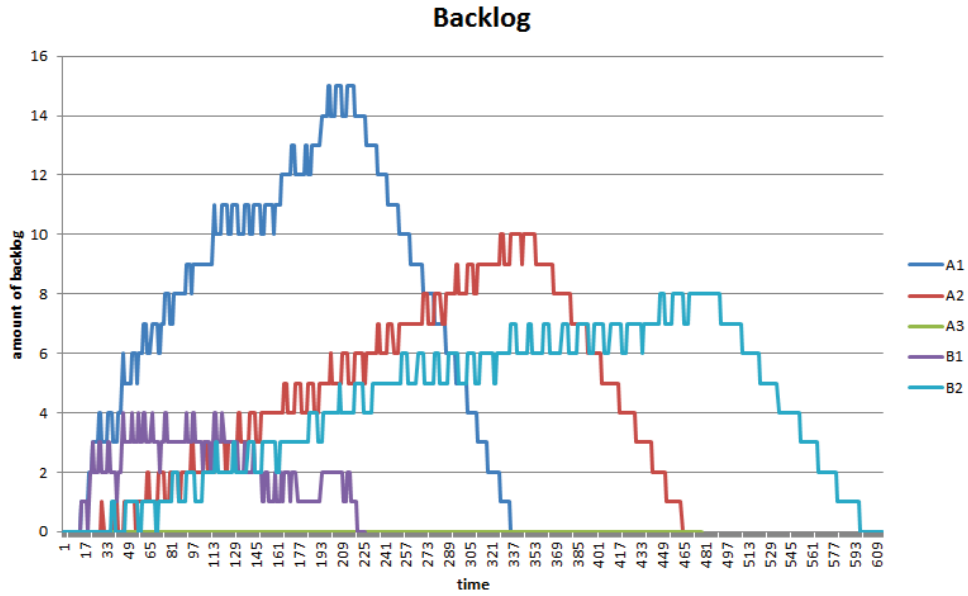


Figure 6-16: The backlog in the as-is model

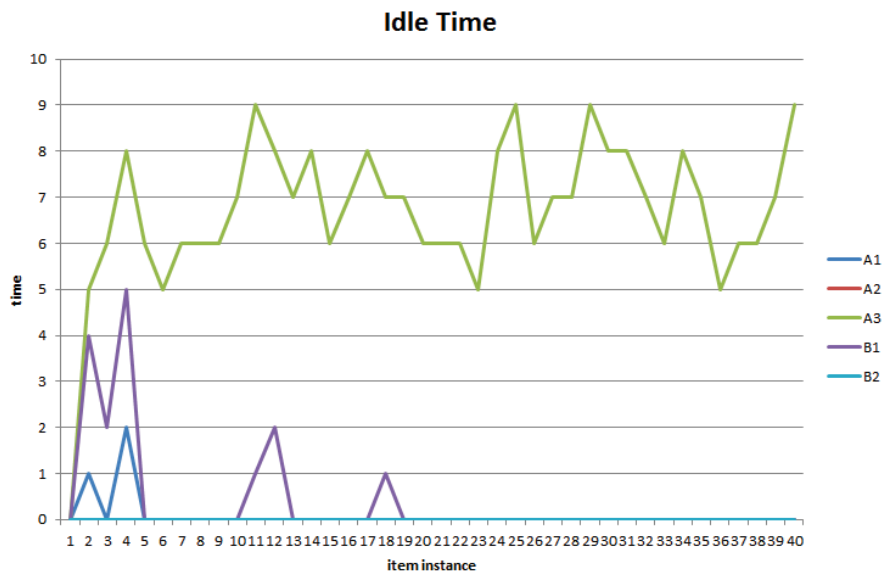


Figure 6-17: The idle time in the as-is model

However, in the to-be situation, as given in Figure 6-18 and Figure 6-19, *A1* is no longer the constraint, and it is *A2* that becomes the new bottleneck from the *backlog* point of view. However, from the *idle-time* perspective the bottleneck becomes the incoming orders, i.e., the constraint now exists in the market, on longer in the system.

