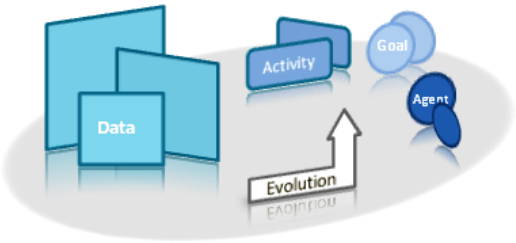
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Object-centered Process Modeling

Towards a Modeling Approach for Data-Intensive Systems

Rui Henriques

Dissertation for the degree of Master in
Information Systems and Computer Engineering

Adviser Ph.D. António Manuel Ferreira Rito da Silva

Jury Committee

President Ph.D. Maria dos Remédios Vaz Pereira Lopes Cravo
Evaluation Jury Ph.D. Diogo Manuel Ribeiro Ferreira
Adviser Ph.D. António Manuel Ferreira Rito da Silva

October 2010

Abstract

The increasingly dynamic and data-pushed landscape, where some systems operate, triggers the need to define new ways to model responsive processes that overcome limitations of traditional activity-centered approaches. New modeling approaches appeared grounded on the premise that the coordination of participants in data-intensive systems is pushed by object dependencies observed at the data level. This thesis structures the set of requirements required for the modeling of responsive data-intensive systems, uses this set to study emergent object-centered approaches from which it retrieves a set of principles that constrain the solution space and, finally, develops a solution that integrates such principles. This thesis updates and employs the event-driven theory to design the solution basis and maps the proposed object-centered process models in YAWL to assure their correctness and executability. Indicators and real-case confrontation reveal the adequacy of the new modeling approach to address the evolution needs of data-intensive systems.

Keywords:
Process Modeling
Object-orientation
Evolution
Data-intensive System

Resumo

O crescente contexto dinâmico e orientado aos dados, onde alguns sistemas operam, desencadeia a necessidade de definir novas formas de modelar processos ágeis que superem as limitações das abordagens tradicionais centradas em actividades. Novas abordagens de modelação aparecem fundamentadas na premissa de que a coordenação de elementos nestes sistemas é afectada por dependências observadas ao nível dos dados. Esta tese define um conjunto de requisitos necessários à modelação de sistemas fortemente orientados aos dados, deriva um conjunto de princípios do estudo de abordagens emergentes centradas no papel dos dados, e desenvolve uma linguagem que integra estes princípios. Esta tese define uma solução centrada na coordenação de objectos baseada em eventos para definir os modelos-alvo, e mapeia-os em modelos YAWL para garantir a sua correcção e exequibilidade. Análises baseadas em indicadores e na aplicação da abordagem em diferentes sectores revelam o seu potencial em suportar a evolução de sistemas fortemente orientados aos dados.

Palavras-chave:
Modelação de Processos
Orientação aos Objectos
Evolução
Sistemas fortemente orientados aos Dados

Acknowledgements

First, I would like to thank Professor António Rito da Silva for his incessant support, guidance and patient towards my way of working, and for always providing me with detailed and relevant suggestions that led to the practical achievement of this work.

I also acknowledge the role of IST and, in particular, of CODE and INOV in supporting my research.

Finally, I would like to dedicate this thesis:

to Miguel Henriques, my brother, Rui Henriques, my father, and Elsa Henriques, my mother, for all their Love, Will and Sacrifice. Ancestral journey mates;

to Maria Flávia de Monsaraz, for revealing me the meaning of Life. My source of inspiration;

to Master D.K. and A. Bailey, who taught me how to disclose the liberating Order of the Universe;

to José Augusto, João M. dos Santos, Gonçalo Ferreira, Francisco Maia, João Belchior, Carlos Antunes, Elsa Torres, Ricardo Louro and Marta Oliveira. Soul friends, whose paths crossed mine in a deep and irreversible way.

To them I owe the strength to approach this thesis with truth and dedication.

List of Acronyms

AM	Activity Model
ASF	Algebraic Specification Formalism
BE	Business Entity
***(WS-)*BPEL**	*(Web Services' extended)* Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modeling Notation
CEP	Complex Event Processing
COREPRO	COnfiguration-based RElease PROject
CRUDE	Create, Read, Update, Delete and Execute
EdBPM	Event-driven Business Process Management
GF	Global Financing
GM	Goal Model
HPMS	Human Process Management System
IT	Information Technology
LTL	Linear Temporal Logic
PBWS	Product-Based Workflow Support
PDM	Product Data Model
PM	Process Model
newYAWL	new Yet Another Workflow Language
NP-hard	Non-deterministic Polynomial-time hard
OLC	Object Life-Cycle model
OM	Object Model
RM	Rule-set Model
SDF	Syntax Definition Formalism
UML	Unified Modeling Language
WfMC	Workflow Management Coalition
XML	eXtensible Markup Language
YAWL	Yet Another Workflow Language

Notation

Some structures apart from the usual text, figures and tables are adopted in this work. Their use aims to better organize and emphasize the ideas to be expressed so its content can be easily assimilated by the reader.

Definitions

The introduction of key concepts are framed by a light gray box. Exemplifying:

Def. n: A **definition** is a passage describing, and possibly formalizing, the meaning of a concept.

Source Code

Algorithms are presented using pseudo-code with a caption framed by a green box. Exemplifying:

Algoritm 1: The Min-Error algorithm with an earliest first heuristic

```
Input: Set of tasks and processors
Output: Mapping of tasks to processors
foreach Task i do
    foreach Processor j do
        Calculate EW(i,j);
        if EW(i,j) ≤ EW(a,b) then
            a = i;
            b = j;
while Unscheduled tasks remaining do
    foreach Processor j do
        Calculate FT(a,j);
        if FT(a,j) ≤ FT(a,c) then
            c = j;
    Schedule(Task a on Processor c) ;
```

Table of Contents

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Contribution . . . 3
1.2.1 Publications . . . 3
1.3 Structure . . . 3

I Conceptual Foundations

2 Thesis Context **5**
2.1 The Systemic Context . . . 5
2.2 System Evolution . . . 6
2.3 Process Modeling . . . 7
2.4 The Role of Data . . . 8
2.5 Modeling Orientation . . . 9
2.6 Bridging the Concepts . . . 9

3 Thesis Problem **11**
3.1 The application scenario . . . 11
3.2 Limitations of traditional approaches . . . 12
3.2.1 Activity-centered Modeling Approaches . . . 12
3.2.2 Traditional Document-based Modeling Approaches . . . 14
3.2.3 Synthesis of Limitations . . . 14
3.3 The *FIVE* Requirements . . . 14

4 Thesis Statement **17**
4.1 Research Artifacts . . . 17
4.2 Research Goals . . . 17
4.3 Research Methodology . . . 18

II Findings

5 State-of-the-Art **19**
5.1 The Chosen Object-centered Approaches . . . 19
5.1.1 Advanced Document-based Modeling . . . 20
5.1.2 Artifact-centric Modeling . . . 21
5.1.3 Business Entities . . . 22
5.1.4 Product-based Workflow Support . . . 23
5.1.5 Data-driven Modeling . . . 25
5.1.6 Case Handling . . . 26

5.1.7 Proclets . . . 27
5.1.8 Object-aware Business Processes . . . 28
5.2 Other Emerging Approaches . . . 29
5.3 Discussion . . . 30

III Solution Architecture

6 Solution Basis **32**
6.1 System as Synchronized Set of State-based Entities . . . 32
6.2 The *Process* of Modeling Data-intensive Processes . . . 34
6.3 Solution Principles . . . 35
6.4 Principles and the Thesis Contribution . . . 37

7 Solution Derivation **38**
7.1 Data Access . . . 38
7.2 Data-state Reaction . . . 41
7.3 Data-based Coordination . . . 43
7.4 Data-based Granularity . . . 46
7.5 Data Modeling . . . 47

IV Development

8 Object-centered Models Formalization **50**
8.1 Object Models . . . 50
8.2 Activity Models . . . 54
8.3 Rule-Set Models . . . 57
8.4 Object-centered Process Models . . . 58
8.5 Object-centered Soundness Criteria . . . 60

9 Modeling Transformations **63**
9.1 Map to plain YAWL models . . . 63
9.2 Map to proclets-enriched YAWL models . . . 65
9.3 Multi-colored places addition . . . 67

V Validation

10 Proof of the Concept **69**

11 Practical Applicability **71**
11.1 General Performance . . . 71
11.2 Sector-oriented Applicability . . . 72
11.2.1 Financing . . . 72
11.2.2 Manufacturing . . . 72
11.2.3 Healthcare . . . 74
11.3 Comparison of the Approaches Performance . . . 76

12 Hypothesis Validation **78**
12.1 System Implications 78
12.2 Hypothesis Approval 80

VI Concluding Remarks

13 Conclusion **80**

14 Future Work **82**

VII Bibliography

VIII Appendices

15 Event-driven Solution **90**
15.1 Object-centered Systems are Event-driven by Nature 90
15.2 Advantages 91
15.2.1 Modeling Traceability and Correctness 91
15.2.2 Modeling Usability 91
15.2.3 Complex Event Processing and Modeling Abstractions 91
15.3 Notes on the Implementation Architecture 92

16 Thesis Boundary **93**
16.1 The Process Model Framework 93
16.2 How Requirements foster Evolution of Data-intensive Systems? 95

17 Implementation Notes **97**
17.1 The Application 97
17.2 Models Syntax and Transformations 98

18 GF Case Annexes **101**

List of Figures

1.1 Thesis Outline 4

2.1 Abstract data-intensive system 10
2.2 Abstract process model 10

3.1 Activity-centered process modeling landscape 11

5.1 Document-based simplified process model 20
5.2 Artifact-centric simplified process model 22
5.3 Simplified business operations model 23
5.4 Product-based simplified data model 24
5.5 Data-driven simplified data model 25
5.6 Case Handling process model 27
5.7 Proclets-based simplified process model 28
5.8 Object-aware simplified process model and types of collective enactment of instances . . 29

6.1 Object-centered Processes Initial Meta-Model 34
6.2 Basic Transformations on the Process of Modeling Object-centered Processes 35

7.1 The new landscape: integrating data in the process modeling landscape 39
7.2 Data-scope specification 39
7.3 *Form*, the center notion for the execution of object-centered process models 40
7.4 Data-scope by omission 41
7.5 Understanding the privilege access levels for an object-based authorization 41
7.6 How compound entities depend and affect the state of its internal entities 42
7.7 Definition of local precedences 42
7.8 Precedences specification recurring to data dependencies 43
7.9 The need for objects' encapsulation 44
7.10 Synchronization of processes derived from objects communication 44
7.11 Default rule-set models dynamically generated 45
7.12 Two different aggregation rules for instances that: *i)* share an upper parent, and *ii)* have similar properties 46
7.13 Zoom operations over different granular levels 47
7.14 Framing activity will collective assignment of data, agent and constraints 48

8.1 Simple Object Model 50
8.2 Two strategies for *inheritance* of data and constraints from a super-object model 52
8.3 Compound Object Model 52
8.4 Atomic Activity Models: plain and using compound states 55
8.5 Modularity of a Process Model based on its Governed Models 56
8.6 Compound Activity Model 56

8.7 Mimic of simple gateway-patterns recurring to rule-set models . . . 58
8.8 Notational convenience to depict advanced rule-set models . . . 58
8.9 Simple Process Model . . . 59
8.10 Compound Process Model with a Marking . . . 60
8.11 How rule-set models affect the soundness of object-centered models . . . 61
8.12 Transgress of *proper termination* criterion . . . 62

9.1 YAWL model generation and reduction . . . 64
9.2 Composition frames for the generated YAWL models . . . 65
9.3 Default rule-set models dynamically generated . . . 65
9.4 Derived proclets model from an object-centered process model . . . 66
9.5 Solution to the proclet's encapsulation problem: definition of a proclet mediator . . . 67
9.6 Multi-colored tokens . . . 68
9.7 Default rule-set models dynamically generated . . . 68

11.1 Object-centered Modeling of the *Global Financing* case . . . 74
11.2 Object-centered Modeling of the *Car Enginneering* case . . . 75
11.3 Object-centered Modeling of the *Clinic Diagnosis* case . . . 75

15.1 Event-driven nature of data-intensive systems . . . 91
15.2 Architecture for the event-driven and object-centered modeling landscape . . . 92

17.1 Extract of the algebraic equations used for the data-contexts definition . . . 98

18.1 Simple data model . . . 101
18.2 Automatic generation of object models skeleton . . . 101
18.3 Manual enrichment of objects with data-dependencies and life-cycles . . . 101
18.4 Dynamic life-cycle generation and manual definition of synchronization points between object instances . . . 102
18.5 Dynamic generation of rule-set models and of an object model sound net . . . 102
18.6 Dynamic derivation of activity execution behavior and default data-contexts (using default visibility criteria) . . . 102
18.7 Dynamic derivation of compound execution effects and manual definition of local orderings and of data-contexts . . . 103
18.8 Automatic derivation of the data-accessible objects (data-contexts can also be defined using object models) . . . 103
18.9 Dynamic derivation of object-centered process models based on object models . . . 103
18.10 Automatic completion of object-centered process models . . . 104
18.11 The mapping of object-centered process models into plain YAWL models and into a proclets' net . . . 104
18.12 Mediator strategy to enable the composition mechanisms within proclets . . . 104
18.13 Data model for the GF case . . . 105
18.14 Object placeholders defining GF points-of-synchronization . . . 105
18.15 GF object model (textual definition or views must be used to restrict complex graphical dependencies) . . . 105

List of Tables

3.1 Problems of Traditional Modeling Approaches in Data-Intensive Landscapes 15
3.2 Data-related Requirements for Object-centered Approaches 16

5.1 Chosen Object-centered Approaches 19
5.2 Evaluation of Advanced Document-based Modeling 21
5.3 Evaluation of Artifact-centric Modeling 22
5.4 Evaluation of Business Entities Modeling 23
5.5 Evaluation of Product-based Modeling 24
5.6 Evaluation of Data-driven Modeling 26
5.7 Evaluation of Case Handling 27
5.8 Evaluation of Proclets Modeling 28
5.9 Evaluation of Object-aware Modeling 30
5.10 Approaches Evaluation [subtitles: + answers; +/– partially answers; – not answers] 30
5.11 Brief Review of some of the Emergent Approaches 31

6.1 Principles for the FIVE Requirements 36
6.2 Source and Novelty of the proposed Principles 37

11.1 Results: Generic Performance of Data-aware Approaches (with an available implementation) 72
11.2 Domain-specific modeling requirements for different sectors 73
11.3 Discussion of the Results for Data-aware Approaches in comparison with Object-centered Modeling 77

12.1 Implications of the use of the object-centered approach to model data-intensive systems . 79

16.1 Process Model Framework's skeleton 93

List of Algorithms

1 The Min-Error algorithm with an earliest first heuristic v
6.1 The *Process* of Modeling Data-intensive Processes 35
7.1 Generation of Rule-set Models 45
8.1 Derivation of a Sound Net for Object Models Composition 53
8.2 Derivation of a Sound Process Net 59
9.1 Mapping of an Object-centered Process to a YAWL Process 64
9.2 Mapping of an Object-centered Process to a Proclets Net 66
17.1 Syntax specification of a *simplified* proclets net model 99
17.2 Brief extract of the algebraic equations for the mapping in plain YAWL models 100

1

Introduction

Evolution is the process of unfolding
a continually increasing power to respond to vibration.
– Alice Bailey, *The Consciousness of the Atom*

The operation of the Earth society system has become so complex that its elements have already lose the whole view and do not know what role their work plays in the overall scheme. These elements are new systems – nations, organizations or, at a more elementary level, agents – needed to carry out particular processes. All of these elements have been fuzzily cooperating to develop and deliver productions needed to keep their upper-systems evolving, i.e., continually increasing their ability to respond.

By decomposing complex systems in manageable subsystems, we are increasing the awareness at the subsystems operation level but offering *integration*, *adaptability* and *agility* challenges to the root system. Thus, to properly evolve, a system must be able to consistently and coherently operate as a unified system and to answer to internal and external changes in a timely manner. Hence, the evolution of a system depends on the system ability to coordinate its elements and their interaction with external systems through exchanges of productions. These exchanges enable the satisfaction of a set of goals that will increase the system effectiveness and efficiency to realize its purpose.

Since a system production is essentially an outcome of a set of actions performed in coordination by a set of system elements, the ability of a system to evolve is strongly determined by the ability of a system to continuously adjust its processes. Thus, to capture and adapt its processes, process models, conceptualizations that abstract and prescribe a system operation, are adopted.

Process models can create an environment for the evolution of a system if they foster: *i)* system integration by prescribing the functional, informational, instrumental and contextual relationships among the system elements; *ii)* system adaptability by promoting flexible and data-centered models; and *iii)* system agility by assuring, through the adequacy of the sources of changes to the type of system, that models' adaptation is performed in a timely manner [70].

An elementary look at a system as a coordinated composition of participants allows us to precisely identify the type of systems and source of evolution targeted by this thesis. A system participant can either be passive if is subjected to transformation during the execution of a set of system actions (e.g. an artifact), or active if performs actions aimed to change passive participants (e.g. an agent). At this scope, a data-intensive system can be defined as a system that highly depends on how its passive participants are defined, related and mediated among its elements to deliver a production to the environment.

Exemplifying, a health-care system operation, that relies on the state and mediation of patients, examinations, reports and historical clinics to deliver a diagnosis, *evolves* by changing the way these passive participants are related and constrained through its process models in order to abstract and prescribe the new desired operation.

The increasingly diversity, complexity, data-dependency, unstructured degree and real-time dynam-

ics of these processes turn the study of their models' evolution a specific and challenging task. Traditional approaches do not adequately answer this problem since fail to foster existing synergies between the system informational and functional views as a result of separating the modeling of processes from the modeling of system data, historically hidden behind applications [98].

This thesis is centered on how modeling approaches centered on passive participants structures, the so-called object-centered approaches, support the continuous adjustment of its process models. The research question can be formulated as following: *How effective are object-centered process models supporting the evolution of data-intensive systems?*

1.1 Motivation

The increasingly uncertain, dynamic and data-pushed landscape where data-intensive systems operate triggers challenges to process models *evolution*, either when processes need to be modeled from scratch (the revolutionary or clean-sheet approach [80][90]) or, as this work focuses, to organically adapt through local improvements (the evolutionary or reference model approach [80][74]).

Barriers for the evolution of data-intensive process models are pointed in [74][64]. Their resolution is particularly important for systems with particular complex data dependencies, systems that strongly rely on the role of passive participants to constrain their operation, as encountered in: scientific workflow systems [4], manufacturing systems [90][77], health-care systems [112], government systems [27] or insurance systems [3].

Common *problems* encountered by traditional approaches when modeling data-intensive processes include the context-isolated enactment of activities causing data-access challenges and a loss of the process global view, the absence of criteria for the activities granularity and the rigidity required to specify networks of activities when loosely-coupled, dynamic and data-based constraints foster models flexibility and expressivity.

Results from research [26][77] support the fact that system formalization with the data and process perspectives integrated reveals opportunities that disrupt traditional modeling discipline. The natural outcome orientation of many administrative and operational processes [106] turn the progress of single process instances not directly dependent on the execution of activities but reactive on data changes [64]. Contrasting to traditional modeling approaches which force the modeling of a system operation into single activity-based monolithic processes, objects capture the operation as a collection of intertwined loosely-coupled life-cycles running at different speeds [112][26] and coping with different levels of granularity [64]. There are also studies arguing that objects provide a natural basis to derive key performance indicators [26], to become the ground of process stakeholders vocabulary [26] and to define privilege access levels [35]. Linehan [67] states that a modeling centered in objects provides a natural way for modeling system constraints.

Since data-intensive processes rely on the premise that relations between the passive participants components implicitly define sub-process dependencies [64], new ways of dynamically support the evolution of these processes can be exploited.

1.2 Contribution

This thesis proposes an analysis of the potentialities and problems triggered by object-centered modeling approaches, to extract a set of principles, restrictions to process modeling freedom, on data-intensive process models, and to define a solution basis where those principles can coexist thus leveraging the ability of data-intensive systems to evolve.

Although several research exist in the scope of process modeling centered on objects [26][77] and on process models evolution [46][99], since existing approaches were developed to face only a small and specific set of concerns [64], their practical applicability is limited to a reduced set of domain-specific scenarios [64]. This seems unaccountable in the contemporary and digital era where, for instance, artificial intelligence systems and several organizations and many of their subsystems are truly data-intensive systems. This observation fosters the need to re-look to them from scratch in order to understand how their potentialities can be combined.

The target audience for this thesis comprises, firstly, the computer science research focused on workflow and process management. Also, researchers from data and knowledge management may have a potential interest as they respectively may find contributions on how system objects are modeled to support process modeling and how objects capture systems tacit knowledge. The software community can use the developed principles to engineer the logical systems for process management. Secondly, researchers focused on enterprise design and engineering have also here a good pretext to broad, integrate and challenge their understanding of these systems on the basis of object-centered models.

Concluding, this work propose validated directions to systematize and conceptualize the object-centered process modeling universe and develops an approach that can be used as a meta-guider and for principles integration on the modeling of data-intensive landscapes and as a catalyst for following increments from studies on tacit and data-intensive processes.

1.2.1 Publications

Hitherto, the results of this research were disseminated to the community through the following publication: Rui Henriques and António Rito Silva, 2010, *Object-centered Process Modeling: Principles to Model Data-intensive Systems*, Proceedings on the 4th International Workshop on Event-Driven BPM – EdBPM'10 (accepted and presented).

1.3 Structure

Accordingly to Fig.1.1, this thesis is divided into five logical parts.

First part, *Conceptual Foundations*, provides a structured context for the universe of discourse and depicts the thesis problem, statement and validation methodology.

Second part, *Related Work*, presents how existent contributions and emergent data-aware approaches answer to the requirements derived from the thesis problem.

Third part, *Solution Architecture*, incrementally describes the properties of the target object-centered modeling approach. Relying on the previous part's findings, it *i)* presents a set of principles that restrict the solution space, *ii)* develops an initial skeleton for their coexistence, and *iii)* derives the modeling approach using a formal reasoning guided by the introduced principles.

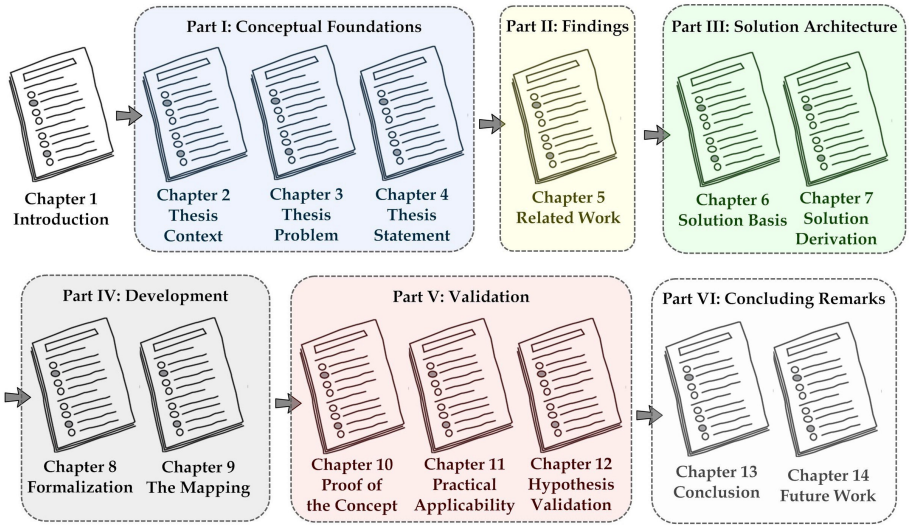


Figure 1.1: Thesis Outline

Fourth part, *Development*, guarantees the correctness and executability of the target models by formalizing and mapping them to an existing stable and expressive modeling language.

Fifth part, *Validation*, evaluates the research question in three distinct steps: *i)* description of the developed proof-of-the-concept to assess the executability of object-centered modeling, *ii)* practical applicability evaluation using metrics and case-based application in comparison to alternatives, and *iii)* retrieval of the main object-centered modeling implications to the evolution of data-intensive systems.

Finally, last part, *Concluding Remarks*, presents a set of theorems resulted from this work and possible lines of though for expanding this work in the future.

I Conceptual Foundations

2

Thesis Context

Urano abre o Mental para uma imediata, intuitiva e global forma de Conhecimento.
As associações mentais passam a ser involuntárias, sintéticas, Universais.
Surgem súbitas, espontâneas, intuídas, como que reveladas.
– Maria Flávia de Monsaraz, *A Onda de Urano*

In order to introduce the theory on how models for data-intensive systems evolve, we need to structure the underlying universe of discourse. *Section 2.1* adopts a systemic view to conceptualize the thesis foundation. *Sections 2.3*, *2.4* and *2.5* incrementally develop the required context to understand the role of data when modeling data-intensive processes. Finally, *section 2.6* consolidates the depicted concepts.

2.1 The Systemic Context

The decomposition of complex systems in manageable subsystems is an important key to understand their operation at arbitrates levels of abstraction. This decomposition increases awareness at the subsystems operation level but offers integration, adaptability and agility challenges to the root system.

Inserted in the context of continually increasing data-intensive landscapes, this thesis adopts the system process perspective to introduce new postulates on the modeling of system elements coordination to minimize these three challenges and foster system evolution. The process perspective sees each system production as the outcome of a set of system reactions to internal and external system actions.

A **system** is a tuple $< R, C, E, G >$, where R is the structure, the set of bond relationships among a composition of system elements C and a set of external elements E, that satisfies a purpose G grounded on exchanges with the system environment [33].
The reader can always specialize the system concept to the *enterprise*[a] notion. This work will preserve the use of the *system* notion as it promotes a simple and focused scope conceptualization and can be easily extended to include other types of systems[b].
The system composition, C, is a set of *subsystems*[c] or, from an elementary perspective, a set of participants P. Participants can either be *passive participants* ($P_P \subset P$) if subjected to transformation during the execution of a set of system activities, or *active participants* or agents ($P_A \subset P$) if performing actions aimed at changing passive participants [17].

[a] a system with *i)* at least two purposeful elements with a common purpose, *ii)* a composition where its parts can respond to each other, and *iii)* at least one element performing the system-control function [6]. In a narrower sense, an organized human undertaking, such as a company, a government agency, a non-profit institution or a temporary project [53]

[b] digital, mechanic or biological systems are possibilities. Systems exchanges can generically be captured as an energetic interplay and the organizational, informational and functional coordination of its elements associated with vibration selection, storage and answering

[c] a system (R_1, C_1, E_1, G_1) is a subsystem of a system (R_2, C_2, E_2, G_2) if and only if $R_1 \subset R_2$, $C_1 \subset C_2$ and $E_1 \subset (E_1 \cup (C_1 - C_2))$ [33]

The modeling of a system operation on the basis of its processes must be understood in broader

systemic context along with the system *ontology*[1] and system *architecture*[2] notions. They are decisive to ensure the system integration, i.e., that the development and operation of a system is coherently and consistently performed [32]. However, they cannot answer to the system improvement needs as they cannot alone assure the system *adaptability* and *agility*[3], i.e., the system ability to translate demands of the system environment into action in a timely manner [53][100].

It is here that the modeling of a system operation centered in processes plays a key role as it promotes the responsiveness needed for these systems to evolve by abstracting, prescribing and auditing the system operation [114][88]. In fact, since process models promote a traceable environment to capture the operation of a system, they leverage the system *self-awareness*[4] and *flexibility*[5], affecting the system ability to *learn* and to *respond* and, thus, fostering an environment for the system evolution [6][121].

2.2 System Evolution

System evolution is the process of increasing the system responsiveness to its environment by continually optimizing the efficiency to pursue its purpose[a] under changing conditions. System evolution essentially depends on the ability of a system to behave as an *integrated*, *adaptable* and *agile* system. Under the functional perspective, the ability of a system to evolve ($S \rightarrow S'$) is determined by the ability to cohenrently, incrementally and timely improve its structure ($R \rightarrow R'$) when internal, external or purposeful changes occur ($\{C, E, G\} \rightarrow \{C, E, G\}'$).

[a] this thesis simplifies the system concept by only targeting open, purposeful and dynamic (or multi-state) systems

Note that since process models are embedded in the dynamic real-world, they are never right as it is stated by the *E-model* perspective [44]. Although it is not possible to model a system that answers all the needs at a certain point in time as each internal or environmental change implies a new model [70], process models can be continuously adjusted to an appropriate level of service based on performance metrics [66] and short feedback cycles [101].

These process-based knowledge increments of a system operation can be captured and shared as a blackboard, where different knowledge sources ($\{\mathsf{modelers}, \mathsf{developers}, \mathsf{executors}\} \subseteq KS \subseteq P_A$) contribute with partial solutions, allowing the required experimentation for a system to evolve [71].

Given this collaborative and emergent nature of process modeling, there is the need to establish a *system governance* competence that, based on the process models trace, coordinate distributed contributions in a way that preserves the system architecture [42]. By providing connections to assets across time and space, process models make it also possible for co-designers to engage in the present by building upon the past in anticipation of a new future [92] alleviating the problems associated with systems memory and "bounded rationality" [11].

Adaptability along with integration and agility determines the ability of a system to evolve. Expressivity[6] (sometimes referred as pre-designed flexibility) and flexibility[7] are at the core of the evolution notion as they define the system adaptability. Aalst and Jablonski [109] suggest a classification of sys-

[1] the understanding of a system in a fully implementation independent way [33]

[2] set of principles, which are essentially regarded as axioms and constraints on the system design and engineering freedom [33]

[3] not related with agile principles for product development – this work only develops theory on the system *modeling* (not operation) level

[4] the sum of each system element's knowledge regarding the whole system operation $\sum_i^{|C|} k(C_i)$ [121]

[5] ability to change without loss of identity [113]

[6] the extent to which a process model captures the system execution constraints

[7] the ability of process models to adapt without loss of identity [86]

tem adaptations based on their: reason, effect and perspectives affected, kind, moment and durability, and set of actions available. Regev et al. [87], although do not clarify the realization options, propose a taxonomy based on the adaptation abstraction level, on its subject and on its properties. Snowdon et al. [102] categorize adaptations according to its causal factors: type (arising from the diversity of information being handled), volume (arising from the amount of information types) and structural (arising from the need to operate in different ways). Along with expressivity, three types of flexibility must be distinguish: *i)* by *deviation*[8], *ii)* by *defer* or underspecification[9], and *iii)* by *change*[10].

2.3 Process Modeling

Functional decomposition of a system defines hierarchies of abstractions needed for the modeling of systems operation [120]. Activities, units of work, are the nodes of such functional decomposition. Since we are addressing open ($E \neq \emptyset$), purposeful and multi-state systems, this work introduces the notion of system behaviour[11] as a set of system events that produces a change to the system state. The source of this change can be external E or internal if caused by a state-change on a system agent P_A, passive participant P_P or goal G. System activities are simply aggregations of system events.

By extracting the functional or activity-based perspective from a system structure R, we have already all the concepts needed to introduce the process notion.

A system **process**, a tuple $< A_\wp, P_\wp, G_\wp, R_\wp >$, is a constrained execution of a set of system activities ($A_\wp \subseteq A$), based on the coordination ($R_\wp \subseteq R$) of a set of system participants ($P_\wp \subseteq P$), to realize a set of system goals ($G_\wp \subseteq G$). Alternatively, a participant-driven sequence of behavior that constitutes a sub-system with a goal-producing function.

A system *model* is the conceptualization, abstracted from details, of an aspect of an object system by a user system [33][28]. System models are used not only to describe, but mainly to prescribe and trace system aspects as its governance, design, engineering or operation. Thus, models must expressively define *specifications*, acting as a blueprint for a set of similar cases, and capture their *instances* that trace the way cases were enacted.

A **process model** is a model for system processes, an abstraction $m(A_\wp, P_\wp, G_\wp, R_\wp)$, which concerns an operational aspect of an object system by an interacting system. Process models *describe* and may *trace* and *prescribe* the operation of a system through process specifications and process instances[a].

[a] *process instance*, $i = (E_i, <_i) \in PI$, based on a *process specification* PS driven from a system structure R, is defined by a partially ordered set of system events E_i such that the ordering $<_i$ either satisfies PS execution constraints [120] or acts as an exception [120]

Process models are described by a *process meta-model*, a complete set of process modeling concepts and associations between those concepts, and expressed according to a specific *notation* [120].

Agostini and De Michelis [8] argue that, independently from the type of system, to support its evolution, very simple process models should be used and exceptions should be dealt with by hand through so-called *linear jumps*. Herrmann [50] seeks a solution by using semi-structured process models. In order to understand how modeling integration, adaptability and agility can be specifically achieved by the

[8] the ability of a process instance to deviate at runtime from its original execution path without altering the specification [99]

[9] the ability to execute an incomplete process model, a process that does not contain sufficient information to be executed to completion [99]

[10] the ability to dynamically modify a process model such that currently executing complying process instances are migrated [99]

[11] either triggered by system reactions (deterministic changes), responses (stimulated changes) or acts (autonomous changes) or simply by system events whose antecedents are of interest. System behaviour, thus, consists of system events whose consequents are of interest

systems targeted by this thesis, we need previously to assess the role of data within process modeling.

2.4 The Role of Data

Different data taxonomies for process modeling can be found in [122][104]. Weske [120] provides a good start to understand the role of data within processes by structuring its aspects according to data visibility, data interaction, data transfer and data-support to routing logic. Aalst [114] distinguishes two main types of data: case and non-case. *Case data* is the data used by system applications to support activities. The non-case data can be divided into *support data*, if it affects the process routing logic, and *management data*, if it is produced by the process execution environment during the enactment of a process instance (e.g. audit trails). Muehlen [122] does the same distinction under a different analysis, but calls to case, support and management data, respectively, content, workflow and workflow-relevant data.

In fact, this thesis simplifies this taxonomy into data generated and consumed by processes [122] and it will focus on the aspects of case data or data consumed by processes. The reason behind this simplification – the increasingly blurry boundary between data exposed for process routing decisions and pure application data – lead us, finally, to the notion of data-intensive system.

In data-intensive system landscapes, the data consumed and generated by some of system elements is related with the production of other system elements [122].

A **data-intensive system** is a system with a structure R that relates its elements C based on data mediation and transformation[a]. Thus, the operation of a data-intensive system is strongly constrained by its passive participants state and relations or, more broadly, by the way the system productions to the environment realize the system goals [6].

[a] the importance and structure degree of signals exchange through vibration emission, storage and response (either gravitational, electromagnetic or quantum) may or not qualify a non-organizational system as data-intensive

For instance, automotive industrial systems rely on the entanglement of data components, the passive participants, to deliver a physical production. Claim-processing systems use claimer's information, regulatory and financial reviews and claim state to audit and deliver a decision. Contrasting, non-data-intensive systems do not particularly depend on data mediation among its elements to affect the way the system delivers a production as, for instance, a front-office or any system resultant from a composition of service-oriented subsystems.