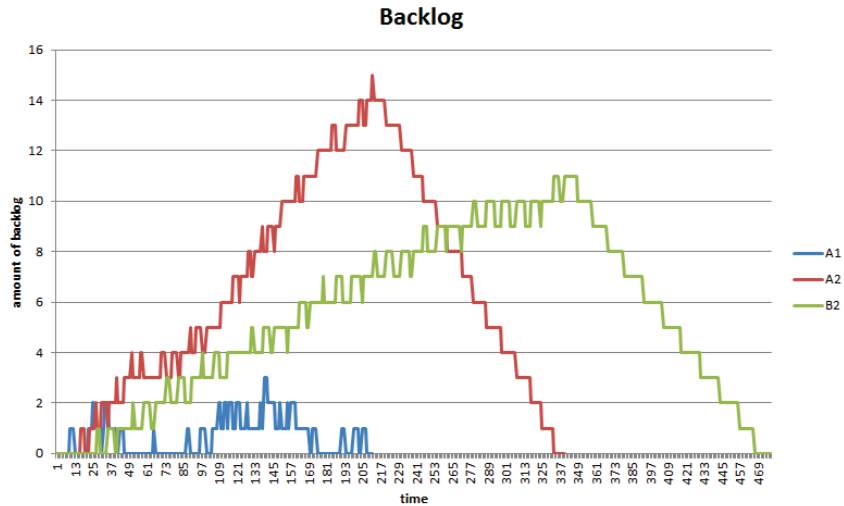


Figure 6-18: The backlog in the to-be model

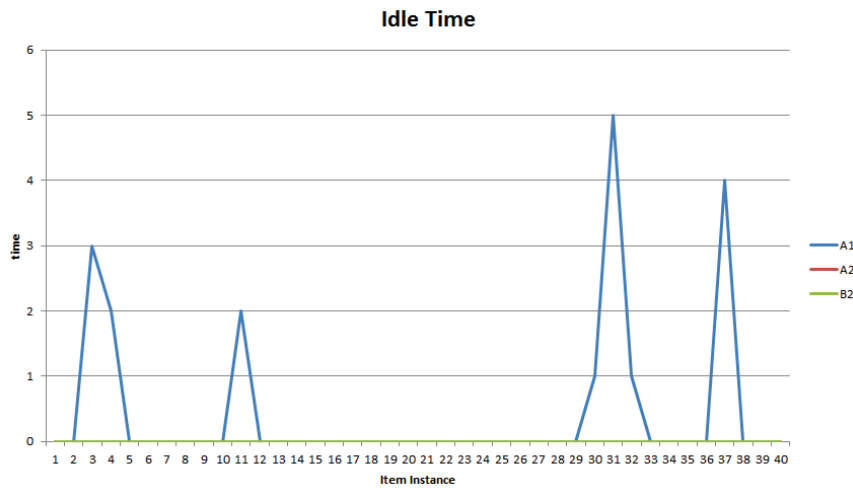


Figure 6-19: The idle time in the to-be model

[Goldratt et al. 1992] pointed out that there will always exist at least one bottleneck within organizations, the key point is that whenever erases one bottleneck, make sure the rest of resources are well rearranged, and the performance improves. We can verify the effectiveness and efficiency of the changes from the perspective of lean. In the following we give some examples.

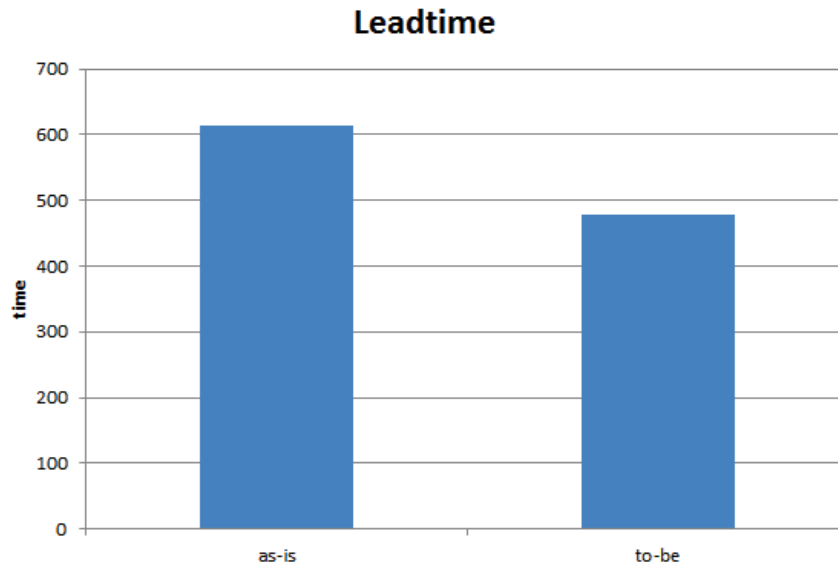


Figure 6-20: The comparison of lead-time

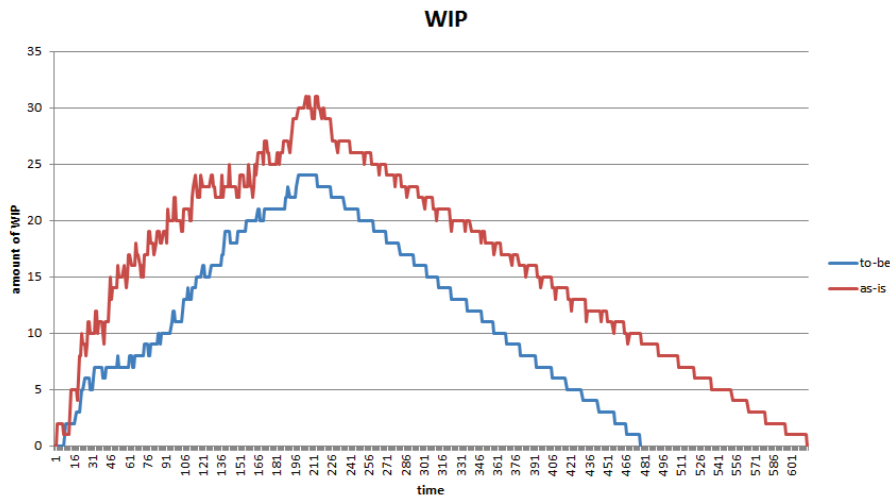


Figure 6-21: The comparison of WIPs

- Lead-time:** As shown in Figure 6-20, the Y-axis of the graph represents the time used for all MCT operations, and the X-axis represents the two situations: as-is and to-be. We can see that the *lead-time* in the to-be model is reduced due to the process restructuring and human resource re-allocation.
- WIP:** The Y-axis in Figure 6-21 refers to the number of *WIPs* existing during simulation, and the X-axis represents the time. From the comparison we can see that the amount of *WIPs* in the to-be situation is reduced due to the changes made.

From the results and analysis above, this case study shows that:

1. both the *lead-time* and the *WIPs* are reduced;
2. potential bottlenecks are well erased;
3. the operational performance is improved.

Consequently, in this case the merger of two units is beneficial, and the manager could make a more reasonable decision based on the simulation results and analysis. Moreover, he could try other change initiatives and run the simulation for each idea, then choose the one the most valuable to the business.

6.7 Conclusion

After given the detailed model transformation rules in chapter 4 and chapter 5, in this chapter we have shown a simple case study relating to the manual credit transfer operation in a bank. We started with the introduction of the case study, and we also explained why the manager wanted to change their case model and how the manager could test his change initiatives. Thanks to our CMD module, based on the recorded event logs the manager could easily get a CMMN case model automatically, instead of constructing one manually which is error-prone and requires additional costs. Once the as-is CMMN model is generated, the manager could start to make modifications and forecast the performance of the new to-be model by using our CMI module. His idea was to merge two teams as one, and the simulation results proved that this change was worthy in terms of process performance from the perspectives of lean and TOC, respectively. In order to make the best decision, he could try other change initiatives and run the simulation for each idea, then choose the one the most valuable to the business.

7. GENERAL CONCLUSION

7.1 Conclusion

In this these, we have explained in the beginning the background and context of the real problem to be solved: more and more case management processes emerge in today's business, but the research work in this area is not sufficient, especially in terms of case model discovery and case model improvement.

In order to make contribution to this challenge, in chapter 2 we have presented a literature review of the state of the art regarding case management modeling approaches and systems, as well as process improvement practices in case management. The case management modeling solutions were discussed under four categories: activity-based, information-based, communication-based, and hybrid activity and information based. We further summarized to which extent these approaches and systems support knowledge workers modeling processes and information involved in a case. We noticed that as modeling languages for case management, most of the approaches come with a graphical representation, but they lack a formal description for their abstract syntax and operational semantics. It is reasonable to consider both the process and information as the first-class citizens when modeling cases, due to case management features, and knowledge workers should be able to cooperate with each other, in a collaborative manner. However, most of the proposed modeling languages focus on only one aspect. We further observed that as commercial case management systems, most of them offer a case modeling language with syntaxes (both abstract and concrete) and operational semantics defined. None of the approaches were developed on the basis of CMMN. In terms of process simulation, process improvement and process discovery, not a lot of noticeable contributions can be found in the literature.

In chapter 3, we introduced the theories and techniques used in this these. We started by explaining the basic components of a system modeling language since the system modeling language was what we used for building our solutions. Then we introduced CMMN, the standard modeling language for case management, including the abstract and concrete syntaxes, and its operational semantics. We also introduced HiLLS, a system modeling language we have adopted for constructing multi-aspect system models. Moreover, we explained the concepts and approaches regarding business process management, including process analytics, process improvement, process reengineering, lean, and TOC. Such concepts and approaches are widely adopted in business domain to help organizations govern and improve their process performance. We also reviewed how organizations used simulation as an approach to predict the potential impacts of proposed process changes, as well as examine and compare all proposed improvement alternatives without actually change the exiting processes. In addition, we also introduced process discovery in section 0, a process approach to explore and construct business process models by merely analyzing raw

data stored in event logs, and the process tree modeling language, a modeling formalism used to depict models generated from process discovery. We presented model transformation in the end of that chapter, an essential approach in MDE to automatically generate target model(s) from source model(s) following a set of transformation rules.

In chapter 4, the CMI module was explained. Essentially, case workers start with case models, which later will be transformed into HiLLS models in order to allow case workers conduct case model simulations. As we have explained in the beginning of this chapter before, the translations between a CMMN model and its targeted HiLLS model are not simply n-n mappings between their meta-models. Especially for the configurations and the transitions between them, the rules must take into account several conditions, e.g., each element's state, sentry state, etc. To this reason, we have adopted several manners to formally specify the CMMN to HiLLS model transformation rules. Regarding to the system structure, we have used tables in which the mappings between CMMN and HiLLS metamodels were defined. In terms of system behavior, which concerns configurations and their transitions, we have used mathematic formulas and tables together to specify how *Configurations* are generated, as well as how they transit from one to another. The rules of configuration transitions serve as the pre-conditions when executing HiLLS model simulations. In addition, we have also proposed a semi-automatic mechanism to generate an EF for a case model, which concerns generating event models, event generator models, as well as an analyzer model. This analyzer which will receive and compute the values of performance metrics we have defined from different point of view: Lean and TOC.