A process within a data-intensive landscape is referred as *data-intensive process*. Scientific research has been focusing on two main types of data-intensive processes: *i)* processes driven by the structure of passive participants [26][3] and *ii)* collaborative and tacit processes that use passive participants as record objects to capture the system operation [7][105].

Since, the research on data-intensive models has been adopting many different directions, we commonly see different terms as *documents*, *products* and *artifacts*. Here all these notions are generalized and captured as objects, either simple or compound (encapsulating a set of related objects). For now, we aside notions as the *case* or *proclet* concepts since they are closer to the system functional view. Objects, either representing logical or physical elements, can be seen as building blocks that bridge the functional and informational perspectives [26][81]. An object model defines a class or object specification that may prescribe the features of its object instances.

A system **object** represents any relevant system element, either a simple participant or a composition of system participants. Its state is defined by the object content and it is modified during its life-cycle as the result of an invocation of a set of activities that act upon its data-attributes.

In the same fashion as we saw with the functional decomposition of a system, the *informational decomposition* of a system defines hierarchies of object models. A *data model* for a system can be identified with the highest level of the object models' hierarchy for that system in a functional independent way.

2.5 Modeling Orientation

Process modeling approaches can be divided according to their main focus of modeling: activity-flow, data-needs or agent-coordination focus lead respectively to *activity-centered* (e.g. [1][119]), *object-centered* (e.g. [26][85][3]) or *agent-centered* (e.g. [72][95][84]) languages. For instance, since agent-centered modeling approaches first concern the order in which active participants (either human or application agents) get and perform their part of work in a process, activities precedences are implicitly derived from agent-interaction constraints. This is the natural choice for processes with strict distribution of responsibilities owned by highly-specialized agents.

Each modeling approach can be characterized by the way execution constraints prescribe the coordination of system participants aiming to increase the system efficiency towards its goals satisfaction. Constraints specification can either be: *imperative* [88][96] or *declarative* [113][3], derived from *roles-assignment* between participants [21][22] or from *actions-assignment* [41][85], derived from *explicit* ordering restrictions [113][96] or from *implicit* (either participant-pushed or goal-driven) coordination [3][41].

Since objects are generically used to model system participants, in particular, simple and compound structures of passive participants in data-intensive landscapes, the execution constraints placed among activities in object-centered approaches is derived from their state and expressive relations.

An **object-centered process model** is a process model that uses the knowledge of the participants structure $P_\wp$ to derive the bond relationships of the system under modeling $R_\wp$. The modeling of participants must be expressive and changes to the system data models must dynamically affect the system process models.

Finally, we can also distinguish *multi-paradigm* approaches where process models do not have a clear orientation [28], and *hybrid* approaches where different approaches co-exist together exposing different views for different users [113].

2.6 Bridging the Concepts

Recovering the system definition, we detect four main perspectives: the *functional*, the *informational*, the *instrumental* (or organizational) and the *contextual*, or, respectively, the activity-based, data-based, agent-based and goal-based views. All the models from each perspective or system view are integrated by a process model, the *governance model*, that establishes relationships between them and constrains their interaction, thus capturing and prescribing the operation of a target system.

Since system modeling is a discipline of trade-offs – flexibility *vs.* traceability, expressivity *vs.* usability – modeling approaches must, depending on the type of system, understand and prioritize the role of each perspective to achieve manageable, suitable and evolutive abstractions.

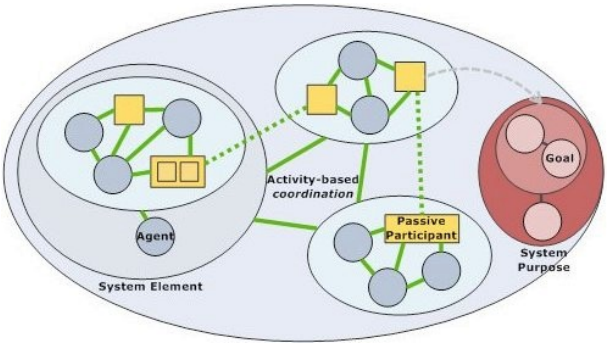


Figure 2.1: Abstract data-intensive system

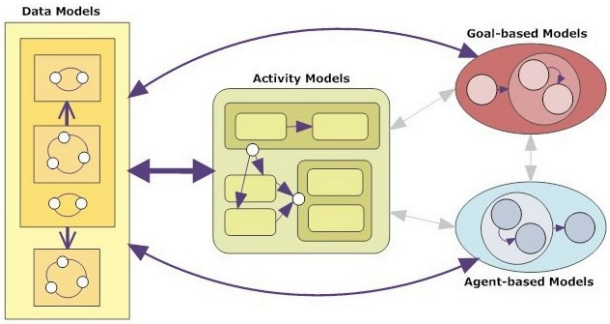


Figure 2.2: Abstract process model

Fig.2.1 depicts an abstract data-intensive system. Fig.2.2 abstracts a structure that separates and relates the models from each system view. Since in data-intensive systems the evolution of process models is triggered by the evolution of object-based models, the relations between systems structure and systems data must be studied in an extended way than the existent contributions, which have been limited real-case applications impact.

3

Thesis Problem

We can't solve problems
by using the same kind of thinking
we used when we created them.
– Albert Einstein

This chapter presents the problem that this research aims to solve. *Section 3.1* introduces the application scenario used throughout this work. *Section 3.2* exploits why traditional approaches are limited in modeling data-intensive processes. Finally, *section 3.3*, uses such knowledge to decompose the problem into a set of requirements.

3.1 The application scenario

To illustrate the problem of this work, we propose a look to the *Global Financing* (GF) case [23]. *GF* is a company specialized in financing hardware, software and IT services. *GF* operates in several countries and aims to execute its operations based on a global standard with disciplined regional variations.

Once a financing opportunity is identified, the customer request is reviewed, negotiating terms and conditions are discussed and possibly agreed through contract signing, involved assets are supplied, purchased and shipped to the client locations and, finally, client payments are tracked until closure. In particular, the review sub-system is multiple times instantiated for every customer request received so several employees are able to analyze it, to possibly interview the customer, to evaluate the request and to take a final decision. If the request is approved, terms and conditions will be defined and negotiated, otherwise a revised version of the commented request is returned to the customer for readjustment and re-submission or for financing withdrawal.

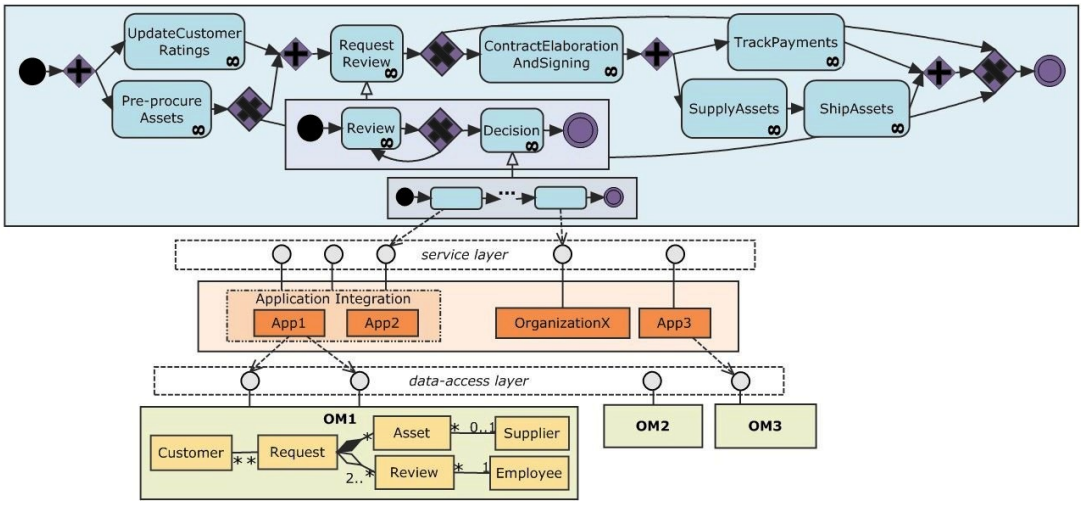


Figure 3.1: Activity-centered process modeling landscape

A simplified and abstract representation of this scenario using a well-known modeling approach is

depicted in Fig.3.1. Although this image introduces already the way process models are linked to a specific setting (here through the instantiation of software agents), this instrumental-tecnhological view was only presented to exploit the true *why* behind the limitations of activity-centered approaches – the reader should not lose the implementation-independent view adopted in this work.

3.2 Limitations of traditional approaches

The problem source of this thesis appears as a consequence of the pushing of passive participants to the processes background that result from an hidden of data behind service-oriented system elements [120][122]. Weske [120] and Muehlen [122] explain in detail the historical reasons for a hiding of business data behind application layers. Only support data, sets of simple data objects needed to affect the process routing logic, are created and maintained by the activity-centered modeling environment [114][122]. Seven reasons are pointed in [98]. However, same reasons block benefits obtained from the coupled evolution coming from a closer intertwine between functional and informational models, i.e.,they do not enable an environment for the integrated management of system data and system activities.

Two main research directions compose the traditional process modeling landscape: the widely-adopted *activity-centered* and the simple *document-based* approaches. They both consider the influence of system participants a sequent aspect over the activities ordering constraints. Let us build upon the limitations resultant from this premise.

3.2.1 Activity-centered Modeling Approaches

Activity-centered approaches model processes by explicitly defining networks of activities. *GF* financial scenario (see Fig.3.1) raises five challenging sets of limitations when this is the chosen approach.

First, activity-centered process models do not want to know how the underlying data is structured (a detailed why is presented in [98]) but only be able to use simple data objects, the support data, when data affects the process routing logic. As a result, support data is redundantly created at the process modeling level, which poses problems of consistency between support and application data.

This missing link between activities and underlying data leads also to a rigid definition of data-access contexts since the scope of data must be explicitly depicted for every activity, usually by specifying data objects as input parameters for each activity (see [119][120]). This aspect leads to non-usable process models when an amount of data must be visible for several sequent activities [64][3], and when ordering decisions are strongly dependent on data constraints [41] as it occurs in data-intensive scenarios. For instance, request data may be maintained during all the Review – an employee may need to access the customer history, credit risk or information regarding the requested assets. Note that even when activities are truly atomic, they may still require access to contextual data, as, for instance, claimed by the previous example.

Second, although it is possible to specify conditions over data objects to constrain the activities occurrence (e.g. as a gateway data-event), activities still have to be related according to a net of ordering dependencies instead of dynamically react on those data conditions. Exemplifying, if Contract Elaboration is dependent on a formula over a set of employees' reviews, it should be immediately executed a soon as this condition is satisfied, independently of the state of each Review process that may not be finished and continue to run in parallel, i.e., activation of an activity does not directly depend on the

completion of other activities. Although the use of data-events together with concurrency patterns (see BPMN [47]) can mimic the result, this solution: *i)* not takes advantage from data dependencies and may contradict them, *ii)* may over-constrain the solution space: limiting flexibility, *iii)* not fosters a modeling towards parallelization, *iv)* assumes that the data is modeled at the process level, and *v)* degrades usability when there are several activities reacting upon data.

Moreover, ordering constraints also difficult the spontaneously repetition of process instances or their stoppage and caught up at a later point in time. For instance, if the payments tracking, depending on the contractual information, can be hold until the asset shipping activity is finished, several paths must have to be incorporated in the process model to avoid the tracking payments path of execution from being blocked. For instance, complex specifications as $OR_{paymentcond}(AND(TrackPayment, SEQ(SupplyAssets, ShipAssets)), SEQ(TrackPayment, SupplyAssets, ShipAssets))$ could be avoided and flexibilized if, somehow, these constraints would come from expressive dependencies among the assets supply, shipping and payment objects.

Third, advanced patterns to coordinate different process instances (either from the same or from different process specifications) are roughly supported by activity-centered approaches, since process instances are executed in isolation to each other. Exemplifying, if we want to separate the assets life-cycle process from their request process, we may want to affect to synchronize progress of the super-ordinate and sub-ordinate processes. Although a few activity-centered approaches try to mimic synchronization patterns [64][96], that is only possible between super and sub-processes – only covers composition relations and not communication among loosely-coupled processes, and processes become blocked until sub-process instances are completed.

Additionally, if one customer starts two requests, both requests will be analyzed in completely ignorance of each other, avoiding possible benefits from their conjunct analysis. Aggregation of activities from different process instances, such as the interviewing or assets supplying activities, is not also possible. Note that multiple-instance support languages, as colored Petri nets [58], only address synchronization for the instances belonging to the same process type, and lack in usability degrading the ability to manage and evolve process models.

Fourth, since activity-centered landscape promote the use of black-box functions provided by the service layer (see Fig.3.1), the attempt to match system activities with those services leads to an absence of a criterion for activities granularity as client application services may reside at different (probably coarse-grained) levels of granularity. For instance, if a customer aims to change its shipping address, fine-grained services to change and read the address must be available. Additionally, there is no criterion to guide the decomposition of the *GF* asset financing process in new functional sub-levels.

Finally, evolution of data and activity models is decoupled. Although it fuzzily seems a good aspect, our initial premise hold the fact that changes at the data level in data-intensive systems implicitly induce changes at their process level. If, for instance, GF begins to accept a single request from multiple customers, several changes are incurred in a typical activity-centered scenario. First, data models should have the ability to migrate since the relationship between Customer and Request is no longer valid. Second, client applications logic to offer the new set of services resulting from this change may also need to be reviewed. And, lastly, the process model needs to evolve in order to reflect these new options. Even the service layer must be continuously adapted to support system evolution when changes have origin in data layers, the most trivial and common way to re-arrange a data-intensive system. Such fact support the need for an integrated modeling environment for the adaptation of a system.

The rigidity leveraged by all these five factors promotes the hard-coded of processes logic within

client applications [64][3]. By going behind the support system's back, process models become more a liability than an asset, leading enterprises to lose control and traceability of its processes and, consequently, turning unviable their continuous improvement.

3.2.2 Traditional Document-based Modeling Approaches

Document-based modeling approaches have their roots in Document Engineering [82][52] and rely on the premise that system activities are centered, constrained and partially grounded in documents [60]. A document is a view for an information set of an organizational system that supports one or more of its processes [60]. The partition of documents in meaningful segments defines work-units, sets of data fields [10]. Documents are also referred as form-units, archival-units or records [10][105] since they capture what it is intended to be processed by an organizational system.

Traditional streams of research [10][60][18] focus on how documents when associated with a process define its content and turn meaningful its routing logic. For instance, a process with a flow dependency between activities A and B uses A to produce a document that is required by B [34]. If process C defines a sharing dependency with A and B, C produce a document that is used by A and B. If A and B have a fit dependency with C, it means that A and B produce collaboratively a document used by C. These dependencies can be extended to a complete support of activity-based control-flow patterns [60].

In fact, this approach can still be considered activity-centered in essence. Although it introduces the notion of documents as a structured way to capture the system operation affected by execution flows, it still has to explicitly define ordering constraints among activities [10][60]. This explains why traditional document-based approaches can be easily reduced to graph-based workflows [119][57].

The limitations of these approaches are, consequently, similar to the activity-centered approaches, with the exception that document-based structures assure the atomicity of activities based on work-units. *First*, the documents view is often totally independent from the system data models, and documents or their work-units still have to be explicitly associated to activities to become visible and accessible in the context of an activity. *Second*, activities still have to be positioned in a network of precedences instead of reacting on constraints possibly defined among work-units. *Third*, since document dependencies derive from activities ordering relations, no sources of expressivity to support advanced synchronization patterns are added over activity-centered models. *Fourth*, documents are plain structures and, therefore, do not offer criteria to guide the functional composition of processes. *Finally*, data modeling at this scope refers to the ability to define and adapt document specification. However document modeling is not expressive as new specifications still need to be manually associated to activities.

Section 5.1.1 discusses how recent document-based directions surpass some of these limitations.

3.2.3 Synthesis of Limitations

A synthesized description of the traditional approaches limitations in supporting the evolution of data-intensive process models is presented in Table 3.1.

3.3 The *FIVE* Requirements

This study led this analysis to a point where the initial problem was break-down into five pieces or specific problems. Table 3.2 structures the five requirements triggered by the introduced limitations of traditional approaches when modeling data-intensive processes.

Area of Concern	Problems	*GF* Challenges (Fig.3.1 upgrades)
Data Access	Process data redundantly created with application data poses consistency problems. Isolated execution of activities cause a loss of process contextual view. Data-access needs to be explicitly specified for every activity, leading to non-usable models. There is no support for an integrated access to old or non-related process data;	Reviews or contract negotiations become: affected by the state of assets procurement (if running in parallel) or other request's reviews, their progress depends on customer's past and similar requests' information, and their contextual data continues accessible without the need to specify all attributes as input parameters for each activity;
Data-state Reaction	Activities have to be related in a net of ordering dependencies, turning difficult a spontaneously repetition of process instances, their stoppage and caught up at a later point in time and a dynamic reaction on data conditions;	Contract negotiation becomes reactively available on a condition satisfaction over reviews' data, independently from executing reviews' progress. Assets and reviews are dynamically instantiated based, respectively, on request and customer's data;
Data-based Coordination	Process instances (from the same or different process types) are executed in isolation to each other, hampering the support for expressive communication patterns among processes;	Decoupling of process segments in a modular way (e.g. Reviews and Assets). Aggregation of related segments: collective reviewal, collective asset supplying and shipping;
Data-based Granularity	Service layers turn impossible the definition of criteria for the processes granularity since client application activities may reside at different granular levels;	Activities for accessing and changing customer or request attributes (e.g. customer address) must be available, and their composition must follow concrete criteria;
Data Modeling	Since data-centered and activity-centered models are separated by an application layer, modeling of data objects is roughly done at the process modeling level.	A request becomes handled by multiple customers (the process modeling environment detects the change of an association multiplicity on an underlying data model).

Table 3.1: Problems of Traditional Modeling Approaches in Data-Intensive Landscapes

Künzle and Reichert [64][63] propose a complementary analysis of data challenges. Other introductory remarks to deepen the understanding of each presented data-awareness requirement can be found in [64][78].

Data Access	Process and application data must be coherently and consistently integrated, meaning that system's activity and data models must be bridged and evolve in a coupled way according to a well-defined set of relations. Process models must avoid data-context tunneling (causing the loss of a broader view on the process) when executing isolated or groups of activities, and additionally must expressively hold data-access contexts from single to multiple activities [3]. Data-scope specification must additionally be usable, seizing benefits of using data models expressivity (e.g. accessing compound or sets of data-objects). Authorized users must access data at any time regardless of the process status [3][63];
Data-state Reaction	Processes must dynamically react on data constraints, turning optional the definition of precedence networks. Since activities are related to objects, they must adapt their behavior (e.g. availability) based on objects' state (horizontal dynamic granularity) [64] and provide a natural method to deduce omission path localization, minimizing sequentiality and, thus, fostering process flexibility;
Data-based Coordination	Processes must use data models constructors to express advanced patterns of synchronization, including: *i)* the aggregation of multiple related instances to reduce execution effort (e.g. grouping related requests – vertical aggregation) [12], *ii)* the definition of asynchronous points of coordination to minimize processes coupling (e.g. synchronize the progress of a set of instances responsible for the assets procurement with their related request) [64], and *iii)* the definition of expressive transition's rule-sets;
Data-based Granularity	Atomicity and composition of activities must be based on the underlying process data [64] to, respectively, safeguard the availability of fine-grained activities and of a criterion to infer compound processes using multiple levels of modeling abstractions;
Data Modeling	There must be possible to model and adapt expressive data models at the process modeling level [64]. Evolution of processes is, thus, fostered by the previous requirements, which assure that execution constraints are dynamically derived from the dependencies of object models in a usable manner, with this one, which enables modeling flexibility.

Table 3.2: Data-related Requirements for Object-centered Approaches

An **object-centered system** is, thus, a data-intensive system with a structure R that satisfies the introduced requirements, i.e., a system where its passive participants are visible to every system agent, dynamically affect activities progress and composition, and are adequately accessed and expressively captured at the process modeling level.

Synthesizing, requirements force the definition of dependencies between process and data models that enable: *i)* a coupled evolution of these types of models and an automatic definition of editable data-access contexts based on data models constructors and agent access levels, *ii)* data-pushed constraints instead of an explicit ordering of activities, *iii)* expressive aggregation and synchronization of process instances, *iv)* guidelines for models' composition and *v)* an integrated environment for their conjunct adaptation. The thesis problem can, thus, be rewritten as the definition of an expressive structure that links the informational and functional perspectives.

This is a relevant problem since the main source of evolution of data-intensive systems begins with the adaptation of informational structures. Also, as it will be exploited, emerging approaches are focused on reduced subsets of those requirements – there is space to develop more mature object-centered modeling approaches. Note, additionally, that this set was developed under the hypothesis that these requirements are not mutually exclusive. Annex 16 exploits the coverage, consistency and coherency of requirements by mapping them in the developed *Process Model Framework*.

4

Thesis Statement

There are two mistakes one can make along the road to truth,
not going all the way,
and not starting.
— Buddha

Hypothesis: ***Object-centered models support the continuous improvement of data-intensive processes***

Chapter 2 conceptualized the universe discourse required for a deep understanding of this hypothesis and *chapter 3* introduced the underlying problem and the motivation for its assessment in the increasingly dynamic and data-pushed environments where systems operate [101][70]. This chapter exploits this statement by presenting the research artifacts, goals and methodology.

4.1 Research Artifacts

The ability to expressively adapt system models determine the system responsiveness and, consequently, its ability to evolve. Since changes to the operation of data-intensive systems mainly concern the way passive participants are structured and produced, the continuous improvement of data-intensive processes highly depends on the ability to dynamically adapt process models when the underlying informational models change.

Support means in this context "to turn possible", to satisfy the requirements triggered by the thesis problem. Therefore, in order to proof that *object-centered process models support the continuous improvement of data-intensive processes* five outcomes must be clearly conceived:

1. a set of **requirements** required to process models evolve in data-intensive landscapes;
2. a **set of principles** based on existing approaches that restricts the solution space;
3. a **meta-model** and a **formalization** that precisely define object-centered models;
4. logical reasoning, acting as a set of **proofs**, on how such models satisfies these requirements;
5. a set of **performance indicators** to measure its practical applicability against alternatives.

These will be the five topics used to approve or reject our thesis statement. Additionally, a **roadmap** exploiting the real-cases where the developed approach can have a positive impact will be presented.

4.2 Research Goals

- to conceptualize the object-centered process modeling universe of discourse;
- to retrieve a set of requirements that compromise the ability data-intensive processes to evolve;
- to define an object-centered structure for the consistent plug of requirements-driven principles;
- to understand and motivate to the potentialities on how to evolve process models based on the adaptation of expressively enriched data models;

- to depict the requirements and implications of the application of object-oriented patterns (as communication, composition and inheritances) in process modeling;
- to define how new soundness criteria can be seized for data-aware modeling approaches;
- to analyze the role of event-driven architectures on the implementation of data-aware process modeling approaches;
- to present how advanced patterns of coordination can be expressively captured at the object level;
- to understand how data-centered criteria can be defined for the default generation of process behavior, access-contexts and encapsulation;
- to detach the applicability of object-centered modeling for different domains in comparison with the available alternatives;
- to systematize the implications of the adoption of the object-centered modeling;

4.3 Research Methodology

The assessment of the thesis statement will neither follow a experimental nor a case-based research but it will be grounded in formal reasoning and theoretical evidence. Nevertheless, a set of metrics applied to three domain-specific scenario will be complementary used to study the practical usability of the proposed approach. The development methodology can be decomposed in the following increments:

1. definition of a consistent and coherent set of requirements that compromise the ability to expressively evolve data-intensive process models;
2. analysis and structuring of how state-of-the-art object-centered modeling approaches answer them;
3. design of principles that process models must comply with to satisfy these requirements;
4. definition of a solution basis that neither neglects nor contradicts the concretion of any principle;
5. derivation of a concrete process modeling approach through logical reasoning;
6. assessment of the correctness and executability of the developed approach;
7. assessment of how the target process modeling answer to data-intensive systems' evolution needs;
8. hypothesis validation by: *i)* proving the object-centered modeling executability and soundness, *ii)* assessing the main implications from its adoption, by *iii)* applying a set of performance metrics to assess its ability to answer to the five requirements in comparison with existent alternatives, and by *iv)* confronting these results with the empirical observations driven from its application to financing, manufacturing and healtchare scenarios.

II Findings

5

State-of-the-Art

Know the Law and Be Free.

Part I introduced the thesis context, problem and statement. It depicted why traditional modeling approaches are limited to evolve data-intensive processes and derived *five* requirements from those limitations. Alternatives to activity-centered ways of modeling exist. *Section 5.1* studies with a moderate detail six mature object-centered approaches according to their ability to answer to the introduced requirements, *section 5.2* presents other emerging directions, and *section 5.3* retrieves a synthesized set of learned lessons.

5.1 The Chosen Object-centered Approaches

Approach		Process constraints driven from...	Belief
Document-based Modeling	[85][5]	documents dependencies	documents shape and track all the operations of data-intensive systems
Artifact-centric Modeling	[41][16][26]	artifacts' state and life-cycle synchronization (restricting activity models invocation)	artifacts' information models and synchronized states fosters data-access and processes' modularity
Business Entities Modeling	[78]		
Product-based Modeling	[90][117]	production components dependencies and quality criteria affecting activity ordering	models for the systems production contain the needed information to affect the process flow
Case Handling	[3][107][120]	data-objects labeled associations and activities precedences	activities of data-intensive cases can be captured and grouped as form-based operations over simple data-objects
Data-driven Coordination	[74][77]	object models' internal transitions and relationship types	dependencies among passive components completely prescribe and support evolution of complex processes
Proclets	[2][112]	interaction of loosely-coupled non-data-container objects' life-cycles	modeling centered on processes communication, instead of ordering emphasis, fosters modeling expressivity
Object-aware Processes	[64][63][65]	objects' life-cycle coordination and reaction upon data-conditions	enriched basis driven from-case handling and centric modeling

Table 5.1: Chosen Object-centered Approaches

The selection of a sample of object-centered modeling approaches to study was based on the practical maturity, data-orientation and diversity of aspects of the existent process modeling approaches. Table 5.1 groups these approaches and clarifies their essential focus.

5.1.1 Advanced Document-based Modeling

More recent document-based research have been applying the document engineering principles earlier discussed to process modeling according to two main streams of research.

The *first stream of research* [85][5] relies on the premise that the documents content and structure determine how multiple processing tasks may be composed dynamically. Put in another way, since documents' work-units have also ordering relations (two work units can have a precedence relation or run in parallel) and conditions, these document constraints implicitly define activities routing. If process dependencies are derived by documents properties, changing a process means changing its documents. Inter-dependencies among work-units (note that work-units may overlap in non-hierarchical ways [5]) and other document properties (e.g. comment regions) are captured in a *policy document* [5]. Thus, the progress of activities is implied by a policy document. Since a set of documents acts as a data model, policy documents of a system must rely on a standardized format [85]. Davenport [30] states that, as document flows defines process flows, returning to a document-oriented view of information means a return to greater simplicity, less detail, and the ability to accommodate less-structured information.

Finally, the *second stream of research* focuses on how documents capture ad-hoc forms of collaboration in knowledge-intensive processes [105][82][54]. These documents, referred as *active documents* [93], foster the tracking of activities based on heterogeneous collaborative applications (e.g. e-mail) since by capturing their routing and tacit comments. They extend simple documents to include custom properties which can contain executable code that can make a document responsive to a great number of situations [37]. We can simplify and think on these active properties as the read and write operations of document fields. The stream of research focused on the collection, distribution, receipt, use and record keeping of documents connected to processes is referred as *Records Management* [93][45][59].

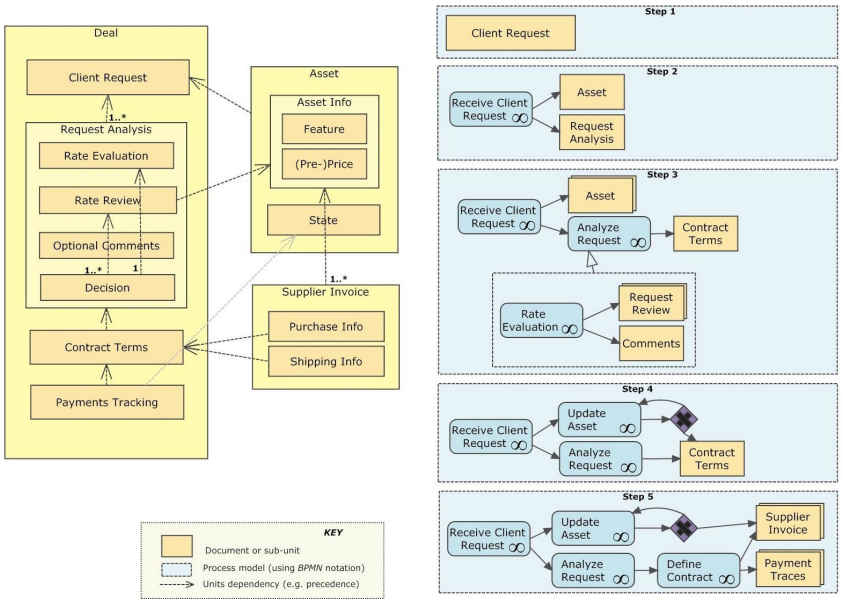


Figure 5.1: Document-based simplified process model

Evaluation: this work adopted the contributions made in [85][5] of the first stream of research to depict the GF application scenario. Fig.5.1 exploits graphically how documents' internal and external dependencies can determine the content of activities that may not be known a priori due to individual peculiarities. This modeling approach does not explicitly define activities content neither precedence, but, as we see on the right side of Fig.5.1 (step 1-5), agents are gradually assisted in selecting the most

appropriate enabled activity and execute them on the basis of documents' fields. Table 5.2 exploits how such approach answers to the previously introduced five requirements.

This scenario is a simplification as it only presents the so-called *functional rules* – documents constraints that support the finding of suitable activities as well as their dynamic enactment. *Organizational* rules that relate documents with system goals based on metrics were not depicted. Using this approach, one could think of models for a data-intensive system as a *system policy* – a set of policy documents prescribing the system operation through functional and organizational rules [85].

Requirement	Support	Evaluation
Data Access	+/−	Two drawbacks are observed. First, system data are constrained to documents' plain structures. Second, each activity still has one unique bounded document or work-unit (the one that supports the user choice among potential activities) and do not recover contextual data by aggregating, for instance, free document-fields. Rahaman et. al [85] introduces, however, the notion of knowledge-base for documents, enabling authorized accesses to documents content;
Data Reaction	+	Conditions on document-fields are possible;
Data-based Coordination	+/−	Note that documents result from the simple aggregation of information chunks. Thus, they are limited in expressing complex data dependencies. Also, it requires document instances number to be defined statically [85]. Nevertheless, dependencies between work-units from the same or different document specifications are possible and dynamically expressed in process models;
Data-based Granularity	−	Atomicity of activities is assured, however functional decomposition of processes is not supported as dependencies among units may cross any hierarchal level of data-fields [85].
Data Modeling	+/−	Changes to document specifications are not possible in run-time and, for some approaches, cannot be targeted by process executors as it requires knowledge of *LTL* formulas [85];

Table 5.2: Evaluation of Advanced Document-based Modeling

5.1.2 Artifact-centric Modeling

The artifact-centric approach[1] was first introduced in [81], which triggered considerable following development efforts [16][41][69]. Like objects, artifacts are either real or conceptual system relevant entities that are created, evolved, and optionally archived as they pass through the system's operation [81]. An artifact type includes both an information model representing the data of a set of system objects during their lifetime, and a life-cycle model, describing the possible ways and timings that tasks can be invoked on these objects [26]. Data values from information models become defined during the artifacts' life-cycle, or in other way, a life-cycle specification describes how an artifact evolve over time as the result of an invocation of a set of services that act upon its data [41]. The set of available activities constrained by artifacts internal state and synchronization dependencies defines a process model.

Artifacts align both process and data views into a unit, serving as the building blocks from which models of a system operation are constructed. Recent derivations [16] capture process goals and allow to track their achievement by dynamically breaking goals in rules that constrain operations over artifacts.

Evaluation: A simplified artifact-centric solution for GF case was developed around three main artifacts (adopted from [26]). Deal artifact, contains the data context and triggers the activities needed to evaluate a client request, to negotiate terms, to sign the contract, to issue invoices and to track payments and their completion. Supplier Invoice is used for the purchase and shipping of the assets to the client locations. And, finally, Asset captures the state of individual assets when accepted from the supplier, titled to GF, delivered to the client, used by the client, and finally sold or disposed.

The Deal artifact, detailed in Fig.5.2, has an information model that starts out empty and over its life

[1] since 2003, IBM Research applied it in different settings [15] resulting in improved understanding and tracking of changes [26][15][69]

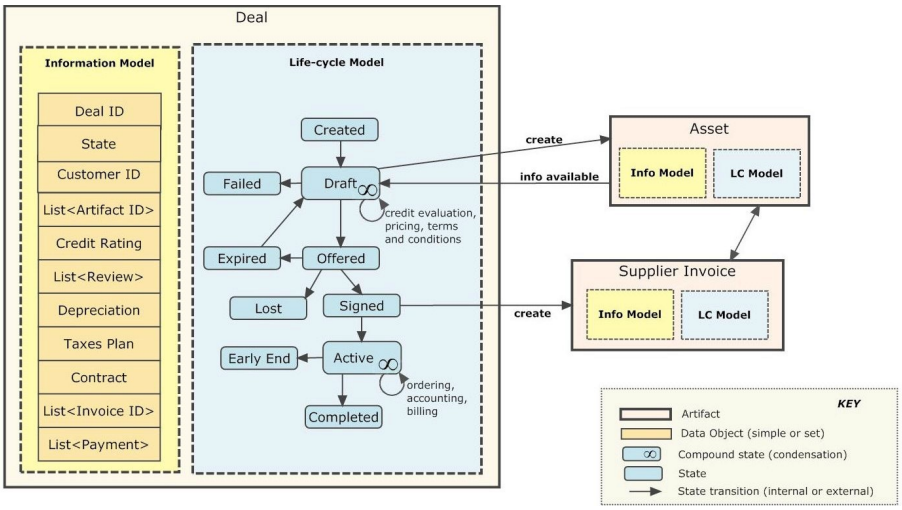


Figure 5.2: Artifact-centric simplified process model

becomes complete and updated, and a life-cycle model that prescribes the artifacts interaction through message passing as they transit between states. Process specification can now be derived from the set of available operations over artifacts' data elements according to their states, i.e., to some *rule-set model* based on data conditions. Table 5.2 exploits how such approach answers to the five requirements.

Requirement	Support	Evaluation
Data Access	−	Although atomic artifacts information model and activities are related, accessing activities availability is restricted by state-based rules instead of authorization-based rules;
Data Reaction	+	Realized by artifacts' internal state (based on information model conditions);
Data-based Coordination	+/−	Dependencies among artifacts are realized through life-cycles' synchronization (see Fig.5.2 to see how state transitions can result in messages that affect the state of other artifacts). These relations, as creation or removal, enable the creation of points of synchronization. However, other advanced patterns are not supported;
Data-based Granularity	+/−	Although the approach enables a natural modularity for capturing a system operation, it does not support the composition of artifacts. An interesting direction would be to use [56] notions, which includes projection and selection on the information model, and a form of condensation of states for the life-cycle model; an analog direction for declarative life-cycles remains open. Atomicity of activities is supported since is driven by artifacts' information model;
Data Modeling	+/−	It does not allow for expressive data-based modeling (as information models only support simple data objects) and dynamic changes was not yet studied in the scope of imperative settings [26].

Table 5.3: Evaluation of Artifact-centric Modeling

5.1.3 Business Entities

Complementary to the artifact-centric direction, IBM is also developing a variant, named Business Entities, with the entity term replacing artifact notion. This approach is centered on how to produce a *Business Operations Model* by bridging the entity definition (using the business entity definition language) with activity-centered processes (using WS-BPEL or BPMN).

Note that the activities definition is not driven by the information model, but being manually defined in a functional basis with explicitly ordering constraints. Such activities have links that enable the access and manipulation of one or several entities, smoothing the modelers ability to think in activity-centered terms but hurting the ability to entities self-evolve by adapting their loosely-coupled interactions.

Evaluation: Fig.5.3 presents an illustration for the GF case, with its analysis being done on Table 5.4.

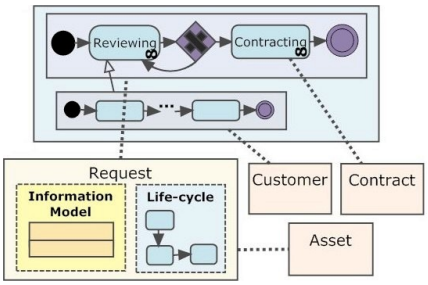


Figure 5.3: Simplified business operations model

Requirement	Support	Evaluation
Data Access	+/−	The aggregation and presentation of data-contexts is handled outside of the process definition using generic service calls based on role and life-cycle state access policies;
Data-based Reaction	+/−	Although a structure based on notifications of state and data change events is used, every BE must be specified in combination with a WS-BPEL or BPMN process that specifies the various processing steps of the accessing BEs. BEs dynamic ability response is limited by the activity-centered precedences introduced by linking layer;
Data Coordination	+/−	Synchronization between BE instances can only be achieved by using processes that manipulate two or more BE instances at the same time. BE life-cycle models do not support parallelism;
Data-based Granularity	−	Intertwining of BEs by WS-BPEL processes with an absence criteria to assure atomicity and uniform composition. No aggregation or composition patterns of BEs is possible;
Data Modeling	+/−	Limited to static-time modeling of BEs with plain information models, whose specification and filling is totally independent from the system data models.

Table 5.4: Evaluation of Business Entities Modeling

5.1.4 Product-based Workflow Support

Product-based Workflow Support (PBWS)is an approach to model data-intensive processes on the basis of their production specification [90]. By centering the modeling and changes on such production a new dynamic and flexible support is possible [117]. It particularly fits models that periodically require a clean-sheet [90]. PBWS is inspired by manufacturing principles, where a close interaction between the design of a product and the process to manufacture is visible [106][118]. Nevertheless, like artifacts, products can be both physical and conceptual. The production specification, the so-called Product Data Model (PDM), has its roots on the manufacturing Bill-of-Material graphs [118], tree-like structures with the end product as root and sub-assemblies, raw materials and purchased products as middle and leaf nodes connected through composition relations.

In general, a PDM consists of a number of data elements linked through operations (or activities) [117]. Each operation has a set of input data elements and produces one output data element. The operation on the input elements, executable when all of them are available and their execution conditions satisfied, can be a calculation, an assessment by a human, or a rule-based decision to determine the output element [116]. Alternative ways to produce the output product are possible if different operations for the same output have different input elements [117]. Finally, operations have a set of quality attributes used for the run-time decision of the most proper next operation to execute [90].

Evaluation: Fig.5.4 presents a simplified PDM for the GF's contract delivery. Although we capture data components dependencies, since they are intertwined by activities there is still an activity-orientation. Note that by adding quality attributes to activities (e.g. $A1$) we not only have process models based on functional behaviour but activities selection strategy also focused on performance. Table 5.5 exploits how PBWS answers to the introduced five requirements.

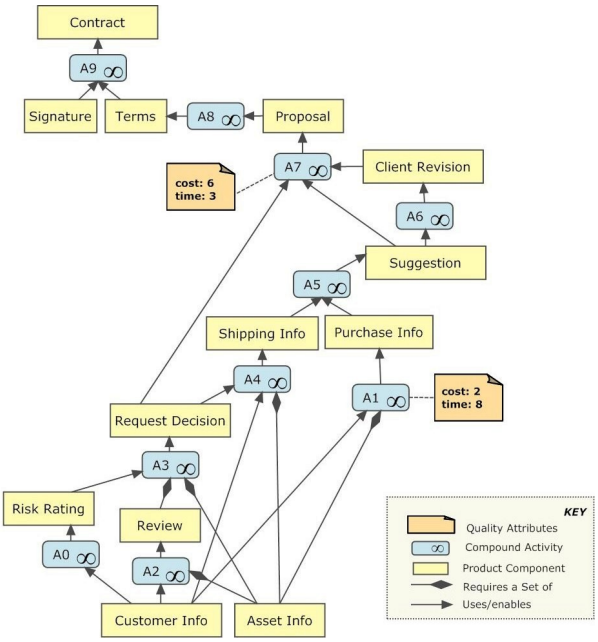


Figure 5.4: Product-based simplified data model

Requirement	Support	Evaluation
Data Access	−	Activities have only access to the explicitly specified input components;
Data-based Reaction	−	Although conditions on data elements are verified by activities, activities do not truly react upon them since they are constrained by an explicit precedence network of operations in a PDM;
Data-based Coordination	−	PBWS applicability depends on how system operations can be described through a product using composition relations between its components, limiting complex data structures and discouraging loosely-coupled entities as they have to be assembled into a single product;
Data-based Granularity	+/−	PBWS foster (but not assures) atomicity of activities based on the input data to produce an intermediate outcome. Although the notion of sub-processes does not exist, granularity can be achieve by by collapsing product model nodes through the use of projections;
Data Modeling	+/−	Only enabled at design time, according to the previously introduced restrictions and intertwined with the modeling of activities.

Table 5.5: Evaluation of Product-based Modeling

5.1.5 Data-driven Modeling

Data-driven approaches are based on the premise that relations between components of a data model indicate dependencies between the activities that modify those components [74]. The specification of data is, thus, possible at the model level and it is used for automated derivation, coordination and maintenance of correspondent processes dependencies that define process model constraints [77]. Although with small variances, this approach follows the principles introduced in the COREPRO framework[2] [75]. Data-driven modeling was triggered by the need to reduce efforts and inconsistencies of modeling processes that consist of numerous concurrently executed interdependent sub-processes as encountered, for instance, in automotive industries or health-care sectors [75]. Such process structures have in common that changes (e.g. adding a dependency between components) and real-world exceptions occur frequently and may affect not only single sub-processes but also the whole process [14]. Such changes are supported as they are derived from the adaptation of data structures.

Efforts have been focused on: *i)* how to describe the processing of single objects (i.e. relations between an object and its modifying processes), *ii)* how to define the processing of the overall data structure (i.e. dependencies between sub-processes in relation to object models), and *iii)* how to automatically derive a proper process structure based on the underlying data structure.

Objects, similarly to artifacts, have a life-cycle (OLC) that define their single processing and that are synchronized with other life-cycles to define the coordination of processes [74] using both internal and external state transitions [77]. An internal state transition is associated with a sub-process modifying the object (thus inducing a state change) and is triggered when the object conditions are met. External state transitions connect states of different OLCs by defining relation types which synchronize related objects. The resultant process model view based on such constraints is the so-called *life-cycle coordination model* which captures this state-based coordination and acts as a blueprint for life-cycle coordination structures which describe a process instance for a particular data structure [74].

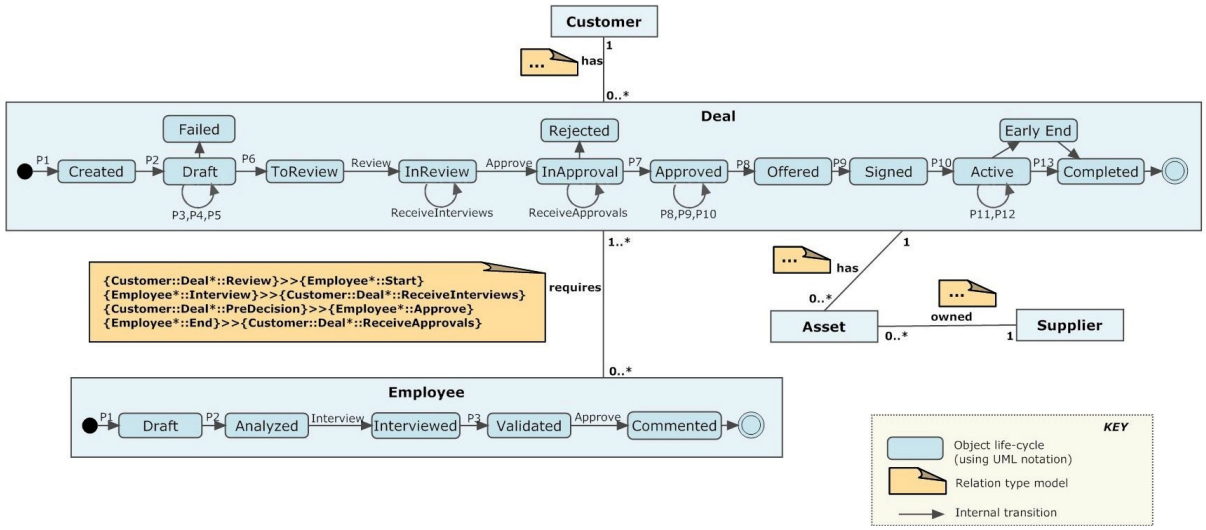


Figure 5.5: Data-driven simplified data model

Evaluation: A simplified data-driven OLC model for the GF operation is depicted in Fig.5.5. Using this approach a monolithic process can be decomposed in loosely-coupled processing objects based on the system data model. If we confront this model with the activity-centered process model in Fig.3.1,

[2]COnfiguration based RElease, a project owned by Daimler AG Group Research

we see that object models are driven by the system underlying participants relation. The potentiality of data-driven relies on its expressivity to synchronize objects on the basis of internal and external state transitions – see, for instance, the requires relation type between Deal and Employee objects. Table 5.6 exploits how such approach answers to the five requirements.

Requirement	Support	Evaluation
Data Access	−	Activities data scope is limited to their OLC data environment (e.g. Deal activities have only access to the Deal data objects) and restricted by OLCs' internal and external state transitions;
Data-based Reaction	−	Since data elements are roughly captured by data-driven process models (only their structure is relevant), data conditions are not supported;
Data-based Coordination	+	Data-driven modeling is the more expressive approach that capture objects relations to define process constraints. A process model can create different types of relations among objects to express a dependency, an aggregation or other patterns for the synchronization of instances belonging to the same or different process types. In Fig.5.5, requires relation between Deal and Employee objects enables the dynamic aggregation of customer's Deals while allows the parallel continuation of the Deal process and the later catch of the triggered Employee Reviews in an asynchronous manner;
Data-based Granularity	+/−	Since COREPRO coordinates processes based on the underlying object-oriented structures, granularities can be freely chosen and be composed according to object-based criteria. Nevertheless, as objects' activities are not driven from data operations, their atomicity is not assured;
Data Modeling	+/−	Although data objects are not specified at the modeling level, data-driven approaches allow dynamic changes to objects structure [74];

Table 5.6: Evaluation of Data-driven Modeling

5.1.6 Case Handling

Case handlingwas first introduced in [107] and posteriorly deepen in [3][9]. Here, the enabling of activities, chunks of work recognized by users, is a mix of data objects conditions and ordering constraints that jointly define the *case* (system's service to its environment) state. By representing in case models the fine-grained data dependencies associated with activities, additional valid executions can be allowed without violating the overall consistency of a process [3]. This offers the flexibility to view and to change data before or after the corresponding activities have been executed.

Providing knowledge with as much information as possible is another important aspect of case handling systems [3]. The focus of case handling is on the whole case, i.e., unlike the previous approaches there is no context tunneling by limiting the scope of data of single process fragments. Thus, all relevant information is presented to the user. Forms are used in this context to simply aggregate and to present different views on the data objects associated to a given case [3]. Forms may also contain data objects that are only mandatory for sequent activities or free data objects associated to the overall case.

Case handling also allows for a separation of authorization and distribution [3]. Access rights take place to adjust the access to data objects. Moreover, the mapping of activities to workers is not limited to the execute role but extended to skip and redo roles.

Evaluation: Fig.5.6 depicts a simplified case model for the GF case. Note that, in order to capture a high-level view of the GF operation, we introduced the notion of sub-case and the notion of compound data-object as an aggregation of data-elements. This image explicit how data is associated to case to activities or to the entire cases through labeled associations realizing the data access requirement. Exemplifying, Interview activity can access and modify the case free data, its optional data fields and mandatory data objects associated to its sequent activities. The process progress or *case state* is defined by *activities state* space and *data state* space, respectively, driven by activities precedences and data

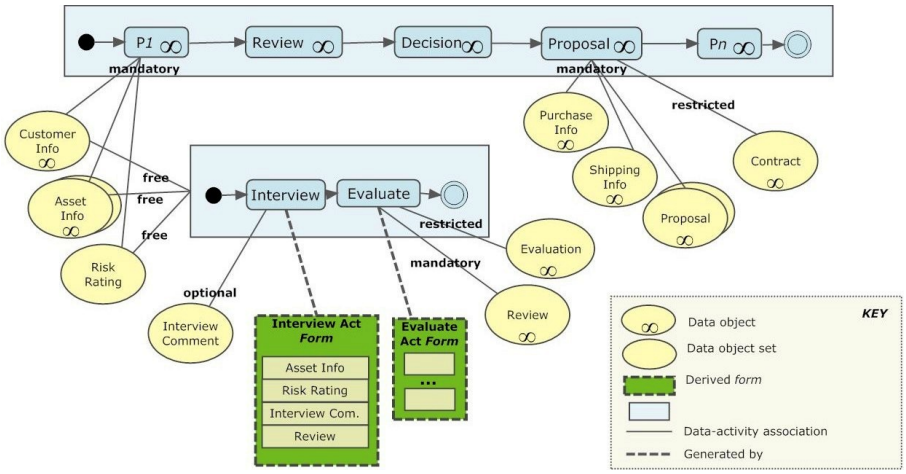


Figure 5.6: Case Handling process model

conditions. Table 5.7 exploits how such approach answers to the five requirements.

Requirement	Support	Evaluation
Data Access	+	Case handling maintains data objects to be read and written within several activities through the use of forms and free, mandatory or optional association labels. Moreover, users can surpass the activities by accessing any data element for which they have privilege access levels;
Data Reaction	+	Although activities precedences exist, activities also react upon data conditions;
Coordination	−	Case handling does not provide support for data objects relation;
Data-based Granularity	−	Although activities are described in terms of atomic data elements, since cases generalization is not considered, hierarchies of processes are rouglhy supported;
Data Modeling	+/−	Dynamic modeling of new data objects is not supported, and new objects still have to be linked to activity models.

Table 5.7: Evaluation of Case Handling

5.1.7 Proclets

Proclets[3] approach is covered in [2][111][115] as an effort to define a modeling approach well suited to environments where processes are fragmented, interaction is important, and tasks are done at different levels of granularity.A proclet can be seen as a lightweight workflow process that interacts with other proclets that may reside at different levels of aggregation [111]. Despite proclets still suffers from data and processes perspectives integration, it is often referred as an object-centered approach as one can think of proclets as objects equipped with an explicit life-cycle or as active documents [37].

Proclets interact via channels, mediums used to transport messages from one proclet to another [115]. Channels support different types of interaction according to the medium type, reliability, security, synchronization, closure and formality [2]. In order to proclets find each other, there is a naming service that keeps track of registered proclets. The messages exchanged among proclets are precisely specified in performatives. A performative is defined by its channel, senders, receivers, type or action and the content exchanged [2]. Proclets are connected to channels via ports. Each port has a cardinality (number of recipients) and a multiplicity (number of performatives exchanged during a proclet life-time) [115].

A proclet defines a class describing the life-cycle and ports for a set of instances. Using these concepts, complex monolithic process definitions can be broken up into smaller interacting proclets. Adding

[3]must not be confused with the worklets notion [46], orthogonal add-in that allows flexibility by the late binding of process fragments

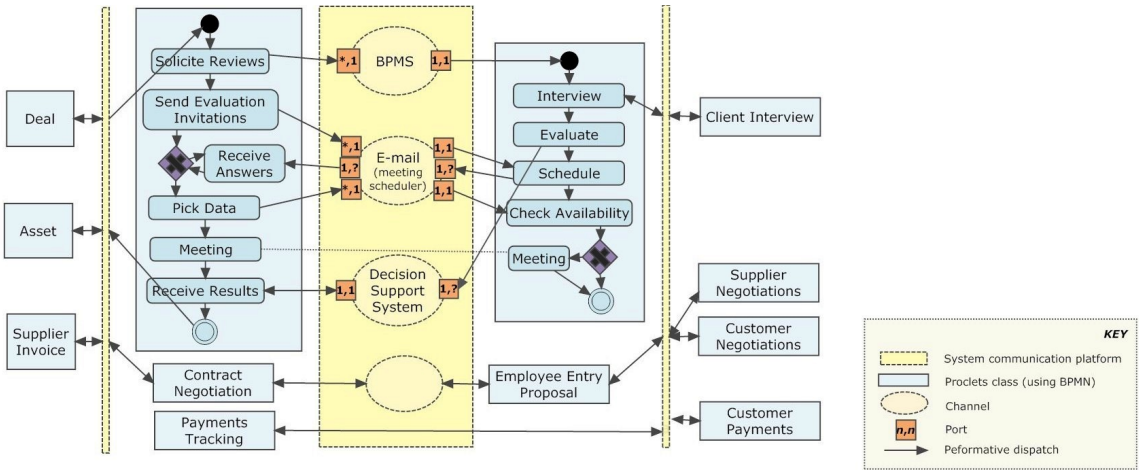


Figure 5.7: Proclets-based simplified process model

proclets to an activity-centered approach promotes a shift from control to communication emphasis [111].

Evaluation: A simplified proclets-based model for GF scenario is depicted in Fig.5.7. The activity-centered entangled process model is, through the use of proclets, divided into simpler objects with an increased emphasis on interaction. Table 5.8 exploits how proclets answer to the five requirements.

Requirement	Support	Evaluation
Data Access	–	Although proclets are decoupled process fragments, each fragment can still be considered an activity-centered model, thus, suffering from same limitations;
Data-based Reaction	–	Activities become available as a result of the work performed (explicit life-cycles where their nodes do not represent states but activities);
Data Coordination	+	The potentiality of this approach resides in the way objects can synchronize with each other through multiple and quality-pushed messages-exchange patterns by recurring to the use of ports and channels. Proclets support *batch-oriented tasks* which enables vertical dynamic aggregation of proclets instances [74]. Although not so expressive as data-driven objects relations in covering advanced synchronization patterns, it provides a basis to integrate system platforms;
Data-based Granularity	–	Although referred as candidates to decompose monolithic processes into decoupled classes [111][115], they neither support atomicity of activities nor proclets' compositions.
Data Modeling	–	Not supported.

Table 5.8: Evaluation of Proclets Modeling

5.1.8 Object-aware Business Processes

The object-aware modeling direction results from a theoretical effort to understand the role of data within unstructured and semi-structured processes driven by user decisions [64]. Although this research stream introduces novel principles, has not yet present concrete constructs to support them [65]. However, its analysis is relevant since object-aware modeling integrates different ideas in an aligned way with the target object-centered modeling[4].

Object-aware process models integrate process and data structures [65]. The *data structure* is composed by object types, their attributes and inter-relations. At runtime object types refer a varying number of inter-related object instances, whereby the number can be restricted by lower and upper bounds. Such number may dynamically evolve depending on the state of the created object instances. *Higher-*

[4]recent object-aware contributions [65][63] were published when the object-centered modeling was already in the implementation stage

level object instances are directly or transitively referenced by a *lower-level* object instances, for instance, a Request is a higher-level instance for a set of Assets, lower-level instances.

The *process structure* assigns behavior to objects using life-cycles to determine in which order and by who object attributes were validly written. Similarly to artifact-centric modeling, a transition may depend on the attributes-value conditions or on the state of its related object instances.

Object-aware research distinguishes form-based activities (used to access or change the objects' attributes) from black-box activities (may integrating advanced functionalities)[5]. Forms progress enable a flexible process execution as change of form-values may lead to the skip or appearance of other form-fields on-the-fly, and also support the re-execution of activities to agents be able to update previously submitted form-fields [65]. Additionally, it integrates the form-view with process-view by by adding form-based activities to the responsible agent's work-list [63].

Form-fields are composed by instance-specific activities, batch activities (collective enactment by using a common form-field) and context-specific activities (collecting related higher and lower-level instances in a form) depending on the number of instances they apply to [65], as depicted in Fig.5.8.

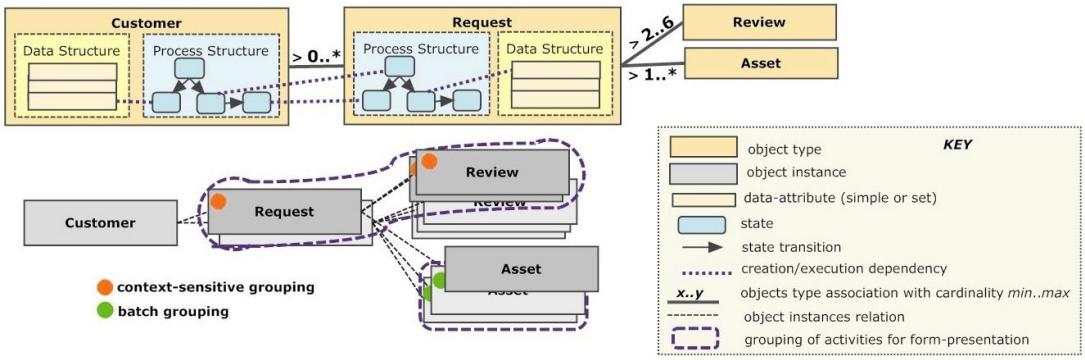


Figure 5.8: Object-aware simplified process model and types of collective enactment of instances

Evaluation: A simplified object-aware process model for GF scenario is depicted in Fig.5.8. Table 5.9 introduces new object-aware aspects by exploiting how it answers to the five requirements. Since this approach was not yet implemented, this evaluation may be optimistic.

5.2 Other Emerging Approaches

Other modeling approaches exist which add important contributions to the reasoning of the target models. Batch-oriented approaches [12] support tasks definition that are based on groupings of small elements to allow simultaneous execution, enabling vertical dynamic aggregation. Multiple instantiation of activities based on simple data structures, such as sets and lists [110][49], is supported by UML 2.0 activity diagrams (Expansion Region) [83] and by BPMN (Multiple Instances) [47]. They enable iterated or concurrent execution of the same activity for each element given by a flat data container.

The Object-Process Methodology (OPM) [36] is an object-centered approach focused on connecting object states and processes through procedural links. Similarly to data-driven modeling, Team Automata [38] provides a formal method to describe the connection of labeled transition systems (automata) via external actions associated with internal transitions based on events.

[5]this work does not make this distinction as: *i)* either state-changes on form-based or black-box activities correspond to a normalized event that can be directly triggered by a software agent, and *ii)* it assumes that atomic black-box activities are as well captured by a data-attribute

Requirement	Support	Evaluation
Data Access	+	Integrates case-handling principles as the inclusion of authorization or the separation of agent authorization and allocation. It adds the concept of *vertical authorization* for the selection of potential actors not only be dependent on the activity itself, but also on the object instance processed by it [65]. Complementary to this instance-based authorization, process- or state-based authorization to access and modify objects' attributes must be available [63];
Data-based Reaction	+	Process progress depends on objects' life-cycle progress which is responsive on data-conditions and state-changes of the related object instances. Execution is flexible as forms dynamically change and their fields-filling order is decided by their enacting agents instead of automatically chosen based on historic data [63];
Data Coordination	+	Object concurrent interactions are supported by three types of state-dependencies: *i) creation* (when the creation of an object instance is dependent on the state of the related higher-level object instance), *ii) aggregation* (when information from its lower-level object instances should be accessible), and *iii) execution* (when an object instance to transit between states depends on the state of another object instance) [65]. However, advanced patterns, as the definition of a synchronization point for all the running *Assets* from a particular *Customer*, are not supported;
Data-based Granularity	+/−	Atomicity is assured, composition is referred as a requirement but not yet presented how modeled in practice and their fit with existing object type relations and grouping of activities;
Data Modeling	+/−	Data is modeled at the process modeling level, but in a redundant way with systems' application data [65]. It does not support expressive patterns, as observed in data-driven modeling. The adaptability of data models is not referred.

Table 5.9: Evaluation of Object-aware Modeling

Support for the product specification, based on how data should be clustered to form the key objects, is an emerging issue [69]. Also, specification of temporal constraints is an important direction [48]. Finally, Operation Specification (OpS) [81] define ways to describe operational semantics within objects.

5.3 Discussion

Results were collapsed in Table 5.10. We say that an approach *answers* a requirement when satisfies almost of its clauses, *partially answers* a requirement when satisfies at least one of its clauses, and *not answers* a requirement if does not approach it. A brief analysis done in Table 5.11 supports the presented results.

Approach	1.Data Access	2.Data-state Reaction	3.Data-based Synchronization	4.Data-based Granularity	5.Data Modeling
Document-based Modeling	+/–	+	+/–	–	+/–
Artifact-centric Modeling	–	+	+/–	+/–	+/–
Business Entities	+/–	+/–	+/–	–	+/–
Product-based Modeling	–	–	–	+/–	+/–
Data-driven Coordination	–	–	+	+/–	+/–
Case Handling	+	+	–	–	+/–
Proclets	–	–	+	–	–
Object-aware Modeling assuming a viable implementation	+	+	+	+/–	+/–

Table 5.10: Approaches Evaluation [subtitles: + answers; +/– partially answers; – not answers]

Discussion reveals that each approach answers differently to the introduced requirements. Neither approach satisfies all of them, which may justify their limited real-case applications coverage and impact. However, lessons can be used to retrieve principles to derive a more mature modeling approach.

Document-based Modeling	It fosters simplicity by modeling constraints recurring to data-dependencies, guides enactment, supports authorized data-access using knowledge-bases and captures ad-hoc forms of collaboration. However, it limits data modeling to plain structures, does not support advanced relations among documents, and instances are defined statically;
Artifact-centric Modeling	It is oriented to business needs and execution constraints are automatically driven from artifacts modeling. However, activity data-access is limited to the related artifact, life-cycles synchronization only supports few patterns and, finally, composition of artifacts is not possible;
Business Entities	This artifact-centric variant offers additional access policies based on role and life-cycle states, although it hurts flexibility by limiting the ability of entities interact in a loosely-coupled way and, consequently, their adaptation, since activities are manually defined and explicitly constrained;
Product-based Modeling	It is good for process models that periodically require a clean-sheet, uses quality attributes for dynamic path choice. However, data-access is restricted to operations' input components, there must exist an explicit precedence network of operations, and applicability relies on the ability to specify productions using composition relationships that can be assembled into a single product;
Data-driven Coordination	It is indicated for large and numerous concurrently executing processes. It adds advanced communication patterns, as the definition of synchronization points among instances belonging to the same or different process type, based on objects relationship types, and allows for complex structures specification that can guide functional decomposition. However it disregards simplicity, data content that leads to data reaction and access problems, and atomicity of activities;
Case Handling	It is unique in providing a global view of the process to its users, data-access is expressive and users can surpass the activities by accessing data for which they have access levels. It allows for horizontal aggregation and, since it is fully state-based, it is easy to conceptualize. However, processes hierarchies and data object-oriented patterns for synchronization purposes are poorly exploited;
Proclets	It promotes a shift from control to communication emphasis, where processes interact according to an agreed level of reliability, security, closure and formality. It supports multiple messages-exchange patterns and batch-oriented tasks, thus, enabling vertical dynamic aggregation of proclets instances. Although proclets are decoupled process fragments, composition is not supported and each fragment can still be considered an activity-centered model, thus, suffering from same data-access and data-state reaction limitations.
Object-aware Modeling	It integrates case handling's data-access, enriching authorization with state-based and vertical (instance-based) levels, and form principles. It supports artifact-centric patterns of coordination. It introduces collective enactment of batch activities, on-the-fly form changes, multiple instantiation using cardinalities, re-execution of submitted activities and offers a natural integration with process-view by assigning mandatory activities to work-lists. However, it does not support vertical aggregation for instances with different higher-level object instances, sound criteria and object-oriented modeling patterns as inheritance and composition.

Table 5.11: Brief Review of some of the Emergent Approaches

III

Solution Architecture

6

Solution Basis

I keep the subject constantly before me
and wait till the first dawnings open
little by little into the full light
— Isaac Newton

This chapter uses the understanding of how emergent approaches answer data requirements to structure a solution basis for the definition of principles. *Section 6.1* introduces a set of object-centered system aspects for the development of an object-centered meta-model, and *section 6.2* identifies the main steps for the modeling of object-centered systems. Relying on this structure, a set of principles are retrieved in *section 6.3* and *section 6.4* will use this set to revisit and clarify the thesis contribution.

6.1 System as Synchronized Set of State-based Entities

A process is here captured as a set of state-based and synchronized entities – objects, activities, goals and time. Exemplifying, a transition between states of an entity A can trigger an event for another transition occurrence in an entity B if B has some sort of dependency with A.

Two key *axioms* must now be introduced. ***First***, process state is a function of time and of its activity, object, and goal models. ***Second***, synchronization among entities is defined through event-driven state transitions recurring to rule-set models.

Object models and *activity models* in the modeling landscape of data-intensive systems are integrated by a *process model* (the governance model) that establishes relationships and constrains their interaction through *rule-set models*. Although the main focus of object-centered approaches is placed on how these models relate, *goal-based models* may also be key governed models.

Relations among these object-centered models will be deepened and formalized through this work. Introduced problems require objects to constitute a basis to derive activities' constraints, context access, coordination and composition. In particular, agents must compose all the behavior needed to support activities. Activities progress must affect objects state, possibly creating, changing or removing related object instances. Rule-sets place constraints within and among object and activity models. Rule-set models must, additionally, bind goals to an object, as system goals satisfaction dynamically derive from its employees or productions state, and, complementary, use goal's formulations to dynamically derive objects' constraints pre-assuring the system ability to deliver its productions [41][20].

Note that the notion of system applications that traditionally intertwined data and process models are in data-intensive landscapes pushed back and captured as workflow systems' additions to implement non-trivial system rules.

A meta-model for an *activity-centered* process model defines: *i)* a set of concepts, the nodes N, that comprise activity models AM, event models EM that turn explicit the process state, and gateway models GM that define a set of control-flow constructs C; and *ii)* a set of relationships between these concepts,

the edges E. Thus, when we want to integrate the modeling of participants, in particular the passive participants, no longer the focus on this formalization[1] suffices.

An **object-centered process model**, $<OM, AM, GM, RM>$, is a model derived from a set of state-based object (or participant) models OM, related activity models AM and goal models GM. These governed models are synchronized through rule-set models RM that constraint the availability to invoke activities based on state restrictions or concessions to the governed models. It can generically be defined as a pair $<N, E>$ where $N = N_{OM} \cup N_{AM} \cup N_{GM} \cup N_{RM}$, and $E \subseteq N \times N$.

Note that this definition broads the initial activity-centered perspective by integrating the informational view and generalizes the notions of event and gateway models into rule-set models. Note also that every activity is related with an object – functional and informational structures are similar, although their content and goals differ. In this sense, in the absence of additional rule-set models, activity instances become available (*initialized→available*) when the corresponding object instance is created. The creation of object instances arise from system agents – either human (*conversation* object is instantiated and manually filled when a physical-mail is received) or artificial (*conversation* object is instantiated and automatically feed when an e-mail is received, assuming that the e-mail platform is integrated with the process execution environment through an agent able to trace collaborative information flows[2]).

A simple scenario is now introduced to clarify the introduced axioms – a system composed by A and B simple objects, by a third compound object C (that associates a B instance with two or more A instances), by their related activities (respectively, X, Y and Z), and by a goal K. The state of the process model derived for this system is defined as the external product among the object, activity and goal models state[3]. Rule-set models are by default (although editable) assigned to every state-transition, for instance, rule γ_1 triggers the $B_{S1} \rightarrow B_{S2}$ state transition when its condition (at least two A object instances related with the target B instance are in the A_{S2} state) is satisfied.

Presently, this system has the A^I and A^{II} object instances related with an object instance B^I, and X^I, X^{II} and Y^I activity instances acting, respectively, upon A^I, A^{II} and B^I data. As activities X^I and X^{II}, in *execution* state, change A^I and A^{II} data, they cause a change of their states from A_{S1} to A_{S2}. When both object instances reach the A_{S2} state, γ_1 is satisfied and B^I object instance transits from B_{S1} to B_{S2} state, causing activity instance Y^I to transit from the *initialized* to *available* state. An agent enacts Y^I, triggering its transition to the *execution* state, and by changing the data of related B^I object instance (that transits from B_{S2} to B_{S3} state) triggers the transition of Y^I to the *succeed* state. Since object instance B^I is in the B_{S3} state, process model's goal instance K^I transits to the satisfied state.

The notion of a data-intensive system as a synchronized set of state-based entities is event-driven by nature. Entities dynamically publish and subscribe sets of events related to data and marking-changes. Annex 15 explain how an event-driven architecture may fit with this paradigm, and its implications on modeling traceability, correctness and usability.

A simplified meta-model for the object-centered modeling landscape, capture its key concepts and their associations is depicted in Fig.6.1. The distinctive value for this approach resides on the behavourial perspective, which not presents complex control-flow patterns, but expressively constrains the systems' operation freedom from both functional and informational perspectives. Note also that

[1] $(N, E, type)$, where $N = N_{AM} \cup N_{EM} \cup N_{GM}$, $E \subseteq NxN$ and $type : N_{GM} \mapsto C$

[2] this field of research is reffered as *Human Process Management* and is centered on governing ad-hoc, human-centric, unstructured processes by integrating office, e-mail and document-based applications while providing end-to-end real-time visibility and complete audit trail [7]

[3] $C^v_{state} \times Z^w_{state} \times K^u_{state}$, with $C^v_{state} = f(A^q_{state}, B^r_{state})$, $Z^w_{state} = f(X^s_{state}, Y^t_{state})$, and $< q, r, s, t, u, v > \in \mathbb{N}$

contextual and instrumental perspectives (based on [68][28][120][101]) are present although not primarly influencing the behaviourial perspective of process models.