The CMD module was detailed in chapter 5. This CMD module aims at analyzing event logs and constructing case models on the basis of recorded process information. This module should serve as a case mode discovering tool for case workers: case workers start with feeding recorded process information into this module and will obtain a corresponding (as-is) case model; after that, thanks to the CMI module they could begin modifying the as-as case model and forecasting the performance of the to-be models and analyzing the real value of their model modifications. Due to the fact that a gap exists between these process discovery modeling specifications and the de-facto modeling standard for CM (i.e., CMMN), we proposed in our CMD module the model transformation from Process Tree to CMMN. Essentially, the meta-model of Process Tree concern Nodes and Edges, where the former relate to different types of tasks and the latter refers to the logical relationships between Nodes. Consequently, we have defined the mappings from Nodes to CMMN PlanItems and from Edges to the connections between PlanItems, respectively. Moreover, in order to ensure that the model transformation could be executed effectively when dealing with a massive quantity of process information, we also defined an algorithm with mapped all the nodes and all the edges in a linear time complexity manner.

In order to provide a prove to our approach, in chapter 6 we have selected a case study to show that how case workers can benefit from our two module in managing their daily case work. This case study might be a simple one in a certain sense since we could not find a real complex case study with real data: most of such cases (e.g., the projects in Michelin) are confidential, and even we claimed to erase all sensitive parts the managers were still not agreed that we could use their real cases.

7.2 Perspectives

Next step, due to the limit of the IEF generated, a more efficient way could be established in order to not generate all possible element events but only the ones case workers need at the first time. This will save some time in terms of establishing an experimental frame for a case model simulation. Besides, we will focus on how to conduct static analysis on case models using formal analysis. This part is not yet mature for this moment. We mainly focused on the dynamic analysis part by using M&S. Moreover, it will be interesting to explore how business intelligence can be used in CM when performing both static and dynamic model analysis together. Another point is, CM is mainly focused on unstructured processes, and BPM is for structured processes. It will be beneficial to use both CM and BPM together in today's business processes since both structured and unstructured processes are already being observed.

REFERENCE

- Ajay, K. (2013). *Managing Unpredictability using BPM for Adaptive Case Management*. ORACLE.
- Aliyu, H. O., Maiga, O., & Traoré, M. K. (2015). A Framework for Discrete Event Systems Enactment. In *Proc. ESM 2015*.
- Anastas, J. W., & Clark, E. J. (2013). *NASW standards for Social Work Case Management*. (N. A. of S. Workers, Ed.). Washington DC: SocialWorkers.org.
- April, J., Better, M., Glover, F., Kelly, J., & Laguna, M. (2006). Enhancing business process management with simulation optimization. In *Proceedings of the 38th conference on Winter simulation* (pp. 642–649). Winter Simulation Conference. <http://doi.org/10.1109/WSC.2006.323141>
- Barnett, M. W. (2003). Modeling & Simulation in Business Process Management. *BP Trends Newsletter, White Papers & Technical Briefs*, 10(1).
- Bézivin, J., Jouault, F., & Touzet, D. (2005). An introduction to the ATLAS Model Management Architecture. *Research Report, LINA (05-01)*, 5(1), 10–49.
- Bhattacharya, K., Caswell, N. S., Kumaran, S., Nigam, A., & Wu, F. Y. (2007). Artifact-centered operational modeling: Lessons from customer engagements. *IBM Systems Journal*, 46(4), 703–721. <http://doi.org/10.1147/sj.464.0703>
- Biehl, M. (2010). Literature study on model transformations. *Royal Institute of Technology, Tech. Rep. ISRN/KTH/MMK*, 291.
- Buijs, J. (2014). *Flexible Evolutionary Algorithms for Mining Structured Process Models*. Technische Universiteit Eindhoven.
- Cafasso, R. (1993). Rethinking reengineering. *Computer World*, 15, 102–105.
- Cao, G., Clarke, S., & Lehaney, B. (2001). A critique of BPR from a holistic perspective. *Business Process Management Journal*, 7(4), 332–339. <http://doi.org/10.1108/EUM0000000005732>
- Castellanos, M., Alves de Medeiros, A. K., Mendling, J., Weber, B., & Weitjers, A. (2009). Business Process Intelligence. In *Handbook of research on business process modeling* (pp. 456–480). Information Science Reference, Hershey, PA.
- Chow, A. C. H., & Zeigler, B. P. (1994). Parallel DEVS: a parallel, hierarchical, modular, modeling formalism. In *Proceedings of the 26th conference on Winter simulation* (pp. 716–722). Society for Computer Simulation International. <http://doi.org/10.1109/WSC.1994.717419>
- Clair, C. Le, & Miers, D. (2014). *The Forrester WaveTM: Dynamic Case Management*. Forrester Research, Inc.
- Clair, C. Le, & Moore, C. (2009). *Dynamic Case Management- An Old Idea Catches New Fire*. Forrester Research, Inc.
- Clark, T., Sammut, P., & Willans, J. (2015). *Applied metamodelling: a foundation for language driven development*. arXiv.org.
- Clauberg, K., & Thomas, W. (2013). *BPM and Simulation*. Signavio, Inc.