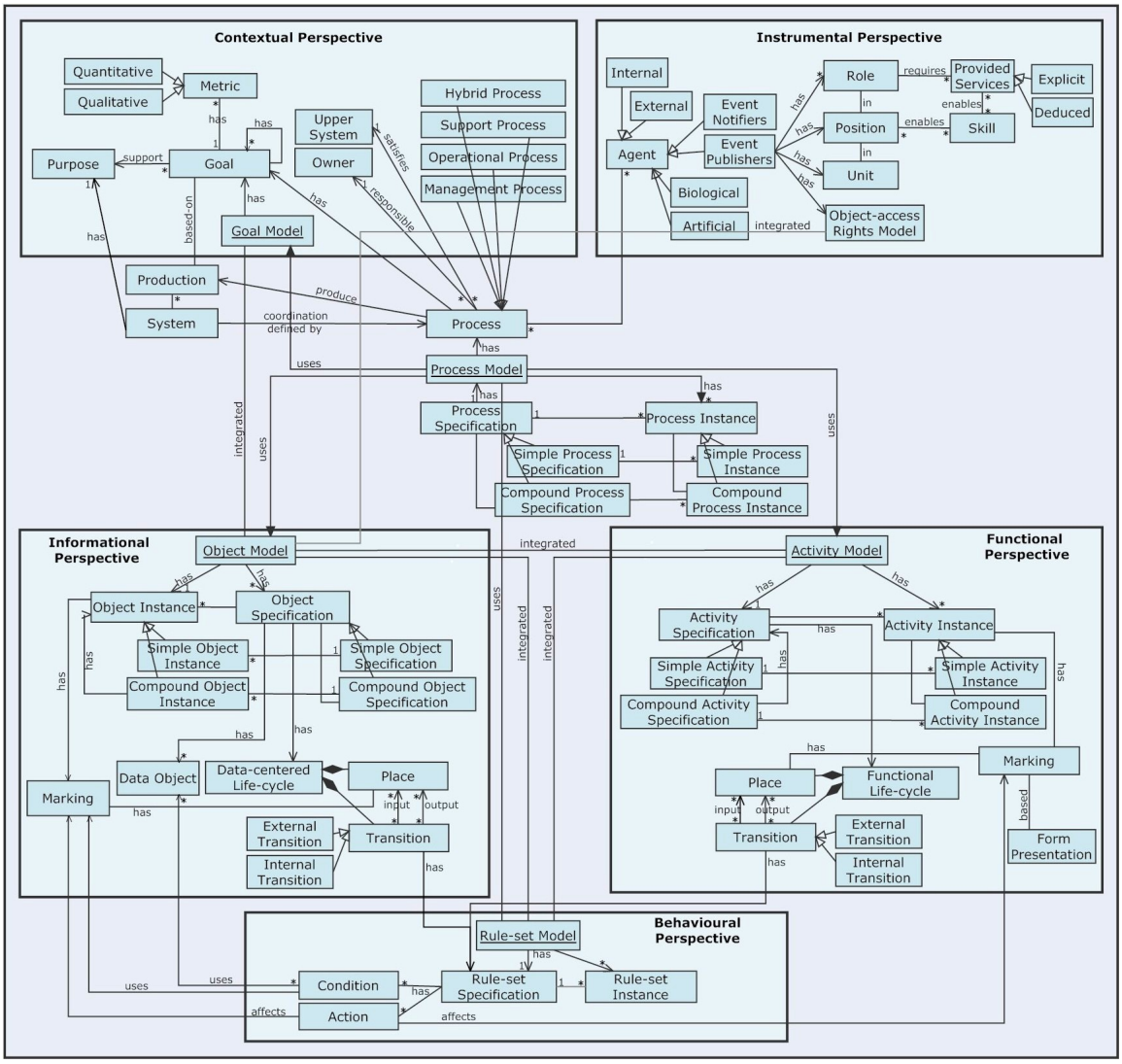


Figure 6.1: Object-centered Processes Initial Meta-Model

6.2 The *Process* of Modeling Data-intensive Processes

With the understanding of the main object-centered modeling concepts and relations, it becomes crucial to distinguish the main steps required to model an object-centered process. The process of modeling the target process models begins with the objects' specification by enriching the system data models with synchronized life-cycles and data dependencies. Rule-set models are used to specify advanced behaviour. Activity models, partially derived from object models, contain editable functional aspects and data-scope. Process models are dynamically derived from the previously defined models.

Object-centered modeling steps are synthesized by the following algorithm:

The object-centered modeling starts with the objects' specification, i.e., with the definition (*step 1*) and enrichment (*step 2*) of system data models by defining synchronized life-cycles and data dependencies. Based on the direction and multiplicity of the transitions that synchronize these life-cycles, rule-set models are automatically derived. Rule-set models can, however, be edited to specify advanced

Algoritm 6.1: The *Process* of Modeling Data-intensive Processes

1. (Manual) Definition of a data model for the target system using UML;
2. (Automatic) Generation of the object models skeleton;
3. (Manual) Edition of object models to capture the system semantics;
4. (Automatic) Test of object models' soundness. Generation of activity models;
5. (Manual) Activity models' constraints and data-scope edition;
6. (Automatic) Test of activity models' soundness. Derivation of the object-centered process models net. Generation of default rule-set models;

foreach *true* **do**

7. (Manual) Adaptation of one of the system object-centered models;
8. (Automatic) Test of models' soundness. Change of the affected models;

behaviour (*step 3*). Activity models are, then, derived (*step 4*) and enriched with additional functional data-access aspects (*step 5*). Finally, a sound process model is derived from the previously introduced object-centered models (*step 6*). Informational and functional views are, thus, coherently bridged and avoid the use of non-source-traceable events that limit sound criteria expressivity.

Note that these steps are only required for a clean-sheet modeling of systems. This work is mainly focused on *steps 7* and *8*, that enable the organic evolution of the target process model through real-time adaptations of both activity models and object models. Fig.6.2 graphically presents the process of dynamically derive an object-centered process model, progressively concretized through this work.

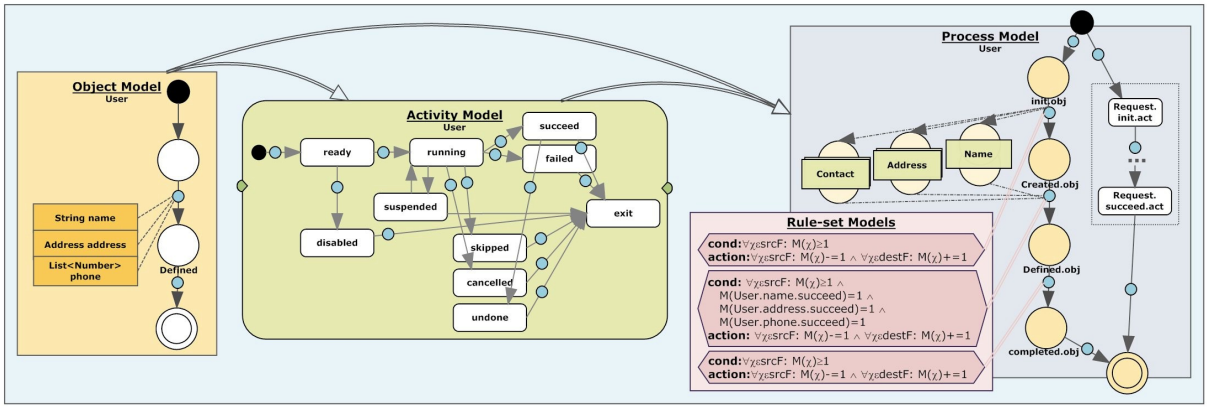


Figure 6.2: Basic Transformations on the Process of Modeling Object-centered Processes

6.3 Solution Principles

Using the knowledge of the previously studied approaches, this section retrieves a set of principles, solution space constraints, that foster the satisfaction of the introduced requirements. In order to assure the consistency and coherency of these principles (i.e., to avoid the non-coverage of requirements and mutually exclusive principles), they rely on the solution basis developed in the previous section. The principles are synthesized in Table 6.1. Following chapters will exploit in detail *how* the target object-centered approach supports each principles.

Data Access	***First***, all system data is captured (by modeling either tacit or non-tacit applications and using their trace to feed their objects), standardized (by using a widely-accepted data modeling notation) and accessible (by not hiding data behind applications). ***Second***, activity models prescribe objects' data access by specifying labeled associations, both imperative and declarative, at any granular level of activities and of objects. ***Third***, related running activities are presented together using forms. Forms fields change dynamically since aggregate activity-related attributes can be alternately submitted and may turn new attributes available (soundness criteria must answer to form's deadlocks). ***Fourth***, there is a default criteria for the automatic definition of the activities' data scope based on the object models. An activity has not only access to its related object or attribute but may access related/internal/super objects and attributes if these associations declare *public* visibility. ***Fifth***, authorization is separated from distribution using an object-based structure to manage agents' privilege access levels. Vertical (instance-based) and state-based authorization also define access levels.
Data-state Reaction	***First***, the object-centered models' interplay assures that activities react on objects state in a traceable manner. Even activity's completion, failure or cancellation behaviour is, by omission (although editable), retrieved from related object specification. ***Second***, it is possible to specify dependencies of different types (e.g. start-to-start, finish-to-start, start-to-finish, finish-to-finish) among data-attributes or any granular level of objects, which generates expressive dependencies on the functional level and fosters an usable parallelization of attributed-based activities and the ability of process models to evolve through object models adaptation.
Data-based Coordination	***First***, object state transitions may depend and affect other object instances' markings. Communication among objects, always mediated by a third object, is derived from object models to process models and it enables the definition of asynchronous points of synchronization between processes. ***Second***, the skeleton for the rule-set models is automatically generated on the basis of their input and output states and of expressive constructors. ***Third***, rule-set models placed in objects' state transitions can comprise advanced formulas based on aggregation constructors, data-scope settings, time conditions and executable code additions.
Data-based Granularity	***First***, each compound activity is a composition of fine-grained activities and for each data-field there is an activity that triggers system acts to access and modify its content. ***Second***, object model's relations of encapsulation constitute a criterion to derive the processes' composition, enabling zoom operations through different operational levels either by hiding internal life-cycles or by collapsing them into compound states.
Data Modeling	Definition and continuously adaptation of data-intensive system is possible through dynamical creation, edition and removal of objects at the process modeling level by assuring that the derivation of processes from data-structures is performed on-the-fly. Soundness is verified before each modeling change and a coherent migration of the affected object-centered instances applied when changes are sound.

Table 6.1: Principles for the FIVE Requirements

6.4 Principles and the Thesis Contribution

Before assessing the degree of novelty proposed by these sets of principles, four key issues must be kept in mind. First, this thesis is centered on the definition of an event-driven solution for the solid plug of principle, i.e. works on a higher level of abstraction providing useful constructors to support principles and adaptability for the integration of new ones. Second, many of the presented principles are novel. Table 6.2 identifies the source of contributions of the introduced principles. Third, studied approaches are only focused on some principles, turning hard the integration of other as their architectures do not respect the previously depicted axioms. Fourth, recently directions, although provide a good principles coverage degree, still lack to support key principles [78][65].

Requirement	Principle	Novel	Source	Singularity
Data Access	1. Data Integration	partial	[64][7]	Modeling landscape forbids hiding of data behind agents. Ad-hoc and collaborative interactions are modeled and fed through HPMS;
	2. Data Contexts	no	[3]	–
	3. Whole View	partial	[65][107]	Criteria for adaptive forms: collective enactment and on-the-fly effects of submit, skip, cancel, suspend and re-execut activities;
	4. Default Access	yes	–	Data-centered criteria to generate data-contexts;
	5. Authorization	no	[63][3]	–
Data-state Reaction	1. Cross-model Affectance	no	[81][74]	–
	2. Data Dependencies	no	[85]	–
Data-based Coordination	1. Process Communication	partial	[115][64][77]	Traceability of the event-based interactions. Encapsulation of communicating processes;
	2. Default Behavior	yes	–	Generation of expressive and editable formal rules grounded on set theory;
	3. Advanced Behavior	partial	[12][37]	Joining of time-constraints, active records and batch-orientation;
Data-based Granularity	1. Atomicity	no	[89]	–
	2. Composition	partial	[89]	Data-centered criteria to compose processes;
Data Modeling	1. Coupled Adaptation	partial	[77]	Coupled evolution of data-models (e.g. change of relations multiplicity) and activity models. Dynamic adaptation.

Table 6.2: Source and Novelty of the proposed Principles

Additionally, the present object-centered investigation is unique in presenting a direction for an approach that: *i)* fully derives and adapts process models from enriched data models ([78] still requires the manual definition of activities for the manipulation of participants), *ii)* defines advanced behavior based on formal rules, *iii)* uses objects data-visibility to formulate new soundness criteria, and *iv)* exploits object-oriented inheritance and encapsulation patterns for an expressive derivation of processes.

7

Solution Derivation

Your work is to discover your world
and then with all your heart give yourself to it.
– Buddha

Through this work we conduct an analysis centered on the problems of modeling responsive data-intensive systems – a set of requirements were defined, principles that answer them were retrieved based on the analysis of emergent approaches, and an event-driven solution basis for the object-centered process modeling were defined. As emerging are centered on a specific sub-set of the overall data-centered process modeling challenges, an extended approach need to be targeted. This chapter, guided by the principles support, concretes such approach. This concretion is here done incrementally by adding features to the developed solution basis.

Before move on, it is important to recover the object-centered three types of governed models required to derive complete and sound process models. First, *object models* specify sets of data-elements, places and transitions that describe the system productions progress traced by marking function. Second, *activity models* specify a normalized (although editable) state-machine, partially derived from related object models and centered on behavioral aspects. Third, *rule-set models*, condition-action pairs associated to each transition of any object-centered model. Conditions are formulas based on the transitions domain (or input places), whose satisfaction triggers an action affecting the marking of the transitions image (or output places). The synergies with a fourth type of governed model, the *goal models*, are out of the thesis scope.

7.1 Data Access

Data Integration. How to use and affect system data if it is hidden behind applications or even lost among collaborative platforms? When modeling a data-intensive system, expressivity arising from its data models must be seized. This statement has one main implication[1]: data models must be visible to the process modeling environment, as illustrated in Fig.7.1.

This roughly seems to hurt the independence among the informational and functional architectures. However, the introduced principle creates an environment where process models are dynamically adapted when changes occur at the data level. Additionally, the unhiding of data from applications may also bring challenges. However, applications, as already referred, are seen in data-intensive landscapes as loosely coupled data-processing additions to the process modeling environment [25]. As exploited in Annex 15, this improve process modeling flexibility since human and application agents can be freely added, removed or upgraded as they only interact with system modeling landscape via events (either by manual filling of forms or through by attaching normalized adaptors [73]).

[1] another implication, out of this thesis scope, is the capture of tacit knowledge. Possible solutions can be leveraged on recent *Active Record* [93][45] directions, where records are used to capture tacit communication, and on the ideas of *ActionBase* [7], where email-based and collaborative-work-based tools trace the information flows of system agent-empowered platforms.

Fig.7.1 captures in a simple this new modeling environment. Note that, since all information flows occurred in a data-intensive system must be captured, tacit interaction through collaborative platforms must be modeled. For instance, a *conversation* through an e-mail application is an object as a *customer* or any other object, with a life-cycle, data-attributes and internal objects, the entries, and external transitions that, by triggering events, may affect other objects (e.g. creation of a customer *request*). The artificial agent or application responsible for the filling of the respective forms must be built upon existing communication platforms for real-time tracking to be possible [7].

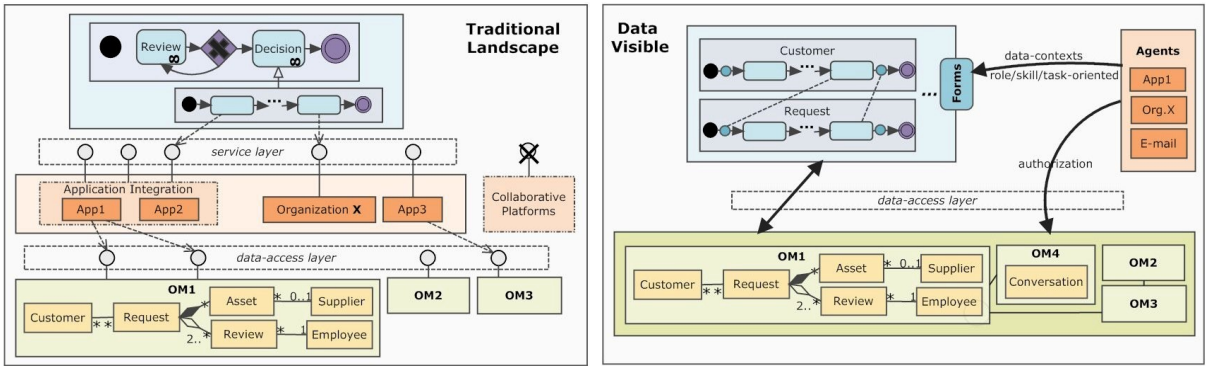


Figure 7.1: The new landscape: integrating data in the process modeling landscape

Data Contexts. How to specify access to contextual data in an usable way? Every simple activity model is related with the read or modification of a data-value. Compound activities may relate with the access of a set of data-values, possibly empty as its internal activities are neither available nor running. However, most of the changing activities (e.g. *risk rating*) may require access to contextual data without the need to specify every data-element as an input parameter for every activity. Thus, our approach needs to provide associations between object models and activity models.

First, note that associations can either be imperative (e.g. activity A have access to data object D) or declarative (e.g. object D is accessed by all activities except by A), depending on the system profile. *Second*, note that associations can be made with any granular level of objects (assuming access to each internal object) and activities (which automatically affects the data-contexts of each internal activity). Compound activities can be always created to group a set of data-context related activities. These properties simplify the modeling task and avoid the complexity of *data-context propagation*. Fig.7.2 illustrates how activities access data for reading purposes.

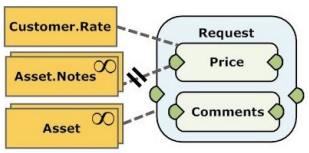


Figure 7.2: Data-scope specification

Whole View. How to avoid data-access tunneling? Although data-contexts solves the problem of reading contextual data to execute an activity by creating a relevant data-environment, since every activity is executed in isolation to other changing activities agents do neither have flexibility nor global picture over cohesive process fragments. Exemplifying, *Review* activities to fill $(\text{InterviewNotes})^{1..n}_{user_I}$, Profile_{user_I}, $\text{AdditionalComments}_{user_I}$ and Grade_{user_I} should not be executed in sequence (even if the agent has the right to choose since edition and complete understanding of the process fragment is vital). A form is, at this scope, used to aggregate all the activities that are on the *running* states. The submission

of a form field is associated with an event that changes the activity state and possibly its related object state. If such changes produce new events that may turn activities *available* and then *running*, new form fields can be dynamically added and even removed (e.g. a submission of a form field may trigger an event that aborts or suspends a running activity).

This environment for the derivation of forms, although it has its roots on Case Handling, does not results from labeled associations to data objects, but derived from the data-filling constraints within the life-cycle. Affecting data-requirements at different states of a life-cycle mimic mandatory, restricted, optional and free aggregations in a more natural and complete way.

Recovering the Review scenario, when a Request instance is created Review object instances transit from *init* to *created* state triggering the transition of the introduced activities from the *init* to *available* state, that become *running* when a valid agent accepts them. As activities change their states at their own rhythm, the compound activity that aggregates them or the absence of precendences that makes them parallel result in a form with dynamically changing fields. Fig.7.3 depicts the form present during the Request execution for two distinct moments.

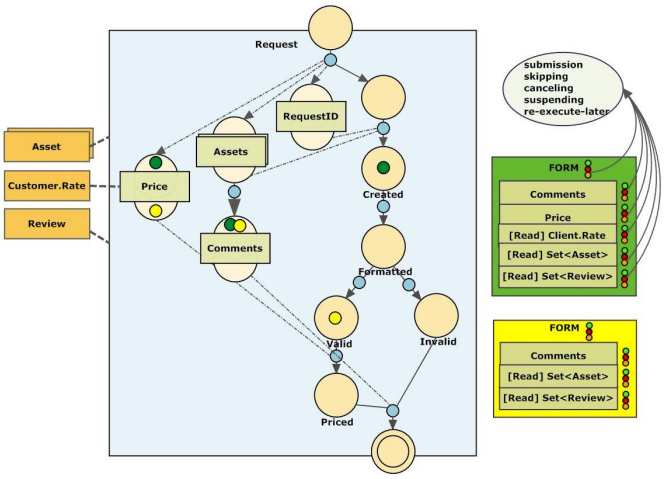


Figure 7.3: *Form*, the center notion for the execution of object-centered process models

Default Access. Is it possible to automatically generate expressive data-contexts? Until here we assumed that changing activities data-scope begins empty and become filled as a result of activity-object associations at different level of granularity. However, since activity instances are intricately related with object instances, advanced support patterns are possible. Object structures constitute a criterion to pre-define activities data scope. Three different criteria must be introduced deepening on the type of relationship and on their visibility label, thus bring the potentialities of the object-oriented paradigm to the process modeling discipline.

First, composition/encapsulation relationships are by default public, meaning that if an activity has access to an object, it has also access to all its internal objects. The same does not happens when the relationship is private and for the inverse relationship (assessing the parent object). *Second*, communication relationships are private by default, however if an object sets its relation with another object as public, its related activity has access to the other object's data. *Third*, inheritance relationships are public by default, meaning that an activity related to a specialized object has also access to its generalized object data as it is inherited by the specialized object.

Recovering the Request scenario, now depicted in Fig.7.4, we can detect that by defining visibility criteria by omission, the task of modeling data-access becomes simpler and, when not adjusted (as

intended to be on the basis of available visibility constructors), constraining of the data-scope to foster the focus of the agents increasing their efficiency is a possible way through the use of strikethrough line.

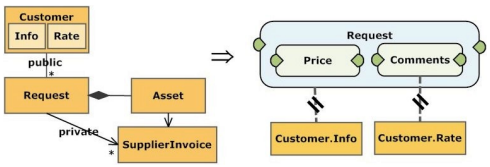


Figure 7.4: Data-scope by omission

Authorization. How to access non-contextual data? When data scope is specified for an activity, although it is provided a chunk of relevant information for an agent answer that task, in fact, we are also limiting its space ability or creativity since he has to answer the task using the provided data set. Without going back to the flexibility-traceability tension again, it is, however, comprehensive that the access problem is not yet finished. For instance, if an agent wants to evaluate a request he may needs to recover some of his past evaluations in order to leverage the accuracy of his task. Thus, authorized users must access data independently from the process progress, which requires a separation of authorization and agent-work distribution.

Domingos et. al [35] presents an object-based framework for the management of privilege access levels. By now we can assume that the authorization is defined according to a role-based access control with *CRUD* permissions being associated to object entities (that may exist at different granular levels) on the basis of agents' roles, skills, position and events-publishing (that can be synthesized as a data-field filling from a form), according to a specific criteria. These structured permissions are link between the informational and instrumental views. Fig.7.5 illustrates such scenario.

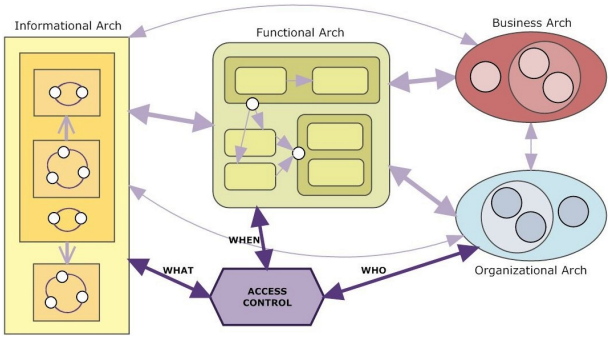


Figure 7.5: Understanding the privilege access levels for an object-based authorization

7.2 Data-state Reaction

In data-intensive systems the activation of an activity must not directly depend on the completion of other activities but occur as soon as its related object triggers a subscribed event. As a data-intensive system strongly relies on the data-flows among its system elements, system entities must instantaneously response to data-driven events and, thus, coordinate their interaction by synchronizing their life-cycles through event publishing and subscription.

Cross-model Reaction. How to turn activities more responsive and not dependent on a network of precedences? One of two sources is responsible to trigger a state-transition within a system entity. First, a form-field submission, cancellation, skipping or suspension. Forms are dynamically changing and their filling triggers all the system agent efforts. When a field is submitted, a data-field from an object

is changed which may affect the state of its related activity to transit to the succeed, failed or running states. Second, a state-transition from a related entity, with whom the target entity is synchronized. Exemplifying, the skipping of an activity may trigger an object transition, which may affect its synchronized objects and, consequently, the availability of their related activities. The event-driven basis that supports the target solution is exploited in Annex 15.

An activity begin- and end-criteria, the transitions affecting its availability and closing, must depend on objects' state-transitions instead of pointing directly to data-values conditions. This fosters the encapsulation, simplicity and adaptability of object models and enforce a correct life-cycle definition.

Two notes must be introduced on the life-cycles synchronization. *First*, as an entity may reside on any granular level, the life-cycle of a compound entity may *depend* on the state of its internal entities. Thus, an occurrence of a transition may not only depend on events triggered by external entities (association) but also by internal entities (composition). Fig.7.6 uses three objects from GF scenario to illustrate it. *Second*, the life-cycle of a compound entity model may *affect* the state of its internal entity models. Fig.7.6 exploits the default scenario for compound activity models.

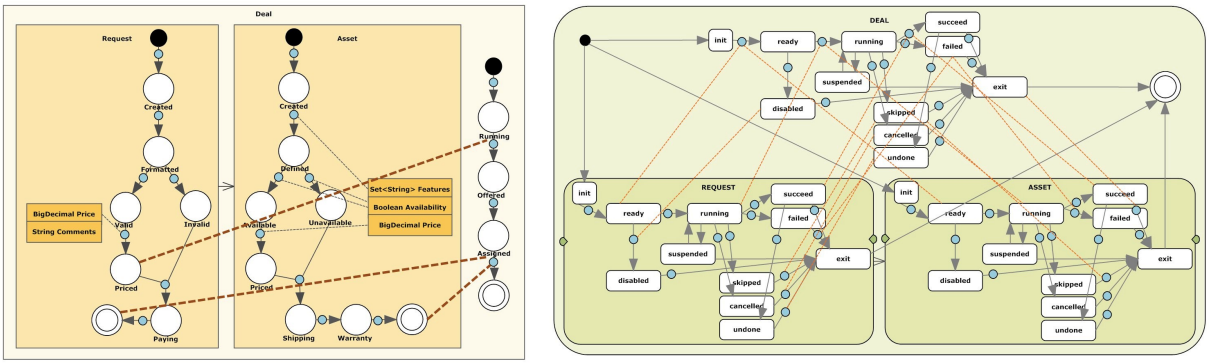


Figure 7.6: How compound entities depend and affect the state of its internal entities

Finally, the target object-centered approach also enables the incorporation of ordering precedences among activities. Note, however, that these precedences are just local, since it is inadvisable the definition complete end-to-end paths as this may hurt entities ability to react to events. Fig.7.7 presents an example of a local ordering constraint.

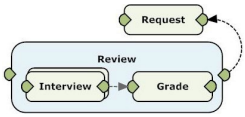


Figure 7.7: Definition of local precedences

Data Dependencies. How to use data models expressiveness to retrieve process basic constraints? If every simple activity is related the access or modification of an object, if a precedence exists between two activity models it also exists between two objects. As the targeted challenge by this thesis is on how to evolve system processes by adapting enriched data models, dependencies must be defined at the data level and dynamically translated into functional constraints. This observation has its roots on advanced document-based modeling.

Dependencies between attributes can be one of four types. First, finish-to-start, when the modification of an object requires the filling of other object. Second, start-to-finish, when the modification of an object it is only accepted – correspondent activity instance transits to *succeed* state – when another

object becomes enabled to change. Third, start-to-start, when a set of objects jointly become available for modification. Fourth, finish-to-finish, when an object change is automatically accepted when another object change succeeds. A useful exercise may be the reader to think on how a form-field entry and submission may affect a complete form if different types of precedences among data-attributes are used – these precedences foster an usable and object-centered way to capture expressive process constraints.

Additionally, such dependencies can reside at any granular level. In compound cases, we are not limiting a dependency to a specific attribute but to all the attributes aggregated by an object. Fig.7.8 illustrates the modeling of different types of data-dependencies at different granular levels.

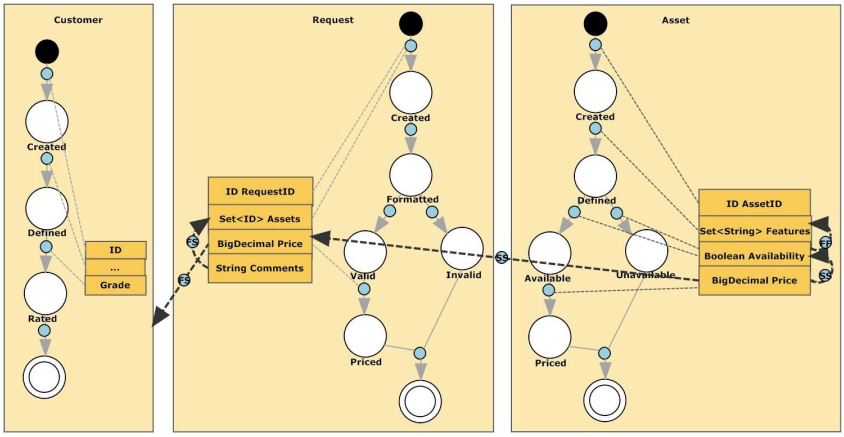


Figure 7.8: Precedences specification recurring to data dependencies

Note that the definition of these dependencies may not extinguish the need to define constraints for activity models, although default behavior is provided. First, activities define other transitions than *init* → *available* and *running* → *succeed*. Second, conditions for these transitions occurrence can be additionally constrained (e.g. definition of local precedences). Third, constraints can be defined for compound activity models not related with a compound object model – these compound activities are used to assign constraints or enacting roles-skills-tasks to a collection of activities – as these constraints cannot be derived using object constructors.

7.3 Data-based Coordination

Since data-intensive processes constrains are mainly grounded on data models expressivity, the specification of dependencies between process models must be based on simple and advanced methods to synchronize object models' state (directly affecting activities occurrence).

Process Communication. How to define loosely-coupled interacting processes? Previously it was presented that a state-transition of an object instance may depend on and affect the state of the related object instances. However, if an object (or, generically, a system entity) can be related to every entity system, the modeling is no longer usable (and, thus, agile and adaptable) since it is hard to understand the system big-picture, i.e., to capture the main events dependencies exchanged among a set of objects.

In order to leverage usability and granular consistency, the relations established by a set of objects are always encapsulated or mediated by a third object. This fosters a stronger intra-component cohesion and a weaker inter-component coupling [51]. It also enables a visual relation of the events affecting the synchronization of a set of life-cycles. Fig.7.9 uses a manufacturing scenario to exploit why encapsulation is relevant on the scope of a system production modeling.

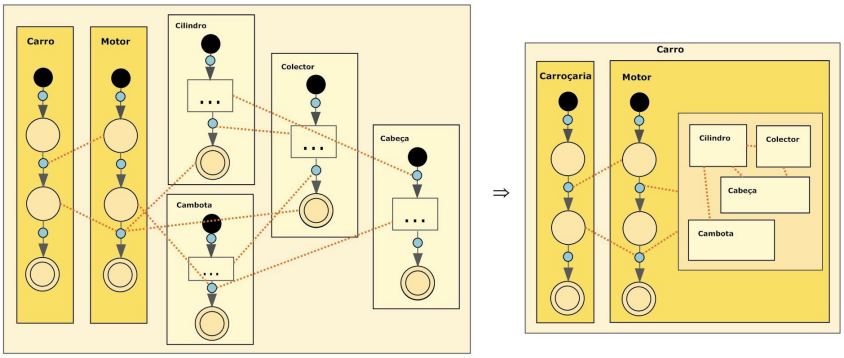


Figure 7.9: The need for objects' encapsulation

This principle is realized in two steps. First, based on data models' composition and aggregation patterns, a skeleton for the communication of objects is generated. Second, such communication basis is detailed by editing *i)* rule conditions that include constraints over the marking of the related objects, and *ii)* rule actions, used to affect those markings. These two simple steps enable the definition of points of synchronization among the processes automatically derived from these expressive objects relations. Each process can simultaneously progress in an asynchronously manner such point is reached.

Fig.7.10 introduces a scenario that depicts how such interaction pattern among objects affect processes synchronization. In this scenario two object models have their life-cycles synchronized, defining six synchronization points among the derived process models, six points-in-time when data is required to progress although they run in parallel – such loosely-coupled interaction is not supported by traditional approaches.

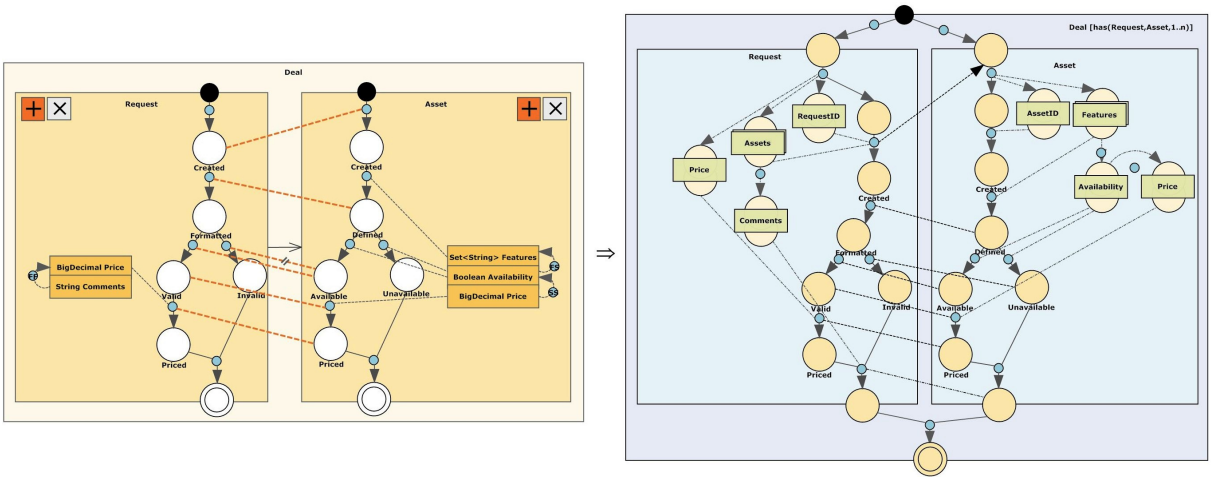


Figure 7.10: Synchronization of processes derived from objects communication

Default Behavior. How to create advanced communication patterns recurring to simple constructors? A system model may have a Request object instance related with an arbitrary number of Asset object instances. A typical point of synchronization for the Request object usually requires that all the Assets to be on a certain state (e.g. on Priced state in order to price the Request) or, at least, one related Asset to be on a specific state (e.g. on Unavailable state in order to reject a Request). Therefore, when using cross-object association arrows to express communicating patterns, the rule-set models must be expressively captured in a dynamic fashion.

A set of rules are used to derive automatically the basis for rule-set models. The *algorithm 7.1* is

a simplified method for this derivation. Fig.7.11 provides the application of this algorithm to two GF objects. Note that these generated rule-set models can be manually edited to express advanced behavior.

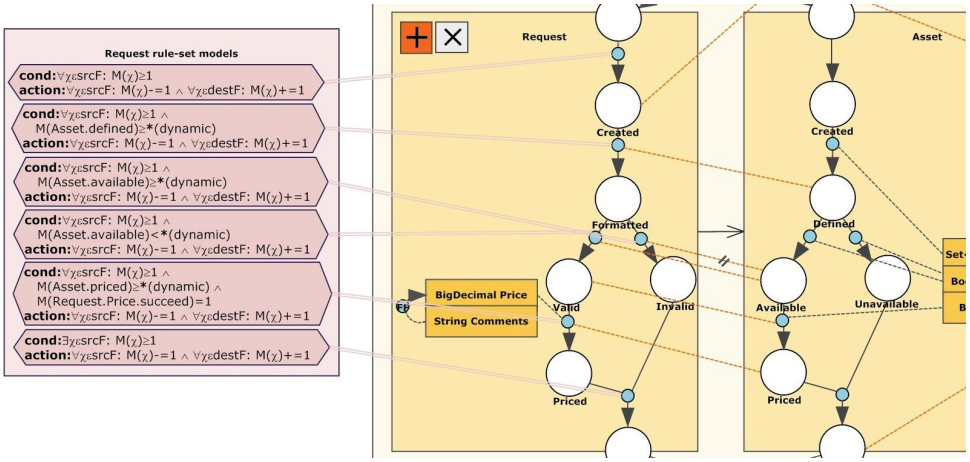


Figure 7.11: Default rule-set models dynamically generated

Algoritm 7.1: Generation of Rule-set Models

Input: Let F be the set of all transitions shared by the rule-set model RM
Input: Let $srcF = U(dom(F))$ and $destF = U(range(F))$ be, respectively, its input and output places
if *condition-not-edited(RM.cond)*$\vee$*condition-is-empty(RM.cond)* **then**
1. For all input places the next condition must be satisfied: $\exists x \in srcFM(x) \geq 1$;
2. For all input places the next condition must be satisfied when a strict (AND) association is used: $\forall x \in srcFM(x) \geq 1$;
3. For all the input places that do not belong to the object where the rule-set model is assigned ($E \subseteq F$) the next condition must be satisfied: $\forall x \in dom(E)M(x) \geq n$, where n is the cardinality (may be established dynamically) of the related object;

if *action-not-edited(RM.act)*$\vee$*action-is-empty(RM.act)* **then**
1. For all input places belonging to the object where the rule-set model is assigned ($A = F - E$) the next action must be taken: $\forall x \in dom(A)M(x)- = 1$;
2. For all output places belonging to the object where the rule is assigned the next action must be taken: $\forall x \in range(A)M(x)+ = 1$;
3. For all output places belonging to the object where the rule is assigned the next action must be taken if random progress (XOR) is used: $\exists x \in range(A)M(x)+ = 1$;
4. If the association enforces related model continuation, the next action must be taken: $\forall x \in dom(B)M(x)- = n \wedge \forall x \in range(B)M(x)+ = n$;

Advanced Behavior. How to model domain-specific behavior? Three main directions are pointed. Only the last one will be fully target as its contribution better addresses the third data-requirement.

First, the use of time formulas within the formal conditions and actions of rule-set models. Streams of research pointed in [48] present important contributions for this field of knowledge. A scenario is introduced in [31] where its modeling require time constructs to be used in relation to objects' historic data to affect rule's decisions. Additionally, process mining input can also be used to enrich the decision-making space assigned to a rule-set model.

Second, executable code can be assigned also to a rule-set model, enabling the extension of our formal syntax expressivity from sets theory to, possibly, a programming language. To support in the target object-centered approach, an engine that runs an algorithm that receives input markings as arguments and returns updated output markings must be developed.

Third, aggregation-separation constructors for sets of related instances must be introduced. Such constructors can also be placed in rule-set models, which result in an integrated multi-tab form, where each tab presents a set of data-fields both specific for each instance and shared by a set of instances.

Two different concerns are involved. First, the aggregation of a set of object instances which may have a common parent or related object instance. Exemplifying, the aggregation of all the running Assets associated to all of the Requests of a specific Customer. This can be done before the supplying of Assets so they can be collectively ordered. This vertical synchronization point is depicted in Fig.7.12 using notational convenience (the generation of its formal expression is studied on the next chapter). In order to satisfy this, relations between child and parent objects must be defined, and due to agents authorization and allocation properties, one agent may lose visibility from one or more of its form-tabs.

Advanced constructors, as the ones provided by Data-driven Modeling (see Fig.5.5) now become fully supported through life-cycle synchronization of multiple instances. Note that the Proclets Modeling potentialities were already supported using the previously introduced communication pattern, although it uses sockets to implement processes interaction possibly mediated by system applications[2] instead of relying on an event-driven structure.

Second, the aggregation of a set of object instances that may share something in common – grouping all the Assets within the same category and still not supplied. Fig.7.12 depicts such scenario.

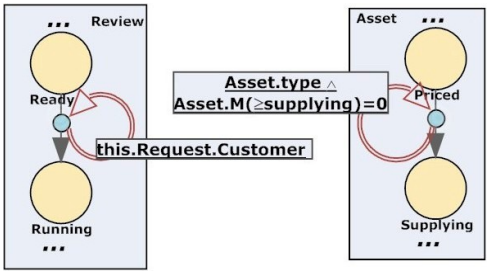


Figure 7.12: Two different aggregation rules for instances that: *i)* share an upper parent, and *ii)* have similar properties

7.4 Data-based Granularity

Modeling of system elements must preserve elementary atomicity and be possible at different levels of granularity, assuring they are coherently bridged and have a criteria for their uniform development.

Process models must comprise two principles at this scope. First, for each object data field there must exist an activity based on a service that accesses and modifies its content or on the semantically composition of these atomic services. Second, relation types among object models must constitute a criterion for the functional decomposition of processes towards atomic activities. For instance, if an object model Deal has a uses relation type with RequesterRiskRating and a composition relation type with a set of Review object models, it is possible to conceive a modular aggregation with two loosely-coupled process models, the Deal process (with Review sub-processes) and the RequesterRiskRating process.

Atomicity. For each object data-field there must exist an activity that triggers a set of system acts (possibly a semantic composition of system agents' efforts) to access, modify and submit its content. Since the basic axiom underlying this thesis is that every relevant action can be captured as an amount

[2]the target object-centered approach interact with agents (whether human or software) through form-fields filling as their intermediation would increase the modeling complexity and spoil the existing independence from the system instrumental layers

of data, the target form-centered approach assures that the system behaviour is always a constrained execution of atomic activities.

Note that the target approach, thus, have three different types of activities: atomic, simple, compound and framing, which are, respectively, related to data-attributes, simple objects, compound objects and framing objects. Simple, compound or framing activities are nothing but a state-based synchronized aggregation of atomic activities. We can hypothesize for future research that, if each atomic activity has a semantic meaning, their aggregation into compound activities (orthogonally from data-based compound activities) can automatically be obtained, and constraints can be automatically derived in order to satisfy the system goals.

An additional and orthogonal point to note is the fact that object authorization levels enable the use of atomic activities to update specific data-attributes. Compositions of atomic activities are possible if an agent is authorized to modify all of the related data-fields. For instance, to retrieve or update a customer phone number must be possible independently from process state.

Composition. Expressivity of data models must be used to guide the automatic functional decomposition of processes towards atomic activities, thus reducing the focus of system modeling that become focused on system constraining and adaptation. Three considerations must be introduced here.

First, relations of encapsulation among objects must constitute a criterion to compose processes. The generated process models must respect the objects' hierarchy and turn possible to zoom in and zoom out on this hierarchy depending on the agent privilege accesses. Fig.7.13 depicts simple constructors to define different levels of visibility for a process model (also valid for any governed entity).

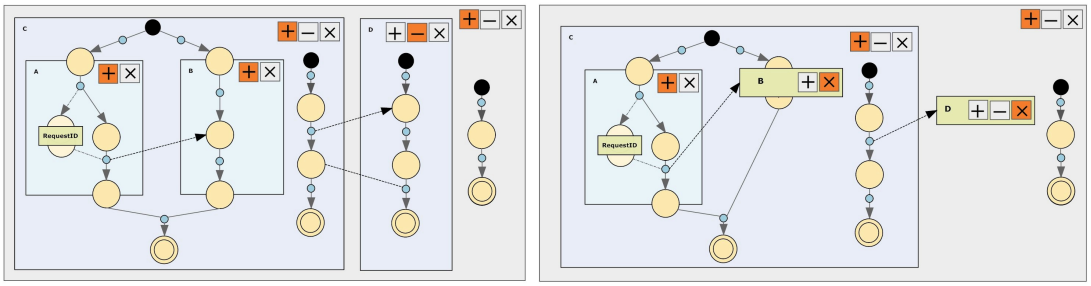


Figure 7.13: Zoom operations over different granular levels

Second, automatic generation of compound activity models based on other objects relation types (e.g. composition, extension) is also possible. For instance, if $O1$ depends on $O2$ and $O3$, and $O2$ depends on $O3$, two compound activity models can be derived: one comprising related activities to $O2$ and $O3$ objects, and another with all the objects. Since this property does not strongly impact composition principle, this thesis will not consider it.

Third, the use of framing objects or activities to aggregate cross-related objects or activities instances is also possible. These framings are used to facilitate the assignment of data-contexts, access-rights and common constraints, as illustrated in Fig.7.14. Note that zoom in and zoom out operations may become unavailable for regions where one frame crosses different levels of granularity.

7.5 Data Modeling

The modeling and adaptation of object models and their rule-set models must be possible at the process modeling level, thus, extending the source of adaptation from behavioral aspects to dynamical creation,

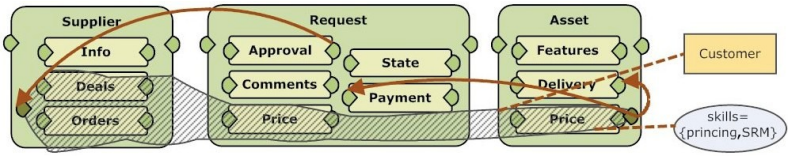


Figure 7.14: Framing activity will collective assignment of data, agent and constraints

edition and removal of objects.

Two main challenges exist. First, to assure that the new models are sound. If a correctness criterion is defined for object models we just need to apply them when a change is submitted to accept or reject with detailed reasoning the adapted model. Second, to support an adequate migration strategy. For now we can assume that executing process instances do not migrate (only new process instances) — the study of this topic will be simplified since the proposed mapping to YAWL as the implementation basis adopts already relaxed ways to migrate process instances.

The adding, removing and edition of object models and of their data-attributes, states, transitions, rule-set models, data-dependencies and inter-relation types must be covered. Since execution constraints for data-intensive processes mainly derive from objects models, these operations become the central source to evolve processes.

Additionally to the introduced source of adaptation, arising from flexibility by change, this thesis also analyzes (although not formalizes in detail) the flexibility of the target object-centered models in terms of deviation and deferring. Flexibility by deviation is used to embrace exceptions, so instances become more reactive as they can temporarily deviate from specifications until possibly standardized [99]. Flexibility by defer or underspecification is used to accommodate unforeseen situations without the complexity of defining all the paths in advance. This is done through the definition of incomplete processes that by specifying one or more placeholders, nodes whose content is unknown and specified later on by late binding, either only affecting present execution (static realization) or every subsequent execution of the process (dynamic realization) [99].

First, deviation patterns must be carefully applied since the target approach breaks monolithic processes into loosely-coupled communicating processes – if an entire process is deviated it may affect all of its related processes in a cascade manner. Two considerations must then be introduced to solve this problem. First, deviation must be reduced to process fragments limited by two points of synchronization. Second, deviation of processes containing one or more points of synchronization is possible since: *i)* the affected process instances become also deviated, *ii)* the deviated places affecting the synchronization points still become filled, or *iii)* the satisfaction of the rule-condition that affects the progress of the related instances forced.

Note that deviation can also be locally applied to rule-set models, as formal and algorithm-based conditions and action can be case-by-case deviated to test the ability of a new system behavior to answer to its internal and external changing conditions in a trial bottom-up fashion.

Second, placeholders definition and their late binding, in line with what was said regarding deviation limitations using the target approach, must also consider the potential effects on the related processes if a placeholder covers object fragments with one or more places associated to a synchronization point. In this case, a placeholder must turn mandatory the use of such places when binding a specification.

Note also an object with an unclear role (e.g. a type of supplier, with whom the target system

does not yet structured the data to exchange) may be itself a full placeholder with a set of mandatory communication-required places. Also, dataholders on the object's information model is possible, if some system production does not have a standardized structure. These aspects enable the definition of incomplete models for systems and their gradual refining and completion.

IV Development

8

Object-centered Models Formalization

Sendo impossível exprimir o Absoluto, só o Símbolo o pode conter.
Pela sua capacidade invocadora, acorda no inconsciente um Saber perdido.
Traz o inconsciente ao consciente.
– Donald Kuspit, *Concerning the Spiritual Contemporary Art*

In last chapters, process models' basis was defined as a set of event-driven synchronized state-based entities and their properties were derived. *Sections 8.1*, *8.2* and *8.3* formally describe the object-centered models and their soundness criteria, and *section 8.4* presents the steps required for the derivation of a process model based on its governed models.

8.1 Object Models

The notion of object was already defined. Objects are system entities containing data-elements, a life-cycle that may relate with other objects, either through communication, inheritance or mediation.

A *simple object model* is a tuple $< A, P, F, C, M >$, where:

- A is a set of data-attributes. Each attribute, $a \in A$, is a tuple $< tp, V >$ where tp is its data-type and V ($\subseteq \wp$, all tp domain values) is a set with zero, one or multiple values (if tp:=*Set*$< tp >$);
- P is a list of places/states;
- F is a set of flow relations or transitions ($\subseteq (P \times P) \cup (P \times A) \cup (A \times P) \cup (A \times A \times \{FF, SS, FS, SF\})$);
- $C : F \rightarrow RM$ assigns a rule-set model to a flow relation[a]. Rules, RM, are essentially pairs $< ID, cond, act >$, where: *i) cond* is a set of conditions either based on the object data-values $\cup_{i \in A} V_i$ (or, more accurately, over the state of their related atomic activity models: $\cup_{am \in AM} M(am)$) and over the marking of its input transitions' places, $srcF = \cup(dom(F))$, and *ii) act* is a set actions performing changes on the marking of the input and output transitions' places, $srcF \cup (range(F))$;
- M is the model state or marking defined by a function $P \rightarrow \mathbb{N}$.

[a] C is not a partial functional, since every transition has a condition. See Alg.7.1 for default rule models' generation

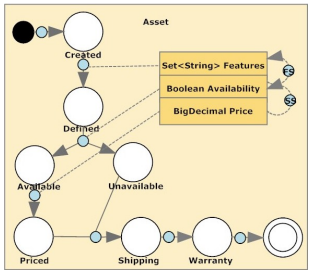


Figure 8.1: Simple Object Model

An example of a simple object model, *Request*, including data-dependencies and generated rules, is

presented in Fig.8.1. Its state is defined by a marking M joining individual places marking. Since P is a list (its elements have an index – e.g. P(0)=init, P(1)=created...), a marking can be defined by a vector. Exemplifying, if M=[0 1 0 0 0 0], M(created)=1 and M(init)=M(formatted) =...=0, there is one Request instance in the *created* state. However if we would want to address all of the Request instances running in GF, different markings such as M=[0 0 0 0 1 2], decomposable in a matrix with six columns and n rows, being n the number of Requests (e.g. M(1,6)=1).

A possible rule-set model assigned to valid→priced transition would be: C(valid→priced)=< *cond*,*act*> with emphcond={Price!=NULL}. However, it is important to understand that each data-attribute has a correspondent atomic activity with an end-criterion (by omission, the submission of a not-null form-field value). Thus, preferably, the condition can be defined as *cond*={Price.succeed} since activities may rely on multiple data-value conditions for each transition.

Let us introduce the semantics to study soundness criteria. Let $< P, T, C >$ be a state-machine and M its marking. The firing of a transition is a state change occurring when its rule condition is satisfied [120]:

- $M \longrightarrow^{t} M'$ indicates that by firing t, the state of the machine net changes from M to M′;
- $M \longrightarrow M'$ indicates that there is a transition t such that $M \longrightarrow^{t} M'$;
- $M_1 \longrightarrow^{*} M_n$ means that there is a sequence of transitions $t_1, t_2, ..t_{n-1}$ such that $M_i \longrightarrow^{t_i} M_{i+1}$ for $1 \leq i < n$;

A state M′ is *reachable* from a state M if and only if $M_1 \longrightarrow^{*} M'$.

A *simple object model*, $< A, P, F, C, M >$, is *correct* if and only if the following conditions hold:

- it is *structurally sound*, that is, it has two distinguished places: $i \in P$ with no incoming edges and $f \in P$ that has no outgoing edges, and every place is located on the path from the i to f;
- for every marking M reachable from state i there exists a firing sequence leading from M to o: $\forall M(i \rightarrow^{*} M) \Rightarrow (M \rightarrow^{*} o)$ with no dead transitions, i.e., $\forall_{t \in T} \exists_{M,M'} i \rightarrow^{*} M \rightarrow^{t} M'$;
- the attribute dependencies, $A \times A \times \{FF, SS, FS, SF\}$, are live and bounded;
- the attribute optional start place markings precedes the end place markings and both are reachable, i.e., $\forall_{a_i \in A, (p_0, a_i) \in (P \times A) \vee p_0 = i, (a_i, p_1) \in A \times P \vee p_1 = o} M(p_0) \rightarrow^{*} M(p_1)$;

If a simple object model is not correct since it has not a distinguished init and final states, an init state must be added with transitions to all beginning states with default rule-set models, and a final state linking all the end-reachable states with a default rule (if no ending state is eligible due to graph cycles, the object is not sound).

An *inherited object model* from one or more object models is a tuple $< \zeta, P, A, F, M, C >$, with ζ being a set with its super-objects, and $< P, A, F, C >$ net resulting from one of the two following methods: *i)* if no super-places are referred, the super-life-cycle (with its attributes-constraints) is, by default, executed in parallel; *ii)* if super-places are referred, it inherits the constraints from the referred fragment. The study of overriding options and polymorphism methods for advanced rules is out of this thesis scope.

Fig.8.2 illustrates the two inheritance methods currently supported by the developed approach. The application of such methods assures the soundness of the derived models.

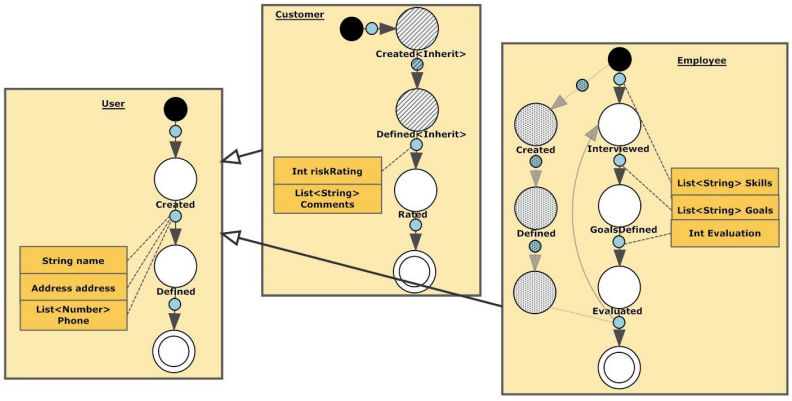


Figure 8.2: Two strategies for *inheritance* of data and constraints from a super-object model

A *compound object model* is a tuple $< P, A, F, M, C, \Theta, P', F', M', C' >$, where:

- $< P, F, M, C, A >$ is a simple object model;
- Θ is the set of all internal object models;
- $< P', F', C' >$ is the derived life-cycle resulting from the synchronization among internal objects, $O_i \in \Theta$, and the compound object life-cycle $< P, A, F, C >$, such that $P \cap P' = \emptyset$, $F \cap F' = \emptyset$, $F' \subseteq ((P \cup A) \times P_\Theta) \cup (P_\Theta \times (P \cup A)) \cup (P_\Theta \times P_\Theta))$ and such that a rule is assigned for each new flow relation, $C' : F' \rightarrow RM$;
- $M' : P' \rightarrow \mathbb{N}^n$ (with $n \in \mathbb{N}$) is the compound object model multi-colored marking.

Fig.8.3 exploits a simplified compound object model, *Deal* or the unnamed *has(Request,Asset,1..n)*, relating one *Request* object instance with a set of *Asset* object instances. The *Algorithm 8.1* defines the object model transformations required to guarantee basic soundness criteria for a compound object.

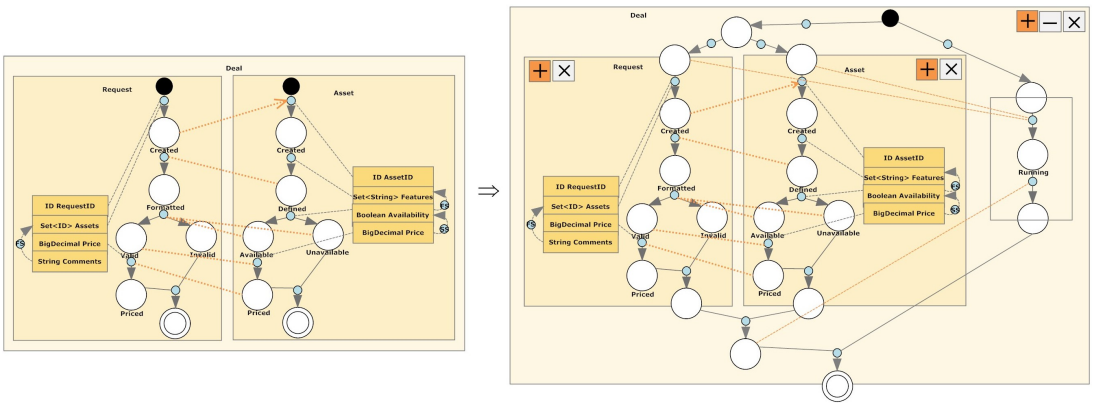


Figure 8.3: Compound Object Model

When the encapsulating object's life-cycle is not explicitly define, $< P, F, C, A >$ is automatically generated with new flow relations to its internal objects, as illustrated in Fig.8.3. A sound place net $<P', T', M', C'>$ results from the interaction of this life-cycle with the internal life-cycles.

Another relevant aspect to exploit in this definition is the M' marking. Since several Asset instances may belong to different Request, no longer a simple marking, $M' : P' \rightarrow \mathbb{N}$, is enough. Instead of decomposable vector, for the Deal example we need to have a 2-dimensional matrix that for each request (lines) obtains a vector with the correspondent assets marking (columns). Depending on the objects under relation this marking is, generically, a matrix with n dimensions. The use of these colored markings is further visited on next chapter, and its application in advanced rules discussed.

Algoritm 8.1: Derivation of a Sound Net for Object Models Composition

Input: P,F,C,A,Θ with | Θ |= $n \wedge \cup_i^n om_i = \Theta$
Output: P,F,C,A,P',F',C'
relatedObjs=∅, newInitialStates= ∅, newFinalStates= ∅, p= *null*, id=newID();
P'=$\cup_i^n(om_i.P)$, F'=$\cup_i^n(om_i.F)$, C'=$\cup_i^n(om_i.C)$;
foreach *internal object* $om_j \in \Im$ **do**
  foreach *internal object* $om_k \in \Im$ **do**
    if $\{om_j, om_k\} \notin$ *relatedObjs* **then**
      relatedObjs = relatedObjs∪({om_j,om_k})∪({om_k,om_j});
      p=$i_(om_j.om_k)$; p=$f_(om_j.om_k)$;
      newInitialStates = newInitialStates ∪ {p}; newFinalStates = newFinalStates ∪ {p};
      P'=P'∪ {p}; F'=F'∪ {p → i_{om_j} }; F'=F'∪ {p → i_{om_k} };
      P'=P'∪ {p}; F'=F'∪ { f_{om_j} →p}; F'=F'∪ { f_{om_k} →p};
P'=P'∪ $i_{compound}$; P'=P'∪ $f_{compound}$;
foreach $i_j \in$ *newInitialStates* **do**
  F'=F'∪ ($i_{compound} \rightarrow i_j$), C'=C'∪ (($i_{compound} \rightarrow i_j$) → ($id, true$));
foreach $f_j \in$ *newFinalStates* **do**
  F'=F'∪ ($f_j \rightarrow f_{compound}$), C'=C'∪ (($f_j \rightarrow f_{compound}$) → ($newID(), \forall_{p \in newFinalStates} M(p) = (\sum_i^{|P|} M(i))$));
if *P=null* **then**
  P={init,running,final};
  F={f1=init→running,f2=running→final,init→newInitialStates,newFinalStates→final};
  C={f1→<M'!=[1 0 0 0 0 0 ... 0],gen-act()>,f2→<M'(final)=1,gen-act()>};
foreach $a_i \in A$ **do**
  If $\nexists_x$ x →a_i Then init→a_i;
  If $\nexists_x$ $a_i \rightarrow x$ Then a_i →final;
generate-rules(P',A,F',C') *%% see Alg.7.1*;
soundness-checking(P',A,F',C');

Having defined the structure of a compound object model, we need to define its correctness criteria.

A *compound object model*, $< P, F, M, C, A, P', F', M', C' >$, is *correct* if the following conditions hold:

- $< P, F, M, C, A >$, viewed as simple object model, is correct;
- all of its internal objects Θ are correct;
- exists an algorithm that, based on internal objects interaction, defines a constrained places' net $<P', F', M', C'>$, with <P=P',F=F',M=M',C=C',A=null>, viewed as a simple OM, being sound.

Returning to our example, the *Deal* object model is sound since: *i)* its $< P, F, M, C, A >$ elements are derived by omission and its correctness automatically assured, *ii) Request* and *Asset* objects are correct, and *iii)* there is an algorithm, Alg.8.1, that assures the third property.

8.2 Activity Models

Traditional approaches use activities as the key aspect to bridge the different system concerns – activities are enacted by some agent (link to instrumental models) in a certain range of time (link to temporal dimension), using a set of resources (link to object models) and affecting the set of system goals (link to contextual models). In data-intensive landscapes these aspects become bridged by expressive data models – agents are assigned to the outcomes they are able to produce, data-contexts are determined by data-delivery dependencies and data-access levels, and goals are driven by the system productions.

Activities, in the target object-centered landscape, encapsulate and abstract from objects all the functional aspects as transactions-concerns, extraordinary behavior like canceling, skipping and suspension options and their non-local effects. Object models become simpler, thus more usable, agile and evolutionary, as they become centered on the data progress and dependencies needed to deliver a production.

Taxonomies exist in literature that classify activities in primitive and complex, elementary and composite or, more interesting, in atomic and compound (if an activity type can be expressed by more than one activity types) [98]. The notion of compound activity is here used as a container for sub-activity models (disregard of constraints) as their atomic parts can actually acquire a life of their own and furthermore they may be regrouped with collections from other process instances [98]. Therefore, seeking consistency, this work does not adopt the WfMC standard definition for activity [34] (equivalent to the sub-process notion), since, as introduced, execution constraints are separated from the activity concept.

Activity models can either be *structured* (prescriptive), if explicitly represented, or *unstructured* (descriptive), if focused on what to do (not on the how) [98][12].The object-centered activity models are unstructured as they base the activity progress on the state of its result (data-filling), thus, do not limiting the agents' enactment creativity. Additionally, *formal* models are used to model activities, although *informal* models or unstructured semantics can be attached to enrich their description [120]. Finally, an object-centered activity can either be *internal* or *external*, depending if they are enacted by the system or environmental elements, as both derive from an complete object and react on standardized events.

An *atomic activity model* is a tuple $< a, P, F, C, M, D, nofi >$, where:

- $a =< tp, V >\in \alpha$, is the activity outcome, its related data-attribute;
- $< P, F, C >$ is a normalized (but customizable) state-machine, with $F \subseteq (P \times P)$ and $C : F \rightarrow RM$, where $RM =< ID, cond, act >$ are defined by omission, but the condition assigned to the transition to the final state is usually edited with an expression to test the validity of the related data-attribute values V when the respective form-field is submitted;
- M is the model state or marking defined by a function $P \rightarrow \mathbb{N}$;
- D is a set with the accessible objects' data-values, the activity's data-scope;
- $nofi = \mathbb{N}_{inf} \times \mathbb{N}_{inf} \times \{dynamic, static\}$ specifies the number of the instances – minimum, maximum, threshold for continuation, and dynamic or static creation of instances, used for multiple iteration when *tp*=*Set*<tp>.

This is a simplified definition. Advanced topics as locks-acquisition for data-manipulation or transactional properties were intentionally left out since they are out of this thesis scope. Note that several authors instead of using state-machine notation refer to events and their orderings with the same propose. Fig.8.4 presents two default atomic activity models, one with a plain structure and another with compound states where execution norms from state-condensation theory must be applied.

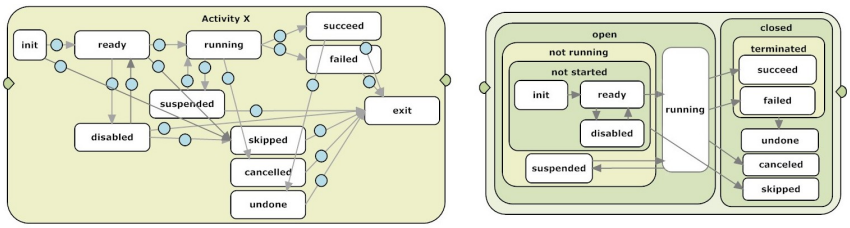


Figure 8.4: Atomic Activity Models: plain and using compound states

Rules can be assigned to transitions between compound states and enforce marking changes at a higher-level. Multiple marking functions for the different composition can be used – e.g. M=[1 0] is a higher making to specify *active* or *closed* instances, and M=[2 0 3 1] is a lower marking that details instances in the *enabled*, *suspended*, *running* and *completed* places. For the sake of simplicity we will be focused on plain nets of places.

Every activity model adopts its name from the related data-attribute, although can edit for informal semantics coherency (e.g. *price* > *toPriceRequest*). An activity becomes available when its form-field becomes visible (which coincides with an event triggered to the skill- and task-based subscribing agents), i.e., when the constraints of its object model turn possible its edition.

Although, transition rules are defined by omission, criteria from the data-attribute's object model can be dynamically use to strengthen the rule. Exemplifying, init→ready, in the absence of additional constraints, immediately occurs when an instance is created. ready→running occurrence depends on the state of the enacting agent. running→succeed transition uses the relaxed condition, $a.V$!=null, to set its occurrence. Additionally, the canceling, skipping, suspension or undone transitions by omission can occur if such option is selected in the form under submission or the respective event is triggered. Activity machine-edition enables to restrict some of these extraordinary options and their compound activities to control their non-local effects.

Finally, $nofi$ enables the run-time creation and iteration of instances. If the *Reviews* activity (related to Set<ID> *Request*'s data-attribute) $nofi$ is [n 5 3 d], this means that, at least, five reviews are dynamically

created and the activity will stop when three instances reach their final state.

A *simple activity model* is *correct* if $<A = null, P, F, C>$, viewed as a simple object model, is sound; and its transitions with rules based on the related data-values can occur $(\forall_{a \in A} \exists_{a.V} (cond(a.V) = true \wedge a.V \subseteq \wp))$.

Activity models can either be atomic or compound, containers for atomic activities. Simple activities are compound activities that are related to simple objects. The hierarchical alignment between activity and object models is illustrated in Fig.8.5.

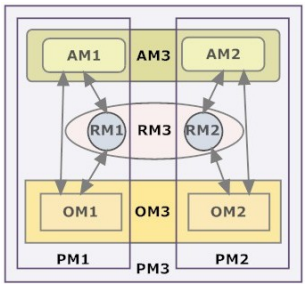


Figure 8.5: Modularity of a Process Model based on its Governed Models

A ***compound activity model*** is a tuple $< O, \Lambda, P, F, M, C, D, P', F', M', C', nofi, effect >$, where:

- O is the related object model, possibly empty if the activity acts as a frame;
- Λ is the set of all internal activity models;
- $<P, F, C>$ is a standardized (but editable) state-machine with a marking M and a data-context D;
- $<P', F', C'>$ is a dynamically derived state-machine based on the interplay of activity places-net with the internal activities places net, with a marking $M : (P \cup \Lambda_P) \rightarrow \mathbb{N}^n$ (with $n \in \mathbb{N}$);
- $nofi = \{\mathbb{N}_{inf} \times \mathbb{N}_{inf} \times \{dynamic, static\}\}$ defines its number of instances;
- $effect : \Lambda_P \mapsto \mathcal{P}(P' - \{i, o\})$ specifies the subnet of the $<P', F', C'>$ state-machine that may become cancelled, skipped, suspended and undone when an internal activity transits to one of those states, where $\mathcal{P}(P')$ denotes the powerset of P', a net region.

Fig.8.6 shows how the state of a compound activity affects and is affected by the state of its internal activities. This complex state-machine is a simplification of the $<P', F', C'>$ net, which is automatically derived on the basis of a formal algorithm that assures its soundness properties. In this example *Deal* compound activity aggregates two simple activities, each one being a container for new atomic activities.