- Committee, M. A. (1999). Theory of Constraints (TOC) Management System Fundamentals. In *Statements on Management Accounting* (p. 36). Montvale, NJ, USA: the Institute of Management Accountants (IMA).
- Conforti, R., Dumas, M., García-Bañuelos, L., & La Rosa, M. (2016). BPMN Miner: Automated discovery of BPMN process models with hierarchical structure. *Information Systems*, 56, 284–303. <http://doi.org/10.1016/j.is.2015.07.004>
- Davenport, T. H. (2005). *Thinking for a Living: How to Get Better Performances and Results from Knowledge Workers*. Harvard Business School.
- Davenport, T. H., & Short, J. E. (1990). *The new industrial engineering: Information technology and business process redesign*. Center for Information Systems Research, Massachusetts Institute of Technology, Sloan School of Management.
- De Lara, J., & Vangheluwe, H. (2002). Atom3: A tool for multi-formalism and meta-modelling. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 2, pp. 174–188). Springer. http://doi.org/10.1007/3-540-45923-5_12
- de Man, H. (2009a). Case Management : A Review of Modeling Approaches. *BP Trends, January, 2009*, 1–17.
- de Man, H. (2009b). Case Management: Cordys Approach. *BP Trends, February, (February)*, 1–13.
- Dehayni, M., Barbar, K., Awada, A., & Smaili, M. (2009). Some model transformation approaches: a qualitative critical review. *Journal of Applied Sciences Research*, 5(11), 1957–1965.
- Deming, W. E. (1986). *Out of the crisis*. Massachusetts Institute of Technology, Cambridge, MA.
- Drucker, P. F. (1999). Knowledge-Worker Productivity: The Biggest Challenge. *California Management Review*, 41(2), 79–94.
- Drucker, P. F. (2011a). *Landmarks of Tomorrow: A report on the new*. Transaction Publishers.
- Drucker, P. F. (2011b). *The Age of Discontinuity: Guidelines to Our Changing Society*. Transaction Publishers.
- Eck, M. L. van, Buijs, J. C. A. M., & Dongen, B. F. van. (2014). Genetic process mining: Alignment-based process model mutation. In *Business Process Management Workshops* (pp. 291–303). Springer. http://doi.org/10.1007/978-3-319-15895-2_25
- Ferhat, E., Moharram, C., & Geylani, K. (2015). *Review of Model-to-Model Transformation Approaches and Technologies*. AG Eindhoven.
- Forster, F. (2006). The Idea behind Business Process Improvement: Toward a Business Process Improvement Pattern Framework. *BP Trends, 2006(April)*, 1–14.
- Garrison, R. H., Noreen, E. W., & Brewer, P. C. (2010). Managerial accounting. *Issues in Accounting Education*, 25(4), 792–793.
- Goksoy, A., Ozsoy, B., & Vayvay, O. (2012). Business Process Reengineering: Strategic Tool for Managing Organizational Change an Application in a

- Multinational Company. *International Journal of Business and Management*, 7(2), 89–112. <http://doi.org/10.5539/ijbm.v7n2p89>
- Goldratt, E. M., Cox, J., & Whitford, D. (1992). *The goal: a process of ongoing improvement*. North River Press Great Barrington, MA.
- Grudzińska-Kuna, A. (2013). Supporting knowledge workers: case management model and notation (CMMN). *Information Systems in Management*, 2(1), 3–11.
- Harel, D., & Rumpe, B. (2004). Meaningful modeling: What’s the semantics of “semantics”? *Computer*, 37(10), 64–72. <http://doi.org/10.1109/MC.2004.172>
- Harrison-Broninski, K. (2005). *Human Interactions: The Heart and Soul of Business Process Management: how People Really Work and how They Can be Helpful to Work Better*. Meghan-Kiffer.
- Hassan, B., Judicaël, R., Gregory, Z., Yves, D., & Hadrien, B. (2016). SLMToolBox: enterprise service process modelling and simulation by coupling DEVS and services workflow. *International Journal of Simulation and Process Modelling*, 11(6), 453–467.
- Hindle, T. (2008). *Guide To Management Ideas and Gurus*. John Wiley & Sons.
- Hlupic, V. (2003). Business Process Modelling Using Discrete Event Simulation : Potential Benefits and Obstacles for Wider Use. *International Journal of Simulation: Systems, Science and Technology*, 7(1), 62–67.
- Holweg, M. (2007). The genealogy of lean production. *Journal of Operations Management*, 25(2), 420–437. <http://doi.org/10.1016/j.jom.2006.04.001>
- Huber, P. (2008). *The Model Transformation Language Jungle - An Evaluation and Extension of Existing Approaches*. Vienna University of Technology.
- Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F., Hobson, S., ... Vaculin, R. (2010). Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *Web services and formal methods* (Vol. 6551, pp. 1–24). Springer. http://doi.org/10.1007/978-3-642-19589-1_1
- Ighoroje, U. B., Maiga, O., & Traoré, M. K. (2012). The DEVS-driven modeling language: syntax and semantics definition by meta-modeling and graph transformation. In *Proceedings of the Theory of Modeling & Simulation: DEVS Integrative M&S Symposium* (p. 49). Society for Computer Simulation International.
- Javadian Kootanaee, A., Babu, K. N., & Talari, H. F. (2013). Just-In-Time Manufacturing System: From Introduction to Implement. *International Journal of Economics, Business and Finance*, 1(2), 7–25. <http://doi.org/10.2139/ssrn.2253243>
- Jones, D. T., & Bell, S. (2013). Lean thinking and knowledge work. *Cutter IT Journal*, 26(4), 6–10.
- Jouault, F., Freddy, A., Jean, B., & Kurtev, I. (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1), 31–39. <http://doi.org/10.1016/j.scico.2007.08.002>
- Jouault, F., & Kurtev, I. (2005). Transforming models with ATL. In *International Conference on Model Driven Engineering Languages and Systems* (pp. 128–138).

- Berlin, Heidelberg: Springer.
- Kaan, K., Reijers, H. A., Voorhoeve, M., van Eerde, W., & Boender, E. (2005). *Modeling support for the Case Management paradigm and Expectations Management in process innovation*. Eindhoven: Technische Universiteit Eindhoven.
- Kalenkova, A., De Leoni, M., & van der Aalst, W. M. P. (2014). Discovering, analyzing and enhancing BPMN models using ProM. In *BPM (Demos)* (pp. 36–40). Retrieved from <http://www.scopus.com/inward/record.url?eid=2-s2.0-84915766041&partnerID=40&md5=3cb028b66dc3bc6951284c616e0325b5>
- Kemsley, S. (2011). Case management and bpm. *BP Trends*. May, 1–8. Retrieved from http://it.braintribe.com/export/sites/it.braintribe.com/de/resources/downloads/Case_Management_White_Paper.pdf
- Kethers, S., & Schoop, M. (2000). Reassessment of the action workflow approach: empirical results. In *Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling LAP* (pp. 151–169). Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.9.6760&rep=rep1&type=pdf>
- Kettinger, W. J., Teng, J. T., & Guha, S. (1997). Business process change: A study of methodologies, techniques, and tools. *MIS Quarterly*, 55–80.
- Kleppe, A. (2008). *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education.
- Kleppe, A., Warmer, J., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional. [http://doi.org/10.1016/S0031-9406\(05\)65759-8](http://doi.org/10.1016/S0031-9406(05)65759-8)
- Kumar, V. (2010). *Customer Relationship Management*. Wiley Online Library.
- Kumaran, S., Nandi, P., Heath, T., Bhaskaran, K., & Das, R. (2003). ADoc-oriented programming. In *Proceedings - 2003 Symposium on Applications and the Internet, SAINT 2003* (pp. 334–341). IEEE. <http://doi.org/10.1109/SAINT.2003.1183067>
- Künzle, V., & Reichert, M. (2011). PHILharmonicFlows: Towards a framework for object-aware process management. In *Journal of Software Maintenance and Evolution* (Vol. 23, pp. 205–244). Wiley Online Library. <http://doi.org/10.1002/smr.524>
- Kurtev, I. (2007). State of the art of QVT: A model transformation language standard. In *Applications of graph transformations with industrial relevance* (pp. 377–393). Springer. <http://doi.org/10.1007/978-3-540-89020-1-26>
- Leemans, S. J. J., Fahland, D., & van der Aalst, W. M. P. (2013). Discovering Block-Structured Process Models From Event Logs - A Constructive Approach. In *Application and Theory of Petri Nets and Concurrency* (pp. 311–329). Springer.