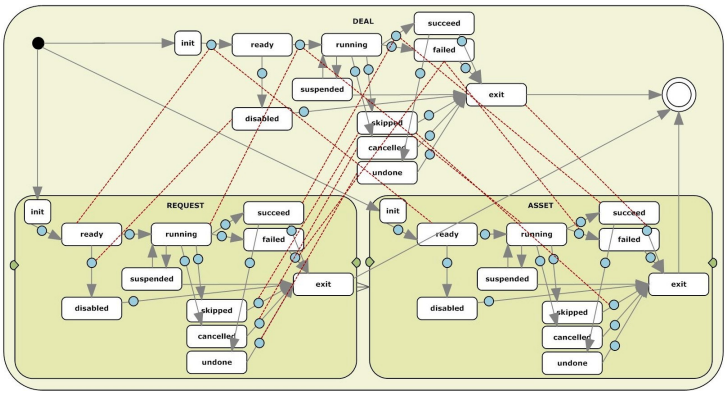


Figure 8.6: Compound Activity Model

The transitions added to develop a sound $<P', F', C'>$ net, have default conditions, that may be adapted into a new template depending on the data-intensive system's properties. For instance, *running* →

finished can either be triggered when all of its inner activity instances are in one of the next states: *finished*, *succeed*, *failed*, *cancelled* and *skipped*, or when its related object instance is one the *final* state, or even when a sub-set of its inner activity instances reach the final place. Two example of the attached conditions are: C:(init$_{Deal}$ →ready$_{Deal}$)⟶{ *id*,$\exists_{am \in AM} M(am.ready) \geq 1$,default-act}, and C:(running$_{Request}$ → cancelled$_{Request}$)⟶{ *id*,super().M(cancelled)+super().M(skipped)=1∨Request$_{form}$.isCancelled(),default-act}.

Additionally to this view, each internal activity model can hide some of its internal aspects. Using this view, an user can add ordering constraints among activities, thus possibly defining activity-centered paths or just important local ordering (recap Fig.7.7). Nevertheless, the definition of such constraints is discouraged since: *i)* ability to evolve process models based uniquely on underlying object models is no longer possible, *ii)* models flexibility is hurted, and *iii)* data-dependencies (FF,FS,SF,SS) can be used at the objects modeling level to define similar local-ordering constraints.

Activities with $\Theta = \emptyset$ are not related with an object model. They just aggregate related-activities that may belong to different objects. These activities act merely as frames, thus, their *nofi* is not editable. They are, by default, instantiated when one of its internal activities become available, and archived when all of its internal activities reach the final place. Several benefits are seized as related activities are aggregate so it is possible to collectively: *i)* assign data-access contexts, *ii)* define the activities tasks and skills for an usable agent-allocation, *iii)* define local constraints, and *iv)* affect activities that are related by non-local effects from extraordinary operations as cancelling or suspension.

A *compound activity model* is *correct* if and only if:

- $< \Theta = \Lambda, A = null, P, F, M, C, P', F', M', C' >$, viewed as a compound object model, is correct;
- all of the internal activity models, $am \in \Lambda$, are correct (if internal activity models are compound models this criteria must be recursively applied until simple activity models criteria is reached);
- the *effect* function changes the marking into a new *reachable* marking.

8.3 Rule-Set Models

Rules are used to model the execution constraints of a process. They are dynamically defined (although editable) at the object, activity and goal modeling levels, and are grounded on formal set theory. The automatic-generation of rules turns the specification of objects the modeling central aspect, leveraging the focus in data clustering, progress and dependencies to deliver the target productions. Additionally, it fosters process modeling simplicity and expressivity required for the objects adaptation.

A ***rule-set model*** is a tuple $< ID, cond, action >$, where:

- ID is the identifier for a set of transitions shared by the rule ($F : src \rightarrow dest$);
- *cond* is an expression based on: *i)* data-conditions, on *ii)* time-state, and on *iii)* the marking of its transitions' source places ($SrcP = \cup^{|F|}_{f \in F \wedge dom(f) \in P} dom(f)$);
- *action* is a function responsible to change the marking of the rule's referred places ($\forall_{f \in F \wedge (dom(f) \cup range(f)) \in P} (dom(f) \cup range(f)) \rightarrow \mathbb{N}$);
- <*cond*, *action*> can be formulated in the scope of: *i)* a *specific* instance or *ii) multiple* instances, either *related* by a super or interacting entity or *non-related* (synchronized through data-similarity).

The way rules aggregate transitions can originate different control-flow patterns. In Fig.8.7 some of the different patterns are exploited. However, the modeler must neglect this gateway-driven awareness to avoid a biased thinking that can limit the ability to specify advanced synchronization patterns.

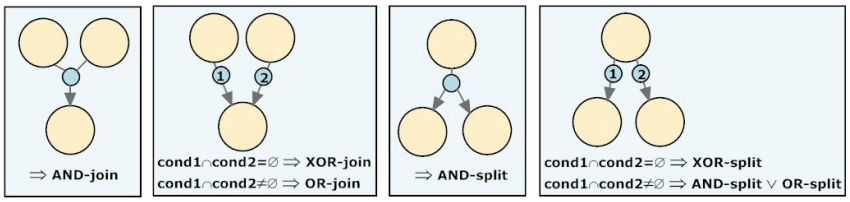


Figure 8.7: Mimic of simple gateway-patterns recurring to rule-set models

Rules are pairs condition-action, and may refer two types of variables: *i)* data-attributes of the object where the rule is placed (e.g. $Assets_{(\in Request.A)}! = null$ or, more accurately, $M(succeed_{Assets_{AM}}) = 1$), *ii)* places markings of the super-, sub- and interacting-objects. These variables are arranged in an expressive formula using set-operators (e.g. $\wedge$,$\vee$,$\exists$,$\forall$), mathematical notation (e.g. Σ,+,Π) and time-constructs.

Modelers of traditional approaches often recur to non-source-traceable events to specify complex rules, as a data-change or the reception of a specific e-mail or supplying order. Non-source-traceability means not knowing what activities originate a change that triggered an event. However, in our object-centered landscape, if events corresponding to an e-mail conversation or order are triggered, is because these entities are modeled as objects, and, thus, can interact with the object under modeling without restricting the ability to trace events-causality.

The syntax for rule-set models fit well with formal languages [31][48][43]. Future research must deepen: *i)* the study of its limitations and potentialities, *ii)* enrich it in a structured way using already existing contributions as [94], and *iii)* turn their definition graphically expressive. In fact, we already provide advanced aggregation-separation constructs in an usable notation as illustrated in Fig.8.8.

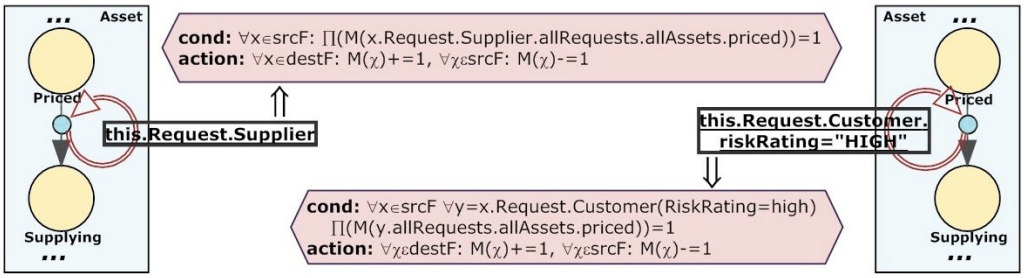


Figure 8.8: Notational convenience to depict advanced rule-set models

A rule is an event-subscriber (subscribes to related data and places-marking changes to evaluate its condition) and an event-publisher (informs entities of the new places-marking when its condition is satisfied). Annex 17 exploits in more detail the implementation concerns. Rules are, in fact, the glue to support the thinking of a system as a set of event-synchronized entities – where their progress (measured by their markings) and data are the key aspects for their dynamic response.

8.4 Object-centered Process Models

Having formalized the notions of object, activity and rule-set models, we need now to bridge all of their information in order to derive a complete and sound process net. Only by joining the operational and functional aspects, it is possible to derive an executable process net, where quality metrics can be studied to detect bottlenecks. This new integrated structure must preserve the causal dependencies of the original models, thus, can also be referred as a causal matrix or a heuristic net.

A *simple object-centered process model* is a tuple $< OM, AM, P, T, F, M, C, D,$*,nofi,effect*$>$, such that:

- OM and AM are, respectively, its related simple object model and activity model;
- T is the set of atomic tasks related to $AM.\Lambda$ and P is a list of places (P=$OM.P \cup AM.P'$);
- F is a set of transitions between places and tasks such that $((P \cup P) \cap (P \cup T) \cap (T \cup P) \cap (T \cup T)) = \emptyset$;
- $C : F \rightarrow RM$ assigns rule-set models to flow relations;
- M is the process model state or marking (M: $P \rightarrow \mathbb{N}$);
- $D = AM.D$ is a set with the accessible objects' data-values, the process data-scope;
- *nofi*=*AM.nofi* and *effect*=*AM.effect* are the iteration and behavior-control for net regions;
- $<OM, AM, P \cup T, F, M, C, D, \emptyset, \emptyset, \emptyset, \emptyset,$*nofi,effect*$>$, viewed as an activity model, is sound.

Fig.8.9 presents how a simple object and activity model compose a process net. This illustration is a simplification of the real scenario where each task T is decomposed in a set of places linked to the functional places $AM.P'$. However, this is the default presentation to preserve the modeling usability. The most important steps for this derivation are captured by the following algorithm:

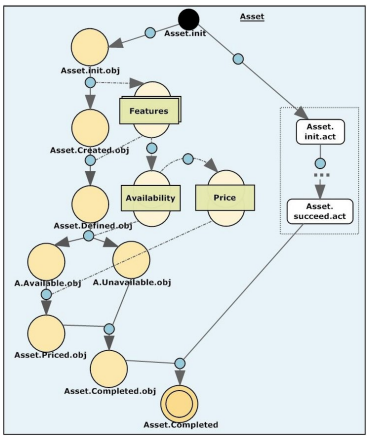


Figure 8.9: Simple Process Model

Algoritm 8.2: Derivation of a Sound Process Net

Input: object-model OM; activity-model AM
Output: $< OM, AM, P, T, F, M, C, D = AM.D,$*nofi=AM.nofi,effect=AM.effect*$>$
1. $< P, F, C >$=retrieve-net-basis-using-name-disambiguations(OM,AM);
2. P=P∪{init}; F=F∪{init→init$_{OM}$}; F=F∪{init→init$_{AM}$}; 3. P=P∪{final}; F=F∪{final$_{OM}$ →final}; F=F∪{final$_{AM}$ →final};
4. T=translate-data-attributes-into-tasks($OM.A$,$AM.\Lambda$);
5. F=F∪unfold-data-dependencies-into-ordering-constraints(T,OM);
6. F=F∪link-tasks-to-init-place-attributes-without-ingoing-arc(T,OM);
7. F=F∪link-tasks-to-final-place-attributes-without-outgoing-arc(T,OM);
8. C=derive-all-rules-not-edited(P, T, F, C);

The later completion of the process net, makes the modeling of data-attributes and objects dependencies not biased by the need to insert them in complete network of precedences. By not thinking in paths specification, the target approach fosters a natural parallelization of activities.

To include the communication and encapsulation aspects, the notion of compound process model must be introduced. Note that inheritance patterns are already supportted by the object transformations.

An *object-centered process model* is a tuple <$OM, AM, \Theta, \Lambda, P, T, F, M, C, D$,$P'$,$T'$,$F'$,$M'$,$C'$,*nofi*,*effect*>, where:

- < OM, AM, P, T, F, M, C, D,*nofi*,*effect*> is a simple process model;
- Θ is the set with all internal object models of OM;
- Λ is the set with all internal activity models of AM;
- $< P', T', F', C' >$ is the derived process net resulting from the synchronization aspects among internal entities, $O_i \in \Theta \cup A_i \in \Lambda$, and the compound process life-cycle $< P, T, F, C >$, such that $P \cap P' = \emptyset$ and $F \cap F' = \emptyset$, with $C' : F' \rightarrow RM$ assigning a rule for every derived flow relation;
- $M' : P' \rightarrow \mathbb{N}^n$ (with $n \in \mathbb{N}$) is the process model multi-colored marking.
- < $OM, AM, P \cup T, F, M, C, D, P' \cup T', F', M', C'$,*nofi*,*effect*>, viewed as an activity model, is sound.

Note that the need to isolate $< P, T, F, M, C >$, and not simply present the derived process net $< P', T', F', M', C' >$, is to turn the encapsulation mechanisms possible. If a new mediator is defined and uses a compound process model as an internal model, it has only access to its high-level life-cycle $< P, T, F, M, C >$ in order to reduce the range of objects' dependencies.

Fig.8.10 illustrates a derived process model with a marking retrieved from the *Deal* object model, assuming that no changes were done to the object-derived activity model. We used multi-colored tokens to distinguish instances. These advanced marking, M', are based on multi-dimensional matrices to improve the ability to specify advanced rules, and manage multiple instances. The following chapter will deepen on its implementation.

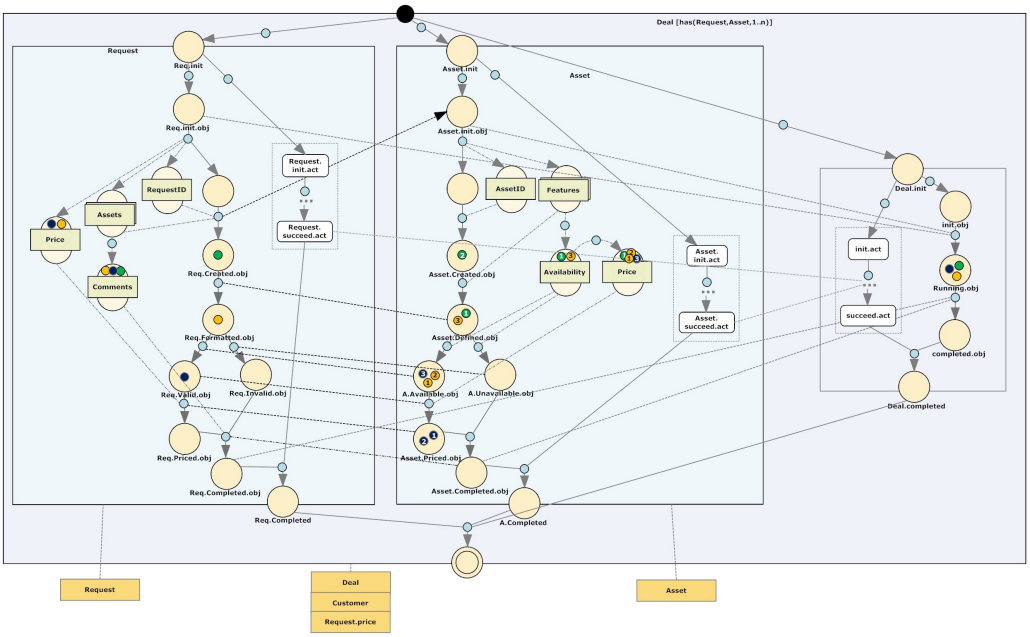


Figure 8.10: Compound Process Model with a Marking

8.5 Object-centered Soundness Criteria

Note that simple correctness criteria were defined for each object-centered model. But which type of soundness does the target modeling approach assures? Normal? Relaxed? Weak? Lazy? The different kinds of soundness criteria are tailored towards specific environments, where the original soundness appears to be too restrictive [120]. In order to assess this, four sound properties are introduced [120]:

- *Termination*: any process instance that starts in the initial state can reach the final state, that is, for

every reachable marking there exists a firing sequence the final state: $\forall_M(i \longrightarrow^* M) \Rightarrow (M \longrightarrow^* o)$

- *Proper termination*: the final state is the only state reachable from the initial state in which there is a token in the final place: $\forall_M(i \longrightarrow^* M \wedge M \geq o) \Rightarrow M = o$
- *No dead transitions*: each transition contributes to at least one instance: $\forall_{t \in T} \exists_{M,M'} : i \longrightarrow^* M \longrightarrow^t M_0$
- *Transition participation*: each transition participates in at least one process instance that starts in the initial state and reaches the final state: $\forall_{t \in T} \exists_{M,M'} : (i \longrightarrow^* M \longrightarrow^t M_0 \longrightarrow^* o)$

An object-centered process model is *correct* if and only if its state-machinge $< (P \mid P') \cup A, T \mid T', C \mid C' >$ verifies *termination* and *no dead transitions* properties.

Note that these first and third properties imply the fourth property. Thus our approach, additionally, verifies *transition participation*.

Since it verifies *termination*, it does not allow deadlock behavior. Thus, it does not support *relaxed soundness*. But since transitions occurrence are dependent on their rule's conditions and actions, how termination is assured? Two possible answers exist: it is assumed that they will occur (in fact, this is what happens in activity-centered models when they attach an event to a transition), or, alternatively and more interesting, an additional criterion must be defined to assess if they can occur and, in that case, if in an independent or coupled way from other rules.

Since rules' conditions and actions essentially depend on traceable marking-changes and data-changes, the occurrence of those event-based control-flows in the object-centered modeling landscape is possible. Fig.8.11 exploits how these criteria are being depict and how they affect the satisfaction of the termination property.

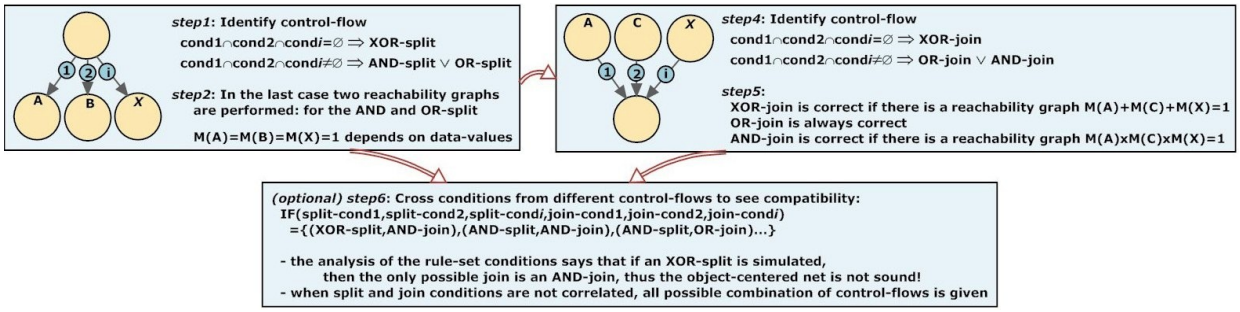


Figure 8.11: How rule-set models affect the soundness of object-centered models

By satisfying the third property we are also disallowing certain parts of the process not to participate in any process instances. However, this is not a problem as the objects' points of synchronization are only verified in the scope of their encapsulating object. By default, there is always an upper object.

Additionally, *proper termination* is too restrictive as it states that from each state reachable, the final state can be reached and that at this point in time there are no tokens left in the net. A example of how this may block life-cycles expressivity is depicted in Fig.8.12. Thus, object-centered modeling thus does not verify the second property and, consequently, does not also support *weak soundness*.

The study of lazy soundness deserves also some attention. Lazy soundness violates all properties as reaching the final state o, there might be additional tokens in the net, reflecting lazy activities, that will not terminate. Typically they result from the use of control-flow patterns as the *discriminator*, the *N-out-of-M join*, and the *multiple instances without synchronization*. Note that our approach can mimic the *discriminator* and the *N-out-of-M join* control-flow patterns. Using the *GF* scenario, an example would

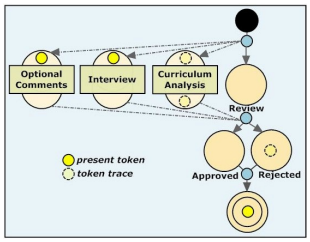


Figure 8.12: Transgress of *proper termination* criterion

be the selection of the first three (or randomly three) available *Reviews* for a *Request* proposal. Note, however, that when this happens the other related-executing instances, the rule mimic defining this behaviour, uses its action to force the tokens of the related-executing instances to terminate, and then clean them. This is also done when using the *nofi* function to create *multiple instances*, that no longer can be named instances without synchronization. Using this strategy, the object-centered modeling can verify the termination property even when using multiple instance creation and selection patterns.

Finally, since proper termination is not verified, object-centered modeling does not support all the four criteria and, therefore, it does not support *normal soundness*. Thus, what kind of soundness is this, that is neither normal, nor weak, nor relax nor lazy? This thesis names it *scoped-termination soundness*.

9

Modeling Transformations

Information is not knowledge.
– Albert Einstein

Through last chapters, the target object-centered modeling approach was derived. However, some questions remain open. How advanced soundness criteria and migration strategies can be defined for this modeling structure? How will object-centered process models be supported in practice?

Since the development from scratch of formal net theories and of a workflow-engine to support the target models are not viable at this stage of the research – such efforts would require a more mature study of its practical implications – this chapter defines a mapping between the object-centered process models and mature process models to demonstrate their verification and executability. Incrementally this chapter defines the mapping, first adopting plain *Yet Another Workflow Language* (YAWL), then the Proclets addition and, finally, a proposed multi-colored places plug-in.

9.1 Map to plain YAWL models

YAWL is a process modeling language developed by the Workflow Patterns Initiative [97] grounded on formal foundations, an extension of Petri Nets with an extensive list of additional workflow patterns using a small set of constructs. Additions, as the *newYAWL* [96] or *proclets* [2], have been including new control-flow patterns, resource perspectives and process decoupling techniques. Currently, YAWL only considers a single implementation and it is not yet being supported by a great number of tools, as BPEL standard language. Although several research exist on defining formal semantics to BPEL on the basis of Petri nets, process algebra and finite state machines, BPEL still fails in providing native support to workflow patterns, processes communication and human-oriented allocation and delegation [97].

The following properties of YAWL justify its choice:

- formal structure to study soundness properties;
- uniform basis that can be easily adapted to other notations such as the standardized BPMN;
- the proclets addition enables the definition of loosely-coupled communicating processes;
- expressiveness and coverage of control-flow patterns;
- native support for sub-processes (composition) and, thus, modular process models;
- executability, thus enabling the abstraction from workflow-engine specificities;
- non-local behaviour, where different process regions can be heterogeneously affected by a cancelling, skipping, suspending or submitting decision;
- notational convenience – direct arcs between transitions and behavior attached to transitions;
- handling of multiple instances patterns, with their number might not know at design-time;
- the execution semantics of YAWL has a good fit with state-based entities synchronization since it combines state-transition diagrams of process activities with Petri net markings;

Finally, since this research is introducing a new modeling paradigm, focus must be paid on the proof

of its effective execution instead on its compliance with standards that may limit modeling freedom.

A simplified definition of YAWL models, and the proposed mapping are presented below:

An *extended workflow net* is a tuple $< C, T, F, split, join, rem, nofi >$ [120], such that:

- C is a set of conditions, which must have a initial condition, $i \in C$, and a final condition, $o \in C$;
- T is a set of tasks, such that C and T are disjoint;
- $F \subseteq (C - \{o\} \times T) \cup (T \times C - \{i\}) \cup (T \times T)$ is a flow relation, with nodes in i to o path;
- $\{split, join\} : T \mapsto \{And, Xor, Or\}$ is a partial mapping that assigns the behavior to a task;
- $rem : T \mapsto P(T \cup C - \{i, o\})$ specifies the subnet cleansed when the task T is executed;
- $nofi = \mathbb{N}_{inf} \times \mathbb{N}_{inf} \times \{dynamic, static\}$ define the number of the task instances.

A simplified *YAWL workflow specification* is a tuple $< Q, top, T', map >$ [120], such that:

- Q is a set of extended workflow nets and $top \in Q$ is the top level workflow net;
- $T' = \cup_{N \in Q} T_N$ is the set of all tasks, such that the conditions and tasks of all extended workflow nets are disjoint, i.e., $N_1 \neq N_2 \implies (C_{N_1} \cup T_{N_1}) \cap (C_{N_2} \cup T_{N_2}) = \emptyset, \forall_{N_1, N_2 \in Q}$.
- $map : T' \mapsto Q - \{top\}$ maps each composite task ($t \in T'$) onto an extended workflow net.

Algoritm 9.1: Mapping of an Object-centered Process to a YAWL Process

Input: object-centered process model *object-pm*
Output: plain-yawl process model *yawl-pm*
if *is-sound(object-pm)* **then**
    1. *petri-net* = decompose-rules (*object-pm*);
    2. *yawl-pm-complex* = generate-control-flow-gateways (*petri-net*);
    3. *yawl-pm-simple* = simplify-paths (*yawl-pm-complex*);
    4. *yawl-pm-data* = enrich-data-access (*yawl-pm-simple*);
    5. *yawl-pm* = define-composition-frames (*yawl-pm-data*);
    if *advanced-sound(yawl-pm)* **then**
        return *yawl-pm*;
    throw sound-exception(*process model is not sound*);

Using an object-centered process model, an intermediary language-independent Petri net is derived, a set of control-flows derived based its rules, and this net simplified according to a set of techniques (see Annex 17). Fig.9.1 exploits graphically how the 2^{nd} and 3^{rd} steps are performed.

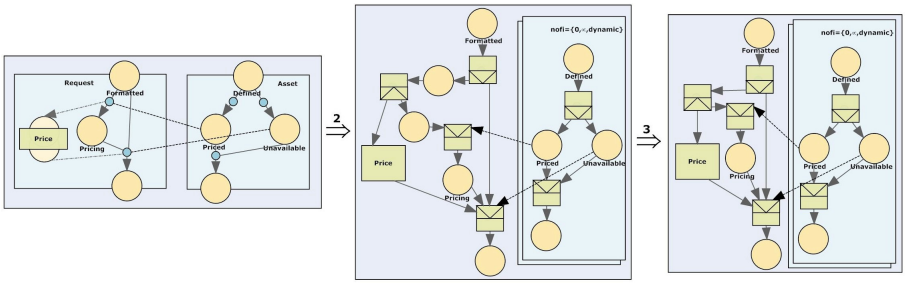


Figure 9.1: YAWL model generation and reduction

For the mapping of data-contexts, each compound data-objects is flatten into simple, sets and lists of data-attributes and their data-access link turned explicit for every YAWL activity recurring to the resources-view provided by the newYAWL [96] addition, using the *formal parameters* construct.

Finally, in the last step, the use of composition frames are defined when a subordinate-process is totally executed between two states of a superordinate-process. Note, however, that communicating

processes may establish bidirectional synchronization points or a third process block the possibility to use a composition frame, as depicted in Fig.9.2.

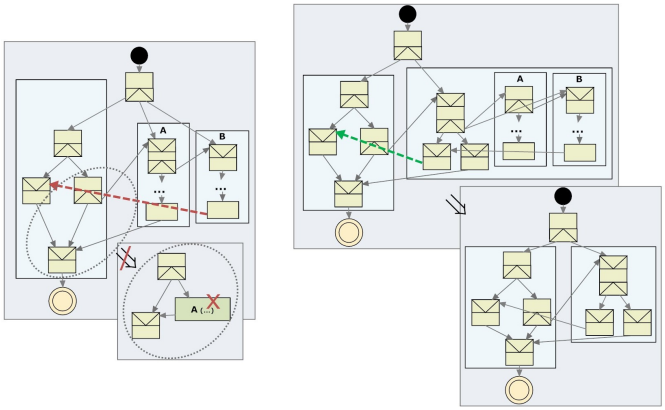


Figure 9.2: Composition frames for the generated YAWL models

The mapping to plain YAWL models presents two main drawbacks: *i)* decoupling intricately related process fragments, and *ii)* the inability to support advanced rules using existent control-flows – events may introduce time-state and data-values condition, although limit the ability to trace the source for those changes. Fig.9.3 presents the generated YAWL process model for a simplified *GF* process model.

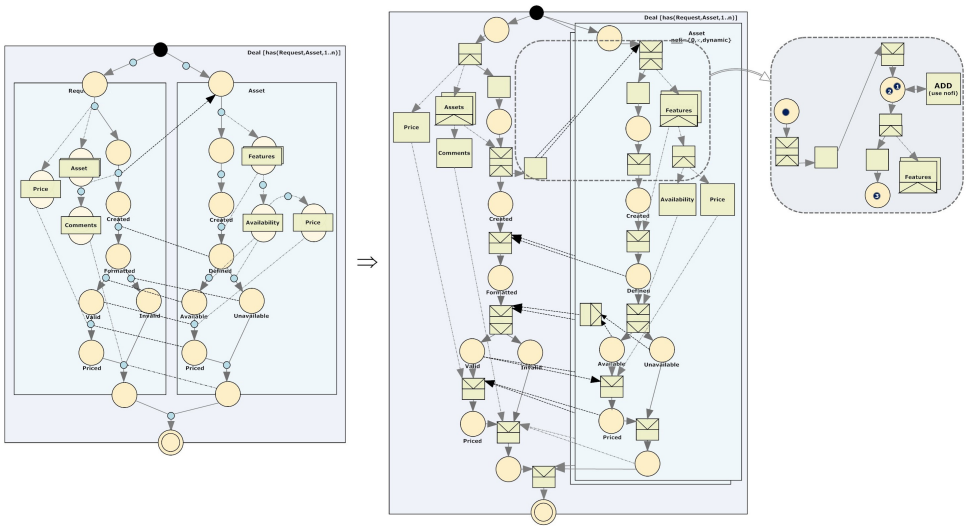


Figure 9.3: Default rule-set models dynamically generated

9.2 Map to proclets-enriched YAWL models

In order to surpass the limitations found when using plain-yawl processes, the proclets-addition will be used by our mapping. A simplified definition and mapping for this addition may be the following:

A *proclet class* is a tuple $< s, P, f, Cond >$, such that:

- s is an extended workflow net that models the proclet life-cycle;
- P is a set of port types, $<direction, cardinality, multiplicity> \in P$;
- $f : T \mapsto P$ assigns a set of life-cycle tasks to a set of ports;
- $Cond$ is a set of conditions that tasks connected to ports may have.

A simplified *proclet-based process model* is a tuple <PC,Ch,Perf,ns> where:

- PC is a set of proclet classes and Ch is a set of channels;
- $Perf, < id, pc \in PC, PC' \subseteq PC, type, content > \in Perf$, a set of performatives exchanged among ports;
- ns is a naming service for proclets dynamic discovery.

Algoritm 9.2: Mapping of an Object-centered Process to a Proclets Net

Input: object-centered process model *object-pm*
Output: proclets model *proclets-pm*
if *is-sound(object-pm)* **then**
1. *petri-net* = decompose-rules (*object-pm*);
2. *object-yawl-pm-complex* = generate-internal-control-flow-gateways (*petri-net*);
3. *object-yawl-pm-simple* = simplify-paths (*object-yawl-pm-complex*);
4. *object-yawl-pm-data* = enrich-data-access (*object-yawl-pm-simple*);
5. *plain-proclets-pm* = define-objects-communication (*object-yawl-pm-data*);
6. *compound-proclets-pm* = decouple-encapsulation-into-new-proclets (*plain-proclets-pm*);

if *advanced-sound(compound-proclets-pm)* **then**
return *compound-proclets-pm*;
throw sound-exception(*process model is not sound*);

First four steps are similar to the plain-yawl proclets mapping but limited to each internal life-cycle. The 5^{th} step defines the ports, channels and performatives needed to perform the communication.

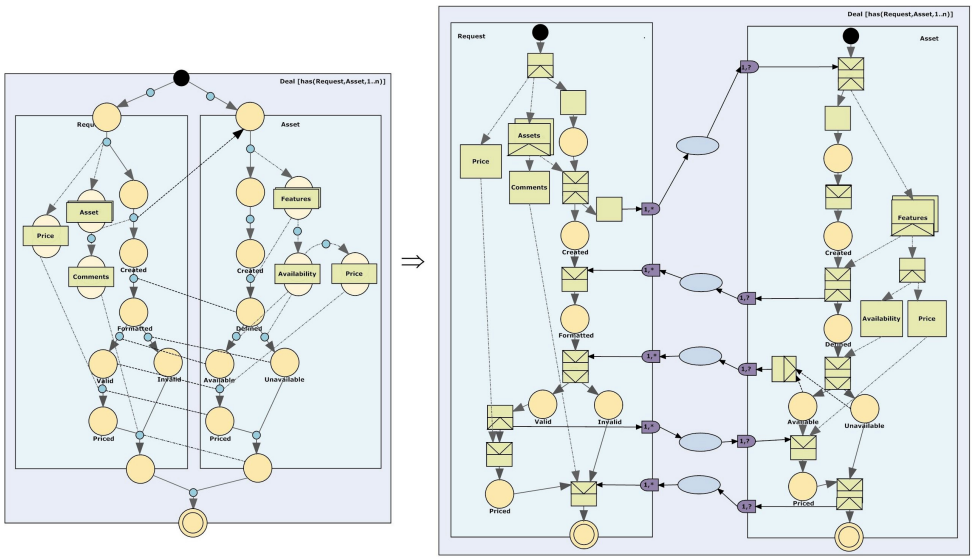


Figure 9.4: Derived proclets model from an object-centered process model

The proclets-addition is used to decouple communicating process models that, otherwise, would have to be in a plain form and, therefore, nor benefiting from the parallel execution achieved by assignment each object-based process to different workflow engines neither offer guidance to the process instances' scheduler. Fig.9.4 presents the generated proclets for a fragment of the *GF* process model.

Nevertheless, the encapsulation problem isn't still solved, since this addition does not support the

notion of compound proclets. In fact, since each proclet communicates through channels with other proclets, the understanding of how a proclet can encapsulate a set of related proclets is not trivial, although required to isolate communication dependencies. To solve this aspect, the notion of compound proclet is added, but implemented using the *mediator design pattern*. Fig.9.5 the mapping 6^{th} step – disaggregation of compound proclets into simple mediator proclets.

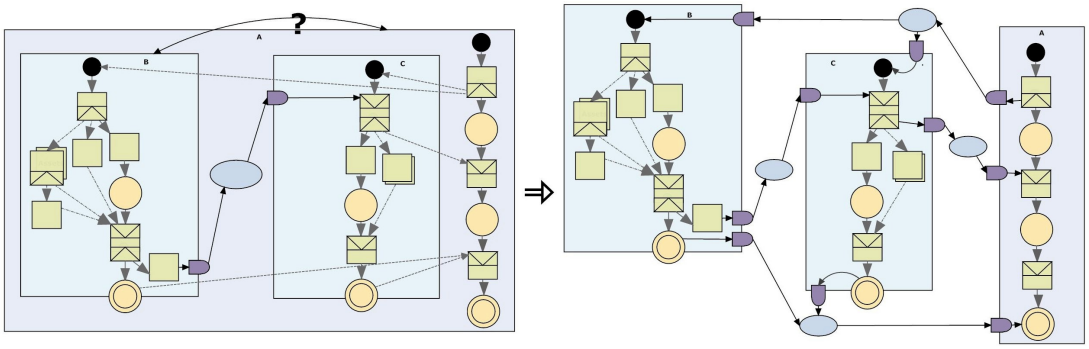


Figure 9.5: Solution to the proclet's encapsulation problem: definition of a proclet mediator

9.3 Multi-colored places addition

To guarantee the executability of the proposed advanced rules, basics from Colored Petri nets[1] must be recovered and used to enrich the existent proclets addition. *newYAWL* [96], through its resource view additions, provides data-enriched tokens and a new construct that supports color-based semantics – link conditions assigned to tasks' outgoing arcs.

The proposed solution assigns colors to net tokens, as depicted in Fig.9.6. The implementation uses multi-dimensional matrixes with identifiers that work as links to their instances' data-attributes. This data-access is constrained by objects-relation visibility and authorization levels (*section 7.1*).

Exemplifying, if a *Request* is related with a set of *Asset*s, each token from a *Request* instance has its own color and each token belonging to an *Asset* instance has a multi-coloring composed by the color of the related instances and its own color. If a set of *Asset* instances is correlated with different *Request* instances in a non-coupled way, simple-coloring functions for both process places are used. Finally, if a *Request* is related with a set of Suppliers and *Asset*s, and each *Supplier* responsible for different *Asset*s from multiple *Request*s, next coloring must be found: $\{C_{req}, Set<C_{asset}>, Set<C_{sup}>\}$ for *Request* places, $\{C_{asset}, C_{req}, Set<C_{sup}>\}$ for *Asset* places, and $\{C_{sup}, Set<C_{req}, Set<C_{asset}>>\}$ for *Supplier* places.

Remember that places exist between and within YAWL's tasks, thus the marking function in addition with its colors (serving as links to recover data) complete describes the system state and it is, therefore, the input for the system monitoring and optimization domains. Multi-coloring additionally enables an expressive textual or graphical environment for the management of multiple-instance patterns.

Fig.9.7 depicts two possible scenarios specifying: *i)* a point of synchronization to supply all the non-supplied assets belonging to the same customer, and *ii)* a point of synchronization to supply non-supplied assets for similar customers in terms of riskRating, which led to the use of the colored-links

[1] a *coloured Petri net* is a tuple $<\Sigma, P, T, A, N, C, G, E>$ such that: *i)* Σ is a finite set of types, called colour sets; *ii)* P is a set of places, T is a set of transitions, and A is a set of arc identifiers, such that $P \cup T = P \cup A = T \cup A = \emptyset$; *iii)* $N : A \rightarrow (P \times T) \cup (T \times P)$ maps each arc identifier to a pair ($start_{node}$,end_{node}) of the arc; *iv)* $C : P \rightarrow \Sigma$ is a colour function that associates each place with a colour set; *v)* $G : T \rightarrow BoolExp$ is a guard function that maps each transition to a predicate; and *vi)* $E : A \rightarrow Exp$ evaluates to a multi-set over the colour set of the place.

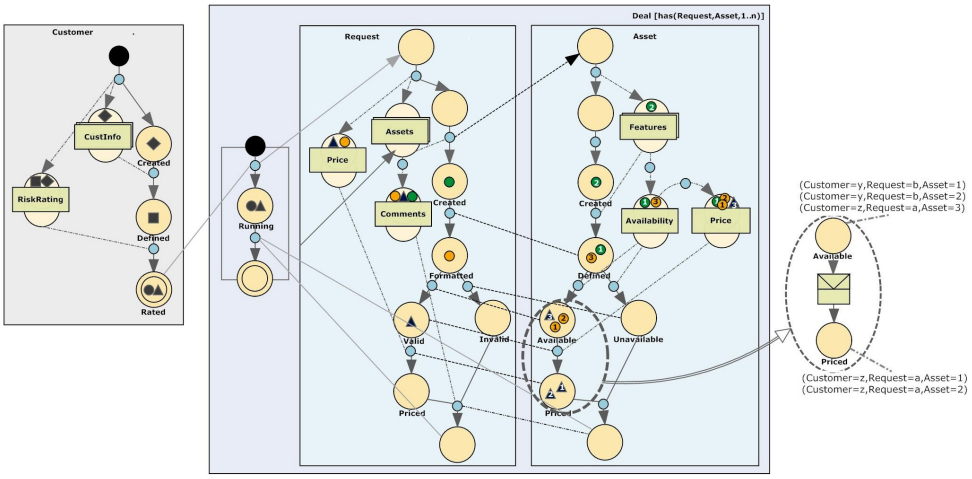


Figure 9.6: Multi-colored tokens

to retrieve data-attributes (which are, in fact, the tokens' colors as defined by Colored Petri nets [120]) when no mediators avoid the recovery of such attributes.

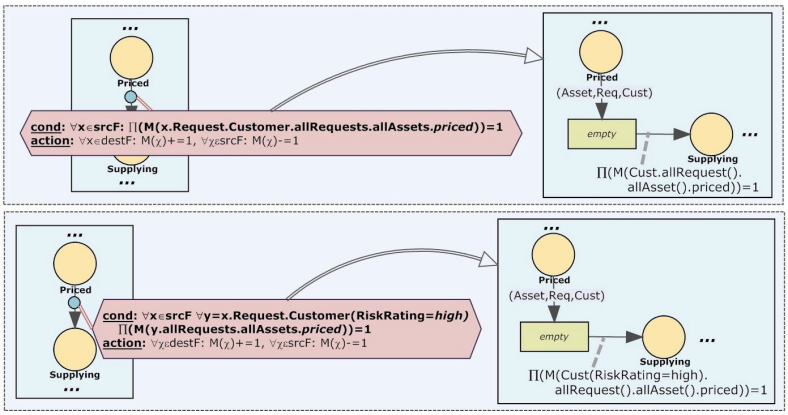


Figure 9.7: Default rule-set models dynamically generated

Concluding, these scenarios show a fragment of a proclets net solution enriched with multi-colored tokens recurring to *newYAWL*'s link conditions. Note additionally that the dynamic multiple instantiation of processes, *nofi*, is already support by YAWL, and the non-local behavior for the cancellation, undone or advanced behavior patterns, *effect* is not fully supported by YAWL but already implemented by newYAWL using the so-called *cleanset*, *forced-completion* and *arc-disablement* constructs.

Finally, the executability and advanced sound verification for the object-centered models are, therefore, dependent on the availability of a workflow engine that jointly supports *newYAWL* constructs (as presented in [96]) and the proclets-addition (still in development [112]).

V Validation

10

Proof of the Concept

Alice: Would you tell me, please, which way I ought to go from here?
The Cat: That depends a good deal on where you want to get to
Alice: (...) so long as I get somewhere.
The Cat: Oh, you're sure to do that, if only you walk long enough.
— Lewis Carroll, *Alice in Wonderland*

How was the mapping implemented in practice? How was the syntax of the object-centered models, newYAWL and proclets captured? How language transformations were applied? Was it possible to realize a complete mapping? If not, what are the implications? This chapter briefly introduces the implementation for the mapping function introduced in previous chapters.

The developed object-centered models can be considered a high-level domain-specific language to lower-level constructs provided by enriched YAWL models. Thus, the developed prototype was centered on model definitions and model-to-model transformations. The combination of *Syntax Definition Formalism* (SDF) with *Algebraic Specification Formalism* (ASF) [19] was chosen for this propose.

SDF models describe language syntax using context-free production rules. This rules generate parse trees. ASF take parse trees as input and by applying functions to their nodes produce re-written parse trees as output that can be unparsed to obtain an output text. ASF models are collections of algebraic equations to guide this term-rewriting. The following aspects justify the ASF+SDF choice [19][61]:

- high-level and modular formalisms for the analysis and transformation of formal meta-languages;
- concrete syntax for source code patterns turning easy and usable to transform languages;
- able to add type checking, formatting, fact extraction, and execution (run models with given input values);
- ability to specify lexical and context-free syntax with the option to define meta-variables for the concrete syntax patterns used in ASF to construct and deconstruct source code;
- automated parse tree traversal, keeps meta programs concise;
- static verification makes sure that code that can not be parsed is not generate;
- list matching, so any element can be accessed in a list without traversal;
- used in connection with graphical Meta-Environment tools to obtain an usable and integrated development environment;
- default equations and test, memo functions, lexical constructor and traversal functions, and robust ways to deal with layout and source code comments;

SDF was used to define the syntax of data models, object models, activity models, rules, object-centered process models, YAWL models, multi-colored link conditions and proclets net. The syntax of YAWL and some of its additions was translated from their official XML specification, courtesy of Queensland University of Technology. Some transcripts can be found in Annex 17.

ASF was used to implement the required transformations, and, among other utilities, the language

pretty-printer and the soundness-checker.

Two types of transformations must be distinguished: *i)* the set of transformations required to derive a sound and complete object-centered process models based on object and activity models that implement the high-level algorithms previously introduced or referred, and *ii)* the set of transformations required to map an object-centered process model in an enriched YAWL model. The length and complexity of these transformations turn their detailed presentation on this thesis an impossible task. However, some transcripts of these algebraic equations can be found in Annex 17.

The verification of sound properties is, presently, performed for object, activity and object-centered process models. Guided by the introduced soundness criteria the following properties are assessed. First, structural soundness is assessed by verifying if a tree (based on the input places and transitions) has *init* node as root and *final* nodes as leaves. Second, it defines a set of live and bound reachability graphs (so its analysis is not a NP-hard algorithm) based on rule-set conditions to verify the absence of dead transitions (automatically satisfied as the composition of the target synchronized objects is previously performed) and termination (final state is reachable from any node of the graph).

Some of the limitations encountered on this mapping include: *i)* the use of omissive quality behavior when defining the proclets' channels and the their ports static registry on the naming-service, and *ii)* the definition of assumptions when specifying the constructs defined by the resource-view of newYAWL as its complete syntax was not obtained. However, none of identified problems refute the argued executability of the target approach.

11

Practical Applicability

Meu devir fez-me, como Deus ao mundo.
– Fernando Pessoa, *Mensagem*

To assess the applicability of the developed modeling approach, this work uses a set of metrics applied in comparison with existent alternatives and uses the target approach to model three domain-specific scenarios: *i)* financing, *ii)* manufacturing, and *iii)* healthcare.

11.1 General Performance

To test the feasibility of the object-centered approach, an indicator composed by a set of weighted metrics was defined. The weighting of the different metrics was based on the degree of relevance attributes by empirical research [68][28][120][114], excepting null-weight metrics, which are metrics may requiring further practical evidence or users' inquiries. The performance indicator is simple, measurable and traceable since it is decomposable in more specific indicators. Note that while some of the bottom-line metrics are just booleans, others are formulas as the relative coverage of some aspects by comparison with the available alternatives. *Modeling Performance* = 0.3×*Modeling Usability* + 0.7 × $\frac{\Sigma_{i=1}^{5} Req_i}{5}$, with:

- *Modeling Usability* = 0.4×*Simplicity* (relative number of objects or process structures × [relative number of their intra- and inter-flow relations]^2 for each granular level) + 0.3×*Guidance* (0.15× expressive default behavior + 0.15×natural and compact specifications of advanced behavior) + 0.2×*Memorability-Learnability* (0.1×number of constructs + 0×improved time and errors for sequent similar scenarios to model) + 0×*Efficiency* (completion time of revolutionary and adaptive modeling + number of errors) + 0×*Satisfaction* (modeling experience grade) + 0.1×*Security* (0.1×qualitative verification criteria + 0×quantitative evaluation criteria);
- Req_1 = *Data-Access* = 0.3×*Modular Data-Contexts Specification* + 0.1×*Default Data-Access Criteria* + 0.3×*Authorization* + 0.3×*System's Data Accessible and Integrated in PM Landscape*
- Req_2 = *Data-Reaction* = 0.6×*Dynamic Response to Object's State-change* + 0.4×*Data-changes Traceability*;
- Req_3 = *Data-based Coordination* = 0.5×*Decoupled Interaction* + 0.3×*Data-centered Expressivity* (constraints coverage at the data level) + 0.1×*Advanced Behavior* + 0.1×*Derivation of Default Behavior*;
- Req_4 = *Granular Pliancy* = 0.2×*Atomicity* + 0.2×*Composition* + 0.3×*Criteria* + 0.3×*Zoom Abstractions*;
- Req_5 = *Data Modeling and Flexibility* = 0.4×*Expressive Modeling of Data at PM Level* + 0.4×*Flexibility by Change* (coverage of constructs + relax degree of migration strategies) + 0.1×*Flexibility by Defer* (placeholders role) + 0.1×*Flexibility by Deviation* (extent of PM regions supporting exceptions).

The results of the application of this indicator, applied to three different systems in financing, manufacturing and healthcare sector, are synthesized on Table 11.1. Although these results give a good insight of how the modeling approach performs, they do not assure its practical applicability as domain-specific aspects may reject the approach if not appropriately captures the domain constraints.

	Object-centered Modeling				Document-based Modeling	Artifact-centered Modeling	Product-based Modeling	Data-driven Modeling	Case Handling	Proclets
	financing	manufacturing	healthcare	average						
avg(nr. objects per granular level)	15	20	16	17	120	80	140	17	120	80
avg(nr. inter-relations per object)	3	6	3	4	6	5	7	5	3	5
simplicity	$(17*4^2)/(17*4^2)$=100%				6%	13%	4%	64%	25%	13%
guidance	0.5×100+0.5×80=90%				0%	50%	0%	40%	40%	50%
learnability	10%				80%	40%	60%	5%	90%	65%
weak-soundness	true				true	true	true	true	true	false
usability	40+27+2+10=79%				28%	38%	24%	49%	50%	33%
data-access	30+10+25+30=95%				50%	50%	30%	15%	100%	15%
data-reaction	60+40=100%				80%	100%	0%	60%	70%	60%
data-based coordination	50+30+10+10=100%				30%	55%	30%	85%	30%	55%
granular pliancy	20+20+30+30=100%				50%	20%	20%	70%	20%	0%
data modeling and flexibility	40+40=80%				40%	50%	40%	50%	40%	10%
overall performance	0.3×79+0.7×95=**90%**				**54%**	**50%**	**24%**	**48%**	**41%**	**30%**

Table 11.1: Results: Generic Performance of Data-aware Approaches (with an available implementation)

11.2 Sector-oriented Applicability

Different types of systems may additionally provide domain-specific challenges. Table11.2 collapse such domain-specific behavior per three different industries, which serve as additional indicators for the object-centered modeling evaluation.

11.2.1 Financing

Fig.11.1 presents a high-level view of the object models defined for the Global Financing case, introduced in *section 3.1*. This simplified illustration presents the multiplicities of the instances under relation, and the points of synchronization between their life-cycles, showing how object instances are created, evolving and archived. This is a clear example of how GF system can benefit from modeling its processes based on its data model using weakly-connected interacting light-weight chunks of work.

The definition of regional variations for the global standard can be achieved by: *i)* using inheritance specializations for the standardized object models, by *ii)* maintaining the object models' coordination points, but allowing adaptations on data-attributes, life-cycles and internal behavior, or by *iii)* defining several placeholders for the parts subjected to change. Note additionally, that a rule deviation sometimes suffices to capture short-period variations. All these strategies enable: *i)* the standard model to evolve in coupled way with its variations, and *ii)* models traceability to comply and cross region-specific models.

Default data-contexts and authorization levels (see section 7.1) are used to recover contextual and non-contextual data. The presented solution also enables the dynamic definition of many-to-many instance relations, with each process instance having their own natural life-time. Finally, in Fig.11.1 were also defined patterns for the aggregation of instances related by a common super-instance or by a data-similarity (e.g. the assets procurement, supplying and delivery, or the requests review and pricing).

11.2.2 Manufacturing

Fig.11.2 synthesizes some aspects of a possible solution for an automobile engineering company – a component-centered structure allowing the definition of loosely-coupled interacting objects progressing at their own-rate in an asynchronous way.

Sector	Modeling Challenges and Requirements
Financing [23]	Data-intensive sector hardly modeled by activity-centered approaches (*section 3.2*). Main requirements: **1.** use of global standards with disciplined regional *variations*; **2.** ability to recover *contextual* and *historical information* to review, evaluate and price requests; **3.** *aggregated execution* of related requests and reviews. Collective procurement, pricing, supplying and shipping of assets; **4.** data-grounded criteria to establish supplier's agreements and customer's contract negotiations; **5.** *long life-time* of processes (payments, shipping, returns) requires relaxed migration criteria and decoupling of processes having distinct life-cycles duration; **6.** *dynamic and multiple instantiation* of assets and reviews, each may triggering non-local effects; **7.** *many-to-many process relations* (e.g. requests may be handled by several customers and suppliers); **8.** *Complex, regulation-compliant and data-centered decisions* to approve and price requests.
Manufacturing [76][74]	The development of complex products, as cars or airplanes, requires the coordinated design, test and release of thousands of processes and inter-dependencies. Adaptations require profound process knowledge, high-efforts and, as impact hundreds of components and their synchronization, are error-prone and may can cause deadlocks blocking the execution of the whole process. Main requirements are: **1.** *dynamic adaptation* of these complex and large process structures (e.g. add or remove of a component as a navigation system); **2.** *multiple types* of processes (e.g. design, testing and release) executed for each component; **3.** *different life-cycle times* of the mechanical, software and hardware components; **4.** the strong linkage of the process structures with the assembly of the product; **5.** real-world *exceptions* (e.g. abnormal termination of a process) occur frequently and may affect the whole process structure (*non-local behavior*); **6.** *reuse* of processes for different productions as standards for development of components are increasingly driven by quality frameworks and engineering guidelines (e.g. testing for the speed sensor).
Healthcare [40][112]	Collaborative involvement, distributed roles, complex data transfer, non-integrated technologies and limited centralized control can lead to fail of providing the right care to right patient at the right time. Requirements: **1.** *collaborative modeling* and *assistance* since complexity and area-specific details often arise; **2.** capture of tacit knowledge, do not enforce structure for the medical staff activities; **3.** *history-based* input for decision-making; **4.** incremental and dynamic *improvement* of processes always capturing the who, how and why; **5.** *concurrency*, *sharing* and *up-to-date* view of processes. Avoid waitings for the end of a process or for obtaining of data, enable collective assignments and turn data-changes immediately visible; **6.** *multi-way communication* among teams and departments; **7.** *secure* development protecting the model under improvement from unauthorized external entities. Comply with *regulations*. Enable *simulations* to capture dynamic behavior; **8.** measurement and prevention of adverse events occurring during or after healthcare delivery; **9.** ability to create *high-level sketch* for similar unities with large degree of freedom to be concreted by each unit as the availability of resources and ratio of manual activities strongly affect unit processes.

Table 11.2: Domain-specific modeling requirements for different sectors

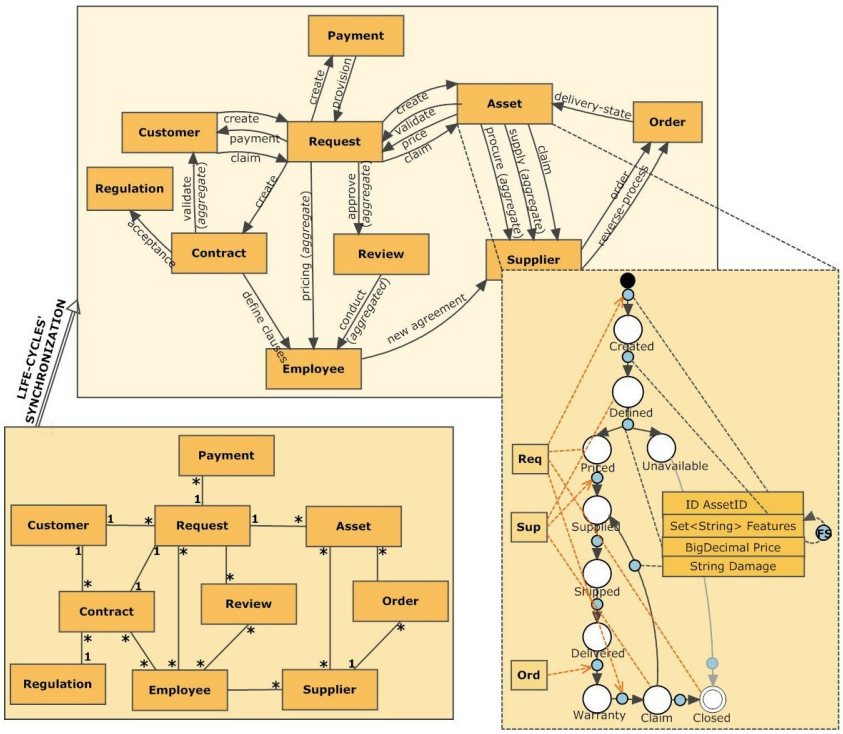


Figure 11.1: Object-centered Modeling of the *Global Financing* case

First, the use of the encapsulation mechanisms is of maximum importance on this sector. However, the sharing of components by different compound objects is an important issue not already assessed by the object-centered approach.

Second, the definition of template objects for the car components' and their dependencies is a relevant aspect, as their different specializations, as the modeling, versioning, development, integration, simulation and test, must respect those dependencies to deliver their output. Inheritance is also used to manage the logistics, employees, supplying assets, and technologies for each component. Note, however, that the notion of object-interface – not yet supported – would have a better fit for this problem.

Third, the capturing of real-world exceptions in components or on their integration and their non-local effects is managed by editing the default activity models.

Finally, this solution presents a natural basis for the reuse and dynamic adaptation of objects.

11.2.3 Healthcare

A compact view for a set of synchronized object models of an healthcare clinic is presented in Fig.11.3. As in other sectors, we see the importance of separating monolithic processes in data-driven object-centered structures. An equivalent complete network of activities would be too complex. Note that the management of visits, exams, analysis, consults, diagnosis, treatments and even the assets supplying (when decreasing below a threshold) is easily modeled in a data-centered and modular way since there is a natural isolation of aspects.

Different modelers can be assigned to each object, and collectively discuss their synchronization points. Also, as the object-centered landscape is oriented to data-attributes filling, there is a natural orientation towards tacit knowledge (e.g. use of form-fields as comments and suggestions, capture e-mail collaborations) rather than enforcement as the way a form-field is filled may not be specified.

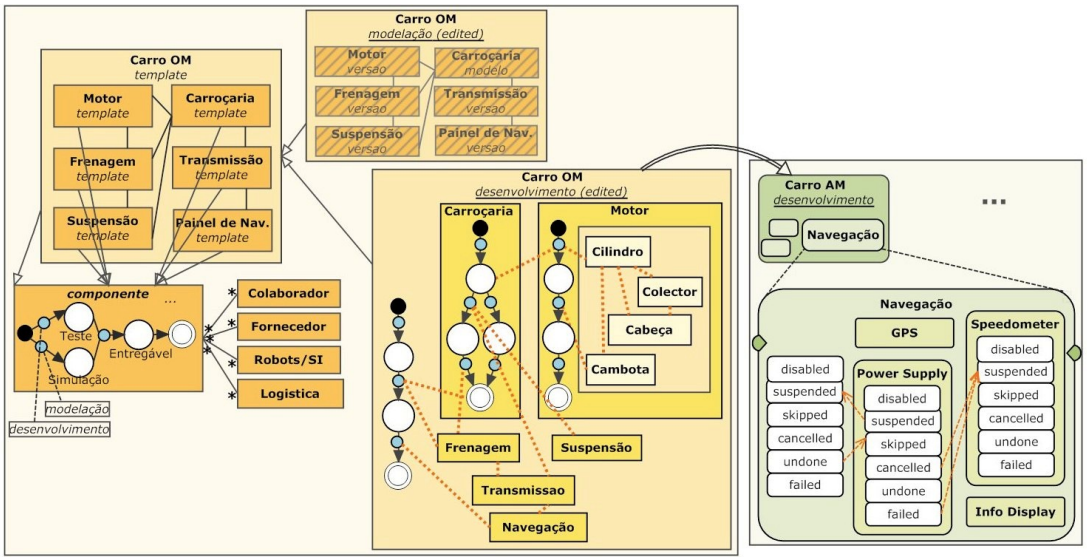


Figure 11.2: Object-centered Modeling of the *Car Enginneering* case

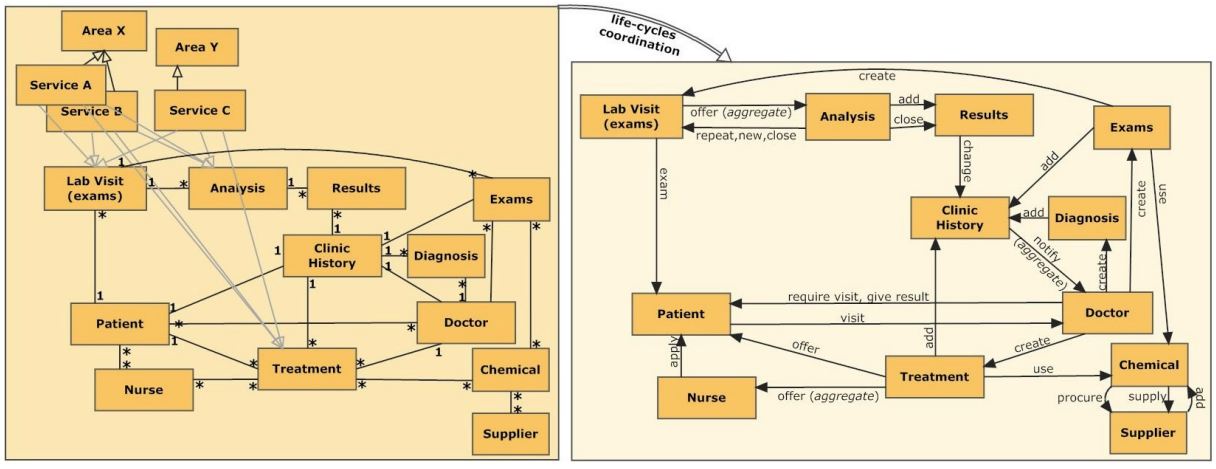


Figure 11.3: Object-centered Modeling of the *Clinic Diagnosis* case

Access to historical data is done through authorization, which must comply with privacy regulations. Authorization table must additionally provide a link to the author, changes and time of each change occurrence. Secure executions are also assured by the applying of the introduced soundness criteria.

The measurement and prevention for adverse events can be done recurring to complex event processing techniques as described in Annex 15.

Optimized data-centered concurrency, sharing and up-to-date view are supported aspects by the developed approach since all the system data is integrated in the process modeling landscape. Finally, since units differ from each other, they must define customizations for some general objects.

11.3 Comparison of the Approaches Performance

Results for a set of performance metrics were already presented in Table 5.7. Case-based analysis was done in previous section. Final aspects resulting from the confront of the developed object-centered modeling approach with its alternatives is done in Table 11.3.

Approach	Substitute?	Description
Document-based Modeling	no	Its customizable enactment and natural way to capture ad-hoc forms of collaboration turns this approach attractive to model healthcare systems. However, since neither supports dynamic adaptation, multi-way communication, nor complex processing, it ends being used for data capturing and tracking proposes as one in many technologies. Its inability to support multiple instance patterns, advanced behavior, specialization and complex data structures turns him a bad candidate for other sectors;
Artifact-centric Modeling	*financing*	Its business orientation, modularity and ability to synchronize multiple instances turns him a good candidate for financing, where practical evidence was already studied [26]. However, it neither offers support for composition nor for vertical aggregation. Since it does not support encapsulation, non-local behavior and data-access methods, it is not appropriate to model manufacturing and healthcare systems;
Product-based Modeling	no	The poor overall performance obtained by this approach and its already introduced limitations (section 5.1.4) makes this approach a non-potential substitute for the object-centered modeling. However, it is the only using quality attributes to affect dynamic path choice, which turns him feasible for the development of some components in the manufacturing sector that may depend on run-time metrics and probabilistic calculus;
Data-driven Coordination	*engineering*	This approach was developed centered on the manufacturing background, where its application was successfully applies [74]. Despite its ability to define expressive interactions between objects, including composition, it is not usable and does not support complex event processing techniques and non-local behavior to deal with manufacturing exceptions. Its inability to model data, required to define contexts and affect either simple or behavior exclude this choice to model systems in other sectors;
Case Handling	no	Although appointed as a potential choice to model healthcare systems due to its natural form-field orientation based on required data, this approach still has to mature or to be applied jointly with other approaches that support multiple-instance interaction patterns;
Proclets	*healthcare*	Proclets fits particularly with the modeling of healthcare systems, although the absence of a final implementation does not fully support this observation. The ability to synchronize proclets, using different applications and platforms, enables the integration required to harmonise the different hospitalar systems. However this fact does not solve its data-awareness limitations, which constrains further applicability;
Object-aware Modeling	*financing* and *healthcare*	Object-aware has a good fit with financing and healthcare since: *i)* it integrates best-practices from artifact-centric and case-handling, granting business orientation and decoupled interaction, and since *ii)* it adds process execution flexibility, form-centered with vertical authorization access (particularly fitting the tacit collaborations and security requirements for healthcare) and batch-driven collective enactment of activities. However, advanced coordination patterns as the vertical aggregation of instances that do not share the same higher-level objects remain crucial for financing and soundness verifications are increasingly demanded for healthcare. Additionally, the object-aware inability to support default and non-local behavior, event-driven exceptional behavior, customization and encapsulation may limit its success on the engineering domain.

Table 11.3: Discussion of the Results for Data-aware Approaches in comparison with Object-centered Modeling

12

Hypothesis Validation

As above, so below.
– Hermes Trismegistus, *Emerald Tablet*

With the understanding of how the object-centered modeling may perform for different real scenarios, it is crucial to synthesize its main business implications and how they affect the ability to evolve data-intensive system's models. Finally, a set of theorems will be derived in order to approve or reject the thesis statement – *object-centered models support the continuous improvement of data-intensive processes*.

12.1 System Implications

Table 12.1 presents a set of system implications of the adoption of the object-centered modeling.

In order to validate our hypothesis, it is required to understand how these implications affect the evolution of data-intensive or object-centered process models. *First*, the modularity of objects in addition to the proposed object-oriented patterns as encapsulation or inheritance enable that changes are performed locally to a set of objects with minimal impact to the others. State-based synchronization when viewed as an interface for interaction with other objects also fosters the ability to evolve models.

Second, its compliance with an incremental and iterative way of modeling, turns it attractive for changes seeking the completion, optimization and expansion of their models. Additionally its rapid prototyping fosters its fit with systems under revolutionary changes, requiring new models periodically.

Third, the capturing of tacit communications, the tracing of events and the measuring of time-ranges between object states or milestones (that may not correspond to an activity conclusion) improve the feedback required to suggest, enforce or guide new changes.

Fourth, the object-centered transformations, default behavior, data-based access and granularity criteria, and complex event processing techniques (fostering abstraction from low-level aspects and extraordinary patterns) turn the evolution of models a simple, guided and focused task.

Fifth, the ability to: *i)* dynamically change models, through the creation, edition and removal of objects and of their inter-synchronization points, to *ii)* not fully specify objects or regions of their life-cycles, and to *iii)* deviate from object models either using exceptional-behavior or creating specializations for a specific set of instances, foster the ability of object-centered models to evolve.

Sixth, the natural parallelization, dynamic reaction to events and objects' loosely-coupled interaction, turn models flexible by design, as process progress dynamically affects its constraints and agents are free to fill form-fields in multiple ways, and flexible by change, as model adaptations do not require the redesign of paths of activities but are expressive and local.

Finally, the argued method of modeling object-centered process models that starts with the expressive enrichment of data models by implicitly capturing system constraints at the object modeling level, with part of its behavioral aspects being dynamically derived and encapsulated in activity models, turns

Positive Implications	▹ improved communication, as it spans multiples silos and provides common vocabulary for stakeholders; ▹ natural alignment with system goals, as life-cycle definition and coordination is driven by operating milestones; ▹ rapid prototype generation since objects easily capture the dynamics of system key entities, providing enough semantic information to deliver an application almost from the first design iteration; ▹ incremental development expressing necessary changes and increasing confidence on the design by adding new objects and refining their data-attributes and life-cycles (growth in scope or in details); ▹ the supported object-oriented patterns foster a new modeling paradigm with similar implications as the ones observed for programming proposes – initial higher costs but further savings as learnability curve rises; ▹ source-traceable events and soundness enforcement ensuring new correctness criteria; ▹ the generation of default behaviour and data-centered guiding criteria to support the modeling task; ▹ since processes are factored around loosely-coupled interacting life-cycles, there is an increased scope for reuse, agility of modeling as changes become local to process fragments – highly flexible fragments may encapsulate changes with minimal impact to a stable base that anchors top-level objects; ▹ unified basis to gather up-front the system requirements and rules decreasing the need of re-factorizing and, consequently, ripple downstream, deployment and maintenance costs; ▹ decreased risk and costly adaptations caused by integrating process and data concerns; ▹ framework to plug key aspects as agent-allocation, goals-measuring, and many other requirements (e.g. rules, user interface, process variations, key performance indicators, low-level logic) centered on data; ▹ for large processes, starting the modeling with a few key objects provides a very early insight, hardly achieved recurring to activity-centered multi-level abstractions (that can still be mimic using object mediation) [15]; ▹ modeling at an arbitrary level of abstraction by using new behavioural patterns based on complex event processing techniques that aggregate chunks of low-level events into new coarser events; ▹ capture of tacit communication through by incorporate model collaborative platforms as objects and feed them; ▹ the increased traceability achieved by integrate data and advanced behavior at the process modeling level improve process monitoring and, consequently, business intelligence activities;
Challenging Implications	▹ since it is a disruptive modeling approach, the data-pushed way to present activities in forms may decrease the agents system-dynamics awareness as they may not be able to position each activity in a fully prescriptive path; ▹ loosely interacting processes may enforce patterns for agents collaboration (objects may represent a team or a role). The cross-functional nature of end-to-end processes turns the arrangement of system agents less prescriptive; ▹ the increasing of parallelism and the absence of a quality-driven path choice by the process modeling landscape turn the ordering execution of activities dependent on agents ability to decide and may disperse them; ▹ the new mindset may bring resistance and challenges to process modeling for system models' stakeholders; ▹ object-centered modeling cannot be consider a what-you-see-is-what-you-get approach, as different types of models are used, default behaviour is automatically added, and transformations applied to derive complete models; ▹ the increased entanglement among event, data and process modeling roles may trigger resistance; ▹ ordering of activities may be totally unknown before the process derivation and execution as constraints are implicitly grounded and dynamically react on data conditions. However, this is threat as long as modelers are biased by the need to see networks of activities, instead of thinking on what essential dependencies underlie such network; ▹ data integration at the process modeling level may implicate changes to the existing applications, as their recodification to use authorization levels instead of direct access or the use of adaptors to mimic the data-presence; ▹ the need to define states in addition to data-driven activities increase the complexity of the models as traditional nets are in essence a constrained composition of activities that pushed milestones or places to background.

Table 12.1: Implications of the use of the object-centered approach to model data-intensive systems

the evolution of processes possible with the evolution of these enriched data models or object models. As it was initially argued, in data-intensive systems, this coupled evolution is the way to assure the consistency of informational and functional perspectives and to turn modeling a flexible task.