- Lúcio, L., Amrani, M., Dingel, J., Lambers, L., Salay, R., Selim, G. M. K., ...
 Wimmer, M. (2016). Model transformation intents and their properties. *Software and Systems Modeling*, 15(3), 647–684.
<http://doi.org/10.1007/s10270-014-0429-x>
- Maiga, O. (2015). *An integrated language for the specification, simulation, formal analysis and enactment of discrete event systems*. Université Blaise Pascal, Université de BAMAKO.
- Maiga, O., Aliyu, H. O., & Traore, M. K. (2015). A NEW APPROACH TO MODELING DYNAMIC STRUCTURE SYSTEMS. In *Proceedings of the 2015 European Simulation Modeling Conference*.
<http://doi.org/10.1017/CBO9781107415324.004>
- Maiga, O., Ighoroje, U. B., & Traore, M. K. (2012). DDML: A Support for Communication in M&S. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on* (pp. 238–243). IEEE. <http://doi.org/10.1109/WETICE.2012.112>
- Manolescu, D.-A., & Johnson, R. E. (2001). *Micro-workflow: a workflow architecture supporting compositional object-oriented software development*. University of Illinois at Urbana-Champaign.
- Maria, A. (1997). Introduction to modeling and simulation. In *Proceedings of the 29th conference on Winter simulation* (pp. 7–13). IEEE.
<http://doi.org/10.1109/WSC.2005.1574235>
- Marin, M., Hull, R., & Vaculín, R. (2012). Data Centric BPM and the Emerging Case Management Standard : A Short Survey Case Management. In *International Conference on Business Process Management* (pp. 24–30). Springer.
- Marjanovic, O. (2000). Supporting the “soft” side of business process reengineering. *Business Process Management Journal*, 6(1), 43–55.
<http://doi.org/10.1108/14637150010313339>
- May, M. (2005). Lean Thinking for Knowledge Work. *Quality Progress*, 38(6), 33–39.
- Mayer, R. J. (1992). IDEF1 Information Modeling. *Knowledge Based Systems*, 73.
 Retrieved from <http://www.staratel.com/iso/IDEF/IDEF1/IDEF1-1.pdf>
- Mayer, R. J., Painter, M. K., & de Witte, P. S. (1994). *IDEF Family of Methods for Concurrent Engineering and Business Re-engineering Applications*. College Station, Tex, USA: Knowledge Based Systems.
- McDonald, M. (2010). *Improving Business Processes*. Harvard Business Review Press.
- Mellor, S. J. (2004). *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional.
- Mens, T., Czarnecki, K., & van Gorp, P. (2006). A Taxonomy of Model Transformations. *Electronic Notes in Theoretical Computer Science*, 152, 125–142. <http://doi.org/10.1016/j.entcs.2005.10.021>
- Muehlen, M. zur, & Shapiro, R. (2010). Business Process Analytics. In *Handbook on Business Process Management, Vol. 2* (Vol. 2, pp. 137–157). Springer.

- <http://doi.org/10.3905/jai.2007.695270>
- Muscatello, J. R., Small, M. H., & Chen, I. J. (2003). Implementing enterprise resource planning (ERP) systems in small and midsize manufacturing firms. *International Journal of Operations & Production Management*, 23(8), 850–871. <http://doi.org/857768973>
- Newgen. (2013). *Newgen Case Management Framework*.
- Nigam, A., & Caswell, N. S. (2003). Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 428–445. <http://doi.org/10.1147/sj.423.0428>
- Nissen, M., & Levitt, R. (2002). Toward Simulation Models of Knowledge-Intensive Work Processes. *Center for Integrated Facility Engineering*.
- OMG. (2008). *MOF Model to Text Transformation LanguageTM (MOFM2TTM)*, 1.0. OMG.
- OMG. (2014). *Case Management Model and Notation*. OMG.
- Ould, M. A. (2005). *Business Process Management: a rigorous approach*. BCS, The Chartered Institute.
- Page, S. (2015). *The power of business process improvement: 10 simple steps to increase effectiveness, efficiency, and adaptability*. AMACOM Div American Mgmt Assn.
- Pega. (2013). *Pega Dynamic Case Management*. Pega.
- Peinl, R., & Maier, R. (2011). SimKnowledge - Analyzing impact of knowledge management measures on team organizations with multi agent-based simulation. *Information Systems Frontiers*, 13(5), 621–636. <http://doi.org/10.1007/s10796-010-9241-5>
- Peterson, E. T. (2006). *The Big Book of Key Performance Indicators*. Web analytics demystified.
- Piggott, B. (1995). Throughput accounting and the theory of constraints. *Management Accounting: Magazine for Chartered Management Accountants*, 73(7), 14–14.
- Radnor, Z. (2010). *Review of Business Process Improvement Methodologies in Public Services*. AIM Research.
- Rooze, E., Paapst, M., & Sombekke, J. (2007). eCase Management. *An International Study in Judicial Organizations*.
- Russo, D., Passacantando, F., Geppert, L., & Manca, L. (2013). Business Process Modeling and Efficiency Improvement through an Agent-Based Approach. *JOURNAL OF SYSTEMICS, CYBERNETICS AND INFORMATICS*, 11(8), 1–6.
- Rust, I. J. (2011). *Business Process Simulation by Management Consultants? Introduction of a new approach for business process modeling and simulation by management consultants*. TU Delft, Delft University of Technology.
- Sayer, N. J., & Williams, B. (2012). *Lean For Dummies*. John Wiley & Sons. <http://doi.org/10.1007/s13398-014-0173-7.2>
- Schunselaar, D. M. M., Verbeek, H. M. W., van Dongen, B. F., Leemans, S. J. J., & Buijs, J. C. A. M. (2014). *Process Tree Package*.
- Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5), 26–32.

- <http://doi.org/10.1109/MS.2003.1231147>
- Selic, B. (2009). The theory and practice of modeling language design for model-based software engineering—a personal perspective. In *International Summer School on Generative and Transformational Techniques in Software Engineering* (pp. 290–321). Springer.
- Slonneger, K., & Kurtz, B. L. (1995). *Formal syntax and semantics of programming languages*. Addison-Wesley Reading.
- Staats, B. R., Brunner, D. J., & Upton, D. M. (2011). Lean principles, learning, and knowledge work: Evidence from a software services provider. *Journal of Operations Management*, 29(5), 376–390.
<http://doi.org/10.1016/j.jom.2010.11.005>
- Staats, B. R., & Upton, D. M. (2011). Lean Knowledge Work. *Harvard Business Review*, 89(10), 100–110.
- Strahonja, V. (2006). Modeling legislation by using UML state machine diagrams. In *Canadian Conference on Electrical and Computer Engineering* (pp. 624–627). IEEE. <http://doi.org/10.1109/CCECE.2006.277286>
- Swanson, C. A., & Lankford, W. M. (1998). Just-In-Time Manufacturing. *Business Process Management Journal*, 4(4), 333–341. <http://doi.org/10.1093/jnci/djs056>
- Swenson, K. D. (2010). *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press Tampa.
<http://doi.org/http://www.openisbn.com/isbn/0929652126/>
- Swenson, K. D. (2014). Demo: Cognoscenti open source software for experimentation on adaptive case management approaches. In *Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International* (pp. 402–405). IEEE. <http://doi.org/10.1109/EDOCW.2014.67>
- Tapping, D. (2007). *The New Lean Pocket Guide: Tools for the Elimination of Waste!* MCS Media.
- Touraille, L., Traoré, M. K., & Hill, D. R. C. (2011). A model-driven software environment for modeling, simulation and analysis of complex systems. In *TMS-DEVS '11 Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium* (pp. 229–237). Society for Computer Simulation International.
- Traoré, M. K. (2008). SimStudio: a Next Generation Modeling and Simulation Framework. In *Proceedings of the First International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems* (p. 67). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
<http://doi.org/10.4108/ICST.SIMUTOOLS2008.3214>
- van der Aalst, W. M. P., Reijers, H. a., Weijters, a. J. M. M., van Dongen, B. F., Alves de Medeiros, A. K., Song, M., & Verbeek, H. M. W. (2007). Business process mining: An industrial application. *Information Systems*, 32(5), 713–732.
<http://doi.org/10.1016/j.is.2006.05.003>

- van der Aalst, W. M. P., Weske, M., & Grünbauer, D. (2005). Case Handling: a new Paradigm for Business Process Support. *Data & Knowledge Engineering*, 53(2), 129–162.
- Vanderfeesten, I. T. P., Reijers, H. A., & Aalst, W. M. P. van der. (2007). Case Handling Systems as Product Based Workflow Design Support. In *International Conference on Enterprise Information Systems* (pp. 187–198). Springer.
- Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., & van der Aalst, W. M. P. (2009). ProM: The Process Mining Toolkit. *BPM (Demos)*, 489(31), 2.
- Wainer, G. (2013). DEVS TOOLS. Retrieved June 8, 2016, from <http://www.sce.carleton.ca/faculty/wainer/standard/tools.htm>
- Wang, J., & Kumar, A. (2005). A Framework for Document-Driven Workflow Systems. *Business Process Management*, 3649, 285–301. http://doi.org/10.1007/11538394_19
- Wang, S., & Traoré, M. K. (2014). DEVS-based case management. In *2014 Symposium on Theory of Modeling and Simulation - DEVS Integrative M and S Symposium, DEVS 2014; 2014 Spring Simulation Multi-Conference, SpringSim 2014* (Vol. 46, pp. 243–249). Retrieved from <http://www.scopus.com/inward/record.url?eid=2-s2.0-84901979215&partnerID=40&md5=d502fb29c9e87cb0ef144b576e96a477>
- Weerd, J. De, Broucke, V., M, S. K. L., & Filip, C. (2014). Bidimensional process discovery for mining BPMN models. In *Business Process Management Workshops* (Vol. 202, pp. 529–540). Springer. http://doi.org/10.1007/978-3-319-15895-2_45
- Weigand, H. (2005). *LAP: 10 years in retrospect. The Language Action Perspective on Communication Modelling, Kiruna, Sweden*. Retrieved from [http://www.vits.org/konferenser/lap2005/Paper 1-LAP.pdf](http://www.vits.org/konferenser/lap2005/Paper%201-LAP.pdf)
- Yan, W., Gregory, Z., Mamadou, T., & David, C. (2017). A tool for mining discrete event simulation model. In *WSC* (pp. 3066–3077).
- Zeigler, B. P., Praehofer, H., & Kim, T. G. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems* (Vol. 2). Academic press.
- Zhu, W., Kirchner, M., Ko, T., Oland, M., Prasad, B., Prentice, M., & Ruggiero, M. (2013). *Advanced Case Management with IBM Case Manager*. IBM.