12.2 Hypothesis Approval

In order to assess if *object-centered models support the continuous improvement of data-intensive processes*, five requirements were retrieved. Centered on these requirements, the following theorems testify the validation or approval of the thesis statement:

- theoretical and empirical-guided study of existing data-aware modeling approaches shows that none of them successfully support the five hypothesis-driven requirements;
- the consistent and coherent coexistence of multiple principles retrieved to satisfy each requirement were argued possible in an object-activity-goal-time state-based synchronization basis;
- a *proof-of-the-concept* was successfully developed, proving the executability and soundness of the properties for a derived modeling approach that obeys to the introduced principles as it implements the object-centered transformations and the mapping to an enriched YAWL net;
- *metrics* were defined to evaluate each requirement, evidencing the object-centered modeling intrinsic ability to satisfy them without hurting the modeling usability, and an overall performance result that is positively demarcated in comparison with its alternatives (+30% than 2^{nd} best choice);
- *practical applicability* was assessed by applying the target approach to three distinct domains, which demonstrate its constructs coverage and its natural fit to deal with domain-specific requirements when compared with its alternatives;
- a set of *implications* resulting from the adoption of this new modeling paradigm was retrieved, and its role on fostering the evolution of data-intensive processes was studied, presenting benefits from expressively enriched and adaptive data models not seized by existing data-aware alternatives and that disrupt traditional modeling landscape.

Further research and practical evidence are required to confront these resulting theorems.

VI Concluding Remarks

13

Conclusion

Simplicity before understanding is simplistic;
simplicity after understanding is simple.
Edward De Bono

A set of concluding remarks can be retrieved from the previous study:

- Activity-centered approaches are limited in modeling data-intensive processes. Such limitations can be translated into a set of requirements. These requirements foster the ability to evolve process models in data-intensive landscapes.
- Emergent object-centered approaches do not successfully satisfy all requirements. This observation may correlate with their limited practical applicability and impact. Such approaches provide important principles to satisfy the introduced data-related requirements.
- Principles to deal with these different requirements are not mutually exclusive in an event-driven solution basis centered on the state-based synchronization of activity, object and goal models.
- An improved responsiveness to events can be achieved by decoupling the traditional notion of processes into interacting objects, since it discourages a thinking on constrained paths of activities and fosters a natural parallelization of the derived activities.
- The use of object-oriented patterns as inheritance, encapsulation and communication, and the specification of dependencies at the data level (implicitly defining local ordering constraints) enable an expressive derivation of process models centered on object models. Additionally, default data-access contexts, default synchronization rules based on objects relations and data-centered criteria to assure the atomicity and composition of object models offer modeling guidance.
- The enrichment of object models and their alignment with activity models, fosters an evolution of process models centered on object models adaptation, since they can expressively capture the system elements coordination and activity models dynamically evolve in a coupled manner.
- Orthogonally, agents can be dynamically added, removed or changing since: *i)* their interaction with the object-centered landscape is standardized through flexible and dynamically changing forms (either filled manually or event-pushed), *ii)* their allocation is based on declarative rules, and *iii)* they do not hide critical data for the modeling environment since they use authorization levels and activities' data-contexts to perform their actions.
- Formal algebraic equations, or model-to-model transformations, were used to derive complete and sound process models based on the governed object-centered process models.
- Executability of the target process models was proven by mapping them into YAWL models enriched with resource constructs and with the proclets addition.
- Finally, this work conducted a studied based on a set of indicators applied to the developed approach in comparison with other emergent object-centered approaches, that presents a novel practical applicability with potential in the financial, manufacturing and health-care domains.

The contribution of this thesis to the process modeling research community centered on the role data within processes can be synthesized in three fronts. First, the presentation of an effort towards a uniform

conceptualization of the object-centered modeling universe of discourse. Second, the definition of a modular, loosely-coupled, traceable and sound event-driven solution for the modeling of data-intensive systems where principles can be integrated in a solid way. Finally, the proposal of a new object-oriented paradigm, leveraged on practical applicability, to model and evolve data-intensive processes centered on the definition and agile adaptation of expressively enriched data models.

By extending the existing contributions on how to model and evolve systems in data-pushed landscapes, this thesis shortens the distance from the Utopian real-time responsive and auditable system by simply enhancing the role of data within processes, whose potential was always there, waiting for a disclosure able to reveal the simplistic nature of data-intensive processes.

14

Future Work

Possible lines of thought for future research may include:

- the object-centered modeling fit with *human-centered processes* [101][7]. Two directions must be explored. First, how adapters or a layer responsible to trigger normalized events may be added in top of collaborative applications, in order to trace e-mail discussions or collective decision-making processes. Second, since every relevant activity results in a data-attribute, how non-written human-interaction can be captured by records without degrading the process agility. Here, the every implication in auditability or flexibility for the object-centered approach must be carefully detailed;
- the *implementation* of the approach either as an addition to YAWL or following another strategy (from scratch, based on COREPRO framework or on the top of the artifact-centric approach);
- the role of *goals* in the object-centered modeling. How to capture and to achieve object-driven system goals? Since it is essential to avoid getting distracted by the activity-centered current modus system operandi, a goal-driven definition of object-centered models plays a decisive role. Thus, the relation between goal models and object models must be studied in detail. How goals can be formulated based on the system productions? How to measure their progress? How to assure that object-based goals are coherent and consistent? How they affect data-access? How object models evolve when goal models evolve? Does it make sense to define an expressive link between them that fosters some sort of dynamic coupled evolution? How the attachment of semantics to objects can dynamically affect their composition, constraints and comply with the system goals?
- the continuously systematization and enrichment of *rule-set models expressivity*. The definition of a clear syntax (e.g. build on top of existing formal languages [43][31] that already support a broad range of constructors), the inclusion of advanced time-state expression or executable-code additions, and the exploitation of new and more declarative ways to represent life-cycles [55];
- the *methods and patterns* used for the objects finding, clustering and relation, according to different types of systems. In *concrete* methods, existing data domain is analyzed to come up with the logical view in a bottom-up fashion. In *conceptual* methods, the system goals are first understood to come up in a top-down way with key objects to the system operation, which are containers for low-level internal objects [26]. Orthogonally, the modeling of objects can either be firstly focused on interaction (event-driven dependencies) by fragmenting the system in a set of synchronized incomplete objects, or firstly focused on their self-completion and then on their interplay concretion;
- the conception of an *hybrid approach for heterogeneous systems* where object-centered modeling plays its part. Here three artifacts are important: *i)* a road-map that helps to identify to which systems an object-centered approach may have a better fit than activity-centered approaches, *ii)* techniques to decompose a system in a set of subsystems in a way that each subsystem has a clear orientation (either to data or to tasks) and that such division is coherent and consistent with the system operation (e.g. departments within an enterprise), and *iii)* a new modeling approach that governs the interaction of processes resulting from these two different modeling approaches. Lines of thought analyzed during this thesis must be seized, as the virtual transit of objects among subsystems, either evolving through the filling of their data-attributes in data-intensive subsystems or evolving

by being accessed and manipulated by task-based activities in service-oriented subsystems;

- the development of a usable *graphical layer* on top of the existing textual models. How modeling constructors should be presented to the user? This topic involves the need to exploit the profile and skills of the object-centered modelers, how they differ from traditional modelers and, based on this information, to trace new ways to improve the usability of the target approach, in particular, with respect to the definition of advanced rules (e.g. visual way to depict points of synchronization, the aggregation of related instances, etc.);
- the improvement of the developed *domain-specific language*, and, in particular, the study of how to reduce its domain-coverage in order to give more guidance on the modeling task in specific domains as the healthcare, manufacturing, financial or human-centered;
- the implications on different process management fields (see annex 16.1) – e.g. how new object-driven mining patterns and techniques may affect process monitoring?
- the shift on the *user roles* for system modeling. Since data, event and process management are coherently bridged through the object-centered modeling environment, the implications of the collapse of these three roles – data, events and process modeler – into one unique role must be studied in terms of the required skills, resistance and of the effects resulting from the possible removal of coordination conflicts among the data, events and process management teams (reported as the central aspect for the failure on the management of data-intensive processes in [91]);
- the implications to the book of knowledge on process management, which may require open discussions with the theoretical and practical community to define an accepted *ontology* for the proposed concepts and their fit with existing way of modeling;
- the role of *exceptions* in the object-centered landscape. Since the improvement of a system operation is not only done through cycles of exploitation or optimization, but also through cycles of exploration that may require deviations or the definition of incomplete processes, it must be clearly assessed how the target approach can cover both scenarios without the need of using separated models. The definition of object-centered relaxed compliance criteria to migrate complying running instances must be present (note that system objects life-time may vary from seconds to months);
- how the natural event-orientation of the target benefit from *complex event processing techniques*, as the filtering or aggregation of lower-level events (e.g. a passenger entry, a mail reception) into new higher-level events affecting instances progress. This research direction it is centered on the implementation of advanced rules, and may determine the ability to specify object models at a natural coarse-grained level of abstraction;
- the assessment, through real-case applications, of the practical *feasibility* of: the data and application layers decoupling, of the definition of relevant tasks as form-field attributes, and of the defaults effects defined for suspension, cancellation, skipping and re-execution of activities;
- the retrieval of requirements for the support of *polymorphism* for the process inheritance pattern, so specializations can be dynamically adopted without requiring the definition of alternative paths;
- the deep review on how object-centered *soundness criteria* centered on reachability graphs (found in polynomial-time) are affected by data-conditions and by rules' actions marking-changes.

VII

Bibliography

Bibliography

[1] Petri C. A. Communication with automata (phd thesis). Technical report, Universit at Bonn, Institut fur Instrumentelle Mathematik, January 1966.

[2] Wil M. P. van der Aalst, Paulo Barthelmess, Clarence A. Ellis, and Jacques Wainer. Workflow modeling using proclets. In *CooplS '02*, pages 198–209, London, UK, 2000. Springer-Verlag.

[3] Wil M.P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: A new paradigm for business process support. *Data and Knowledge Engineering*, 53:2005, 2005.

[4] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Modeling and verifying active xml artifacts. *IEEE Data Eng. Bull.*, 32(3):10–15, 2009.

[5] Alan S. Abrahams and David M. Eyers. Using annotated policy documents as a user interface for process management. In *ICAS '07*, page 64, Washington, DC, USA, 2007. IEEE Computer Society.

[6] Russell L. Ackoff. Towards a system of systems concepts. *Management Science*, 17(11), 1971.

[7] ActionBase. Actionbase. http://www.actionbase.com/ (accessed June 1, 2009), 2009.

[8] Alessandra Agostini and Giorgio De Michelis. Improving flexibility of workflow management systems. In *BPM, Models, Techniques, and Empirical Studies*, pages 218–234, London, UK, 2000. Springer-Verlag.

[9] Pallas Athena. *Case Handling with FLOWer: Beyond workflow*. Pallas Athena BV, Apeldoorn, The Netherlands, 2002.

[10] Hyerim Bae and Yeongho Kim. A document-process association model for workflow management. *Comput. Ind.*, 47(2):139–154, 2002.

[11] Albert-Laszlo Barabasi. The architecture of complexity: the structure and the dynamics of networks, from the web to the cell. In *KDD '05*, pages 3–3, New York, NY, USA, 2005. ACM.

[12] Paulo Barthelmess and Jacques Wainer. Workflow systems: a few definitions and a few suggestions. In *COCS '95*, pages 138–147, New York, NY, USA, 1995. ACM.

[13] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. In *VLDB '05*, pages 613–624. VLDB Endowment, 2005.

[14] U. Bestfleisch, J. Herbst, and M. U. Reichert. Requirements for the workflow-based support of release management processes in the automotive sector. In *ECEC'05*, pages 130–134, April 2005.

[15] K. Bhattacharya, N. S. Caswell, S. Kumaran, A. Nigam, and F. Y. Wu. Artifact-centered operational modeling: lessons from customer engagements. *IBM Syst. J.*, 46(4):703–721, 2007.

[16] Kamal Bhattacharya, Cagdas Evren Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 288–304. Springer, 2007.

[17] Ilia Bider. Choosing approach to business process modeling - practical perspective. In *Inconcept*, january 2005.

[18] R.W.H. Bons, R.M. Lee, R.W. Wagenaar, and C.D. Wrigley. Modelling inter-organizational trade using documentary petri nets. *Hawaii International Conference on System Sciences*, 0:189, 1995.

[19] M. Brand, P. Klint, and J. Vinju. The language specification formalism asf+sdf. Technical report, 2008.

[20] Eric D. Browne, Michael Schrefl, and James R. Warren. A two tier, goal-driven workflow model for the healthcare domain. In *ICEIS (3)*, pages 32–39, 2003.

[21] Artur Caetano, António Rito Silva, and José M. Tribolet. Using roles and business objects to model and understand business processes. In Hisham Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *SAC*, pages 1308–1313. ACM, 2005.

[22] Artur Caetano, António Rito Silva, and José M. Tribolet. A role-based enterprise architecture framework. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 253–258. ACM, 2009.

[23] Tian Chao, David Cohn, Adrian Flatgard, Sandy Hahn, Mark Linehan, Prabir Nandi, Anil Nigam, Florian Pinel, John Vergo, and Frederick Wu. Artifact-based transformation of ibm global financing. In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo Reijers, editors, *Business Process Management*, volume 5701 of *Lecture Notes in Computer Science*, pages 261–277. Springer Berlin / Heidelberg, 2009.

[24] Peter Checkland and Sue Holwell. *Information, Systems and Information Systems: making sense of the field*. Wiley, Chichester, UK, 1998.

[25] Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Trans. Softw. Eng.*, 29(9):796–810, 2003.

[26] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009.

[27] F. Corradini, A. Polzonetti, R. Pruno, and L. Forastieri. Document exchange methodology for collaborative work in e-government. In *DEXA '06*, pages 283–287, Washington, DC, USA, 2006. IEEE Computer Society.

[28] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Commun. ACM*, 35(9):75–90, 1992.

[29] Feriel Daoudi and Selmin Nurcan. A benchmarking framework for methods to design flexible business processes. *Software Process: Improvement and Practice*, 12(1):51–63, 2007.

[30] Thomas H. Davenport. *Process Innovation – Reengineering Work through Information Technology*. Harvard Business School Press, 1993.

[31] Stéphane Demri and Deepak D'Souza. An automata-theoretic approach to constraint ltl. *Inf. Comput.*, 205(3):380–415, 2007.

[32] Jan Dietz and Jan Hoogervorst. Enterprise ontology and enterprise architecture, how to let them evolve into effective complementary notions. *GEAO Journal of Enterprise Architecture*, 2, 2007.

[33] Jan L. G. Dietz. *Architecture - Building strategy into design*. Academic Service, The Hague, Netherlands, 2008.

[34] Document. Workflow management coalition terminology glossary, 1999.

[35] Dulce Domingos, António Rito Silva, and Pedro Veiga. Workflow access control from a business perspective. In *ICEIS (3)*, pages 18–25, 2004.

[36] Dov Dori. Object-process methodology as a business-process modelling tool. In *ECIS*, 2000.

[37] P. Dourish, K. Edwards, J. Howell, A. LaMarca, J. Lamping, K. Petersen, M. Salisbury, D. Terry, and J. Thornton. A programming model for active documents. In *UIST '00*, pages 41–50, New York, NY, USA, 2000. ACM.

[38] Gregor Engels and Luuk Groenewegen. Towards team-automata-driven object-oriented collaborative work. In *Formal and Natural Computing*, pages 257–276, 2002.

[39] Martin Fowler. Domain specific languages. http://www.martinfowler.com/bliki/DomainSpecificLanguage.html (accessed June 1, 2009), 2009.

[40] J. Framinan, C. Parra, M. Montes, and P. Perez. Collaborative healthcare process modelling: A case study. In Luis Camarinha-Matos, Hamideh Afsarmanesh, and Angel Ortiz, editors, *Collaborative Networks and Their Breeding Environments*, volume 186 of *IFIP*, pages 395–402. Springer Boston, 2005.

[41] C. Fritz, R. Hull, and J. Su. Automatic construction of simple artifact-based business processes. In *ICDT '09*, pages 225–238, New York, NY, USA, 2009. ACM.

[42] Raghu Garud, Sanjay Jain, and Philipp Tuertscher. Incomplete by design and designing for incompleteness. *Organization Studies - SAGE Publications*, 29(03):351–371, 2008.

[43] Cagdas Gerede and Jianwen Su. Specification and verification of artifact behaviors in business process models. In Bernd Kramer, Kwei-Jay Lin, and Priya Narasimhan, editors, *Service-Oriented Computing ICSOC 2007*, volume 4749 of *Lecture Notes in Computer Science*, pages 181–192. Springer Berlin / Heidelberg, 2010.

[44] Willy F. Gochet and Manfred W. Padberg. The Triangular E-Model of Chance-Constrained Programming with Stochastic A-Matrix. *Management Science*, 20(9):1284–1291, 1974.

[45] E. C. Goodman. Records management as an information management discipline – a case study from smithkline beecham pharmaceuticals. *International Journal of Information Management*, 14(2):134–143, 1994.

[46] F. Gottschalk, W.M.P. van der Aalst, M.H Jansen-Vullers, and M. La Rosa. Configurable workflow models. *International Journal of Cooperative Information Systems*, pages 223–255, 2008.

[47] Object Management Group. Business process modeling notation specification (omg final adopted specification), February 2006.

[48] The Object Management Group. Semantics of business vocabulary and business rules. Technical report, The Object Management Group, 2008.

[49] Adnene Guabtni and Francois Charoy. Multiple instantiation in a dynamic workflow environment. *Advanced Information Systems Engineering*, pages 175–188, 2004.

[50] Thomas Herrmann, Marcel Hoffmann, Kai-Uwe Loser, and Klaus Moysich. Semistructured models are surprisingly useful for user-centered design. In R. Dieng, A. Giboin, L. Karsenty, and G. De Michelis, editors, *Designing Cooperative Systems. Proceedings of Coop 2000*, pages 159–174, Amsterdam, 2000. IOS Press.

[51] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proc. Intl. Sym. on Applied Corporate Computing*, 1995.

[52] Josef Hofer-Alfeis. Document engineering: a preferred partner discipline in knowledge management. In *DocEng '09*, pages 1–2, New York, NY, USA, 2009. ACM.

[53] Jan A. P. Hoogervorst. *Enterprise Governance and Enterprise Engineering (The Enterprise Engineering Series)*. Springer, 1 edition, February 2009.

[54] Greg Horvath, Joe Bolinger, Jay Ramanathan, and Rajiv Ramnath. Document-centric collaborative spaces for increased traceability in knowledge-intensive processes. In *CTS '09*, pages 400–407, Washington, DC, USA, 2009. IEEE Computer Society.

[55] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM'08*, pages 1152–1163, Berlin, Heidelberg, 2008. Springer-Verlag.

[56] Richard Hull, Nanjangud C. Narendra, and Anil Nigam. Facilitating workflow interoperation using artifact-centric hubs. In *ICSOC/ServiceWave*, pages 1–18, 2009.

[57] Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. Ahead: A graph-based system for modeling and managing development processes. In *AGTIVE*, pages 325–339, 1999.

[58] Kurt Jensen. *Coloured Petri Nets : Basic Concepts, Analysis Methods and Practical Use. Volume 1 (Monographs in Theoretical Computer Science. An EATCS Series)*. Springer, April 2003.

[59] Jay Kennedy and Cherryl Schauder. *Records management: a guide for students and practitioners of records and information management*. Longman Cheshire, Melbourne, 1994.

[60] Jae Ho Kim, Woojong Suh, and Heeseok Lee. Document-based workflow modeling: a case-based reasoning approach. *Expert Syst. Appl.*, 23(2):77–93, 2002.

[61] Paul Klint and Jurgen Vinju. Writing language definitions in asf+sdf. Technical report, 2007.

[62] K. Kumar and M. M. Narasipuram. Defining requirements for business process flexibility. In *BPMDS'06 (CAiSE'06)*, Luxembourg, June 2006.

[63] V. Künzle and M. Reichert. Integrating users in object-aware process management systems: Issues and challenges. In Rinderle-Ma, Sadiq, and Leymann, editors, *BPM Workshops*, volume 43 of *Lecture Notes in BIP*, pages 29–41. Springer, 2009.

[64] Vera Künzle and Manfred Reichert. Towards object-aware process management systems: Issues, challenges, benefits. In *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 197–210. Springer Berlin Heidelberg, 2008.

[65] Vera Künzle, Barbara Weber, and Manfred Reichert. Object-aware business processes: Properties, requirements, existing approaches. Technical report, University of Ulm, Germany, 2010.

[66] Diego Adrian Naya Lazo. *OSWorkflow*. Packt Publishing, 2007.

[67] Mark H. Linehan. Ontologies and rules in business models. In *EDOCW '07*, pages 149–156, Washington, DC, USA, 2007. IEEE Computer Society.

[68] Beate List and Birgit Korherr. An evaluation of conceptual business process modelling languages. In *SAC '06*, pages 1532–1539, New York, NY, USA, 2006. ACM.

[69] Rong Liu, Kamal Bhattacharya, and Frederick Y. Wu. Modeling business contexture and behavior using business artifacts. In John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 324–339. Springer, 2007.

[70] R. Magalhães and A. Rito Silva. Organizational design and engineering. Technical report, Center for Organizational Design and Engineering, Lisboa, Portugal, 2009.

[71] David Martinho. An organizational blackboard for business process management. Technical report, Center for Organizational Design and Engineering, Lisboa, Portugal, 2009.

[72] Michael Merz, Boris Liberman, and Winfried Lamersdorf. Using mobile agents to support interorganizational workflow-management. *International Journal on Applied Artificial Intelligence*, 11(6), 1997.

[73] B.M. Michelson. Event-driven architecture overview. *Patricia Seybold Group, Feb*, 2006.

[74] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *CAiSE '08*, pages 48–63, Berlin, Heidelberg, 2008. Springer-Verlag.

[75] D. Müller, M. Reichert, J. Herbst, and F. Poppa. Data-driven design of engineering processes with corepro. *Enabling Technologies, IEEE International Workshops on*, 0:376–378, 2007.

[76] Dominic Müller, Joachim Herbst, Markus Hammori, and Manfred Reichert. Information technology support for release management processes in the automotive industry. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *BPM*, volume 4102 of *Lecture Notes in Computer Science*, chapter 26, pages 368–377–377. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2006.

[77] Dominic Müller, Manfred Reichert, and Joachim Herbst. Data-driven modeling and coordination of large process structures. In *OTM Conferences (1)*, pages 131–149, 2007.

[78] P. Nandi, D. König, S. Moser, R. Hull, V. Klicnik, S. Claussen, M. Kloppmann, and J. Vergo. Introducing business entities and the business entity definition language. Technical report, IBM, 2010.

[79] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *WWW '02*, pages 77–88, New York, NY, USA, 2002. ACM.

[80] Mariska Netjes, Irene T. P. Vanderfeesten, and Hajo A. Reijers. "intelligent" tools for workflow process redesign: A research agenda. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812, pages 444–453, 2005.

[81] A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Syst. J.*, 42(3):428–445, 2003.

[82] Y. Nomaguchi, M. Yoshioka, and Tetsuo Tomiyama. Document-based design process knowledge management for knowledge intensive engineering. In *IFIP TC5 WG5.2*, pages 131–144, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[83] OMG. *UML 2.0 Superstructure proposal v.2.0.*, January 2003.

[84] Martyn A. Ould. *Business Processes: Modelling and Analysis for Re-Engineering and Improvement (Role-Activity Diagrams)*. John Wiley and Sons, 1995.

[85] Mohammad Ashiqur Rahaman, Yves Roudier, and Andreas Schaad. Document-based dynamic workflows: Towards flexible and stateful services. *Services Part II, IEEE Congress on*, 0:87–94, 2009.

[86] Gil Regev, Ilia Bider, and Alain Wegmann. Defining business process flexibility with the help of invariants. *Software Process: Improvement and Practice*, 12(1):65–79, 2007.

[87] Gil Regev, Pnina Soffer, and Rainer Schmidt. Taxonomy of flexibility in business processes. In Gil Regev, Pnina Soffer, and Rainer Schmidt, editors, *BPMDS*, volume 236 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[88] Manfred Reichert and Peter Dadam. Adeptflex—supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems, Special Issue on WfMS*, 10(2):93–129, 1998.

[89] Hajo Reijers and Jan Mendling. Modularity in process models: Review and effects. In *Business Process Management*, pages 20–35, 2008.

[90] Hajo A. Reijers, Selma Limam, and Wil M. P. Van Der Aalst. Product-based workflow design. *J. Manage. Inf. Syst.*, 20(1):229–262, 2003.

[91] Clay Richardson. Process data management (keynote). Technical report, 2010.

[92] Paul Ricoeur. *Time and Narrative*. University of Chicago Press, 1988.

[93] Mary F Robek, Gerald F Brown, and David O Stephens. *Information and records management: document-based information systems*. New York : GLENCOE/McGraw-Hill, 4th ed edition, 1995.

[94] Willem De Roover and Jan Vanthienen. Unified patterns to transform business rules into an event coordination mechanism. In *edBPM'10*, 2010.

[95] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual (Collaboration Diagrams)*. Pearson Higher Education, 2 edition, 2004.

[96] N. Russell, A.H.M. ter Hofstede, and W.M.P. van der Aalst. newyawl: Designing a workflow system using coloured petri nets. *PNDS'08*, 2008.

[97] Nick Russell, Arthur, Wil M. P. van der Aalst, and Natalya Mulyar. Workflow control-flow patterns: A revised view. Technical report, BPMcenter.org, 2006.

[98] Wasim Sadiq, Karsten Schulz, Maria E. Orlowska, and Shazia Sadiq. When workflows will not deliver - the case of contradicting work practice. In Witold Abramowicz, editor, *BIS 2005*, pages 69–84. Wydawnictwo Akademii Ekonomicznej w Poznaniu, 2005.

[99] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, and Wil M. P. van der Aalst. Process flexibility: A survey of contemporary approaches. *CIAO! / EOMAS*, 2008.

[100] Reza Shafii. Kaizen, bpm, and agile methodologies. http://www.oracle.com/technology/pub/articles /dev2arch/2008/05/kaizen-bpm-agile.html (accessed December 1, 2009), 2008.

[101] António Rito Silva, David Martinho, Ademar Aguiar, Nuno Flores, Filipe Correia, and Hugo Sereno Ferreira. The adaptive object-model architectural style. In *The Second Workshop on Business Process Management and Social Software*, IST/UTL Center for Organizational Design and Engineering - INOV and INESC Porto, 2009.

[102] R. Snowdon, B. Warboys, R. Greenwood, C. Holland, P. Kawalek, and D. Shaw. On the architecture and form of flexible process support. *Software Process: Improvement and Practice*, 12(1):21–34, 2007.

[103] M. Suntinger, H. Obweger, J. Schiefer, and M. E. Groller. Event tunnel: Exploring event-driven business processes. *Computer Graphics and Applications, IEEE*, 28(5):46–55, Sept 2008.

[104] Hong Linh Truong and Schahram Dustdar. Integrating data for business process management. *IEEE Data Eng. Bull.*, 32(3):48–53, 2009.

[105] G. Van Bussel, F. Ector, G. Van der Pijl, and P. Ribbers. Building the record keeping system (rks). process improvement triggered by management of archival documents. In *HICSS '01*, page 8060, Washington, DC, USA, 2001. IEEE Computer Society.

[106] W. M. P. van der Aalst. On the automatic generation of workflow processes based on product structures. *Comput. Ind.*, 39(2):97–111, 1999.

[107] W. M. P. van der Aalst and P. J. S. Berens. Beyond workflow management: product-driven case handling. In *GROUP'01*, pages 42–51, New York, NY, USA, 2001. ACM.

[108] W. M. P. van der Aalst and Ter A. H. M. Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[109] W. M. P. van der Aalst and S. Jablonski. Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science and Engineering*, 15(5):267–276, September 2000.

[110] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[111] Wil M. P. van der Aalst, Paulo Barthelmess, Clarence A. Ellis, and Jacques Wainer. Proclets: A framework for lightweight interacting workflow processes. *Int. J. Cooperative Inf. Syst.*, 10(4):443–481, 2001.

[112] Wil M. P. van der Aalst, R. S. Mans, and Nick C. Russell. Workflow support using proclets: Divide, interact, and conquer. *IEEE Data Eng. Bull.*, 32(3):16–22, 2009.

[113] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - RD*, 23(2):99–113, 2009.

[114] Wil M. P. van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[115] W.M.P. van der Aalst, R.S. Mans, and N.C. Russell. Workflow support using proclets: divide, interact and conquer. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 32(3):16–22, 2009.

[116] Irene T. P. Vanderfeesten, Hajo A. Reijers, and Wil M. P. van der Aalst. Product based workflow support: Dynamic workflow execution. In Zohra Bellahsene and Michel Léonard, editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 571–574. Springer, 2008.

[117] I.T.P. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product based workflow support: a recommendation service for dynamic workflow execution. Technical report, Eindhoven: BPMcenter.org, 2008.

[118] E.A. van Veen and J.C. Wortmann. New developments in generative bom processing systems. *Production Planning and Control*, 3(3):327–335, 1992.

[119] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. *Hawaii International Conference on System Sciences*, 7:7051, 2001.

[120] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[121] M. Zacarias, R. Magalhães, A. Caetano, H. S. Pinto, and J. Tribolet. Towards organizational self-awareness: An initial architecture and ontology. In P. Rittgen, editor, *Handbook of Ontologies for Business Interaction*, pages 101–121. Idea Group Inc, 2007.

[122] Michael zur Muehlen. Volume versus variance: Implications of data-intensive workflows. *IEEE Data Eng. Bull.*, 32(3):42–47, 2009.

VIII

Appendices

15

Event-driven Solution

When implementing and executing the developed modeling approach, some considerations arise. First, all the system entities creation, life-cycle progress and archival is dependent on rule-conditions' evaluation. Understandably, for hundreds of entities to interact in an efficient manner, active listening is not an option to implement rules. The object-centered modeling, thus, adopts an event-driven solution. Every system entity dynamically publishes and subscribes a set of events.

Published events can either be attribute value-changes or state-changes – when an entity instance's marking change, a new event associated with the entity instance is triggered with the new marking. In fact, marking-change events are, in essence, a composition of a set of atomic events related to each life-cycle place that suffered a change on its tokens. For instance, [1 1 0 0]→[0 0 0 1] is a composition of three atomic events corresponding to the change of tokens associated to three places.

These events are triggered by rule's actions that, in essence, fire, multiply or remove the tokens of the set of referred places. Additionally the submission, cancellation, suspension or skipping of forms and of its fields also trigger events, whose specification includes their identification, signal and filled-values. Exemplifying, an atomic activity model related to the price data-attribute is on the running state and, thus, have an available form-field. The filled form-field is submitted and, consequently, an event is triggered. Since the *running→succeed* condition depends on such event (price>0), its condition is evaluated and an action is triggered. Finally, if an object model transition depends on the price attribute, m(price.succeed)=1, now can possibly progress to a new place.

Similarly, the *subscribed events* also relate to changes in data and in places' marking. They are required for the evaluation of rule-set conditions, that is performed whenever an atomic event related with a rule input place is triggered. Note that not only the places from the target entity may be referred but also places from an external or upper entity with whom the target entity is synchronized.

15.1 Object-centered Systems are Event-driven by Nature

Back to the introduced axioms on the modeling of data-intensive systems, systems can be modeled as a set of state-based entities – objects, activities, goals and time – that interact with each other by synchronizing their life-cycles. If system entities interact in an aynchronous and loosely-coupled manner and coordinate their progress through event publishing and subscription, several benefits can be seized:

- *system responsiveness* – since events can occur at any time from any source and the use of ordering precedences constraining the ability to scan these events is limited (as occuring in activity-centered processes), activities respond to events immediately, whenever and wherever they happen;
- *system agility* – since the adaptation of system processes is dynamically retrieved from event-driven objects, processes can response in a more timely manner to changing system requirements [25];
- *system flexibility* – since agents interact with process modeling landscape via events (either through adaptors or by manual filling of forms), all the human and application participants can be added, removed or upgraded without affecting process modeling [73].

An event-driven solution applied to the object-centered modeling can additionally integrate external system interactions using an unified interface [73]. External entity systems must subscribe events for those entities with whom they want to contact and also publish events corresponding, for instance, to a new order or to a notification's response. Both notifications and publications are standardized events that contain the entity, its new state and optionally a set of data-values. Fig.15.1 presents this landscape.

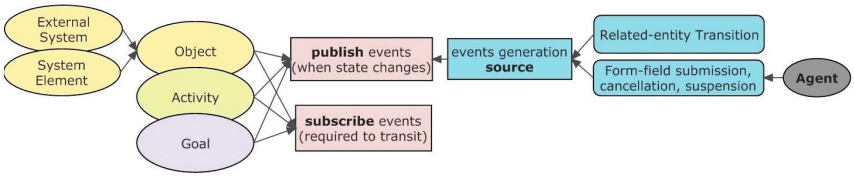


Figure 15.1: Event-driven nature of data-intensive systems

15.2 Advantages

15.2.1 Modeling Traceability and Correctness

One of the greatest problems of traditional process modeling approaches, not so often referred, are their inability to trace events. In a simple scenario, where the occurrence of an activity is dependent on a control-flow with an event being triggered on a data-condition satisfaction, if the event is triggered the activity becomes available. But how does one know that such event is anytime triggered? If its occurrence may depend on another event triggered by the waiting activity, this event will never occur, so the process model is not sound. Correctness is a big issue in traditional modeling landscapes [47][108].

But even if this event can occur as result of a data-change, how does one know which system entities were responsible for that change? In activity-centered landscapes events are hardly related, this works names this problem as no-source-traceability.

In object-centered modeling no longer this problem arises since data and time are modeled at the process modeling level, and since the entities' coordination dependencies are used to build a complete source-traceable net of events to formulate advanced soundness criteria.

15.2.2 Modeling Usability

Another concern arising from the use of events, is the possible great amount of event-dependencies among all the system entities that would degrade the ability to evolve object-centered models as it may require the understanding of all the event-based entities interplay. The object-centered modeling, however, fosters the use of encapsulation methods to synchronize objects through events. By limiting the scope of synchronization, it is easier to trace graphically the events' dependencies. This results in an improved usability and locality of changes that foster models' evolution.

15.2.3 Complex Event Processing and Modeling Abstractions

Complex-event processing (CEP) is an emerging discipline to deal with event-driven behavior that enables the processing of large amounts of events, using them to monitor, steer, optimize and discover exceptional situations or opportunities in system processes with minimal latency [73]. The presented way of publishing and subscribing events offers a formal and stable criterion to correlate temporally and semantically events.

This brings two clear advantages. First, the ability to support advanced coordination patterns as the vertical aggregation of activities based on the management of events. For instance, the joining of all the assets already procured for collective supplying, can be done recurring to collective events-publishing after a time-buffer, which can be easily implemented recurring to CEP rules. Advanced vertical aggregation constructs implemented through CEP are well-covered in [103].

Second, the real-time selection, aggregation, and event abstraction may generate higher-level compound events of interest. Entity instances may produce lower-level events as attribute-changes corresponding to RFID tags or log-file entries [73]. Such events may be correlated to produce new higher-level compound events, not only to be used by monitoring, business intelligence capabilities or exceptional behavior but also to dynamically affect the normal progress of object instances [25]. Note that one of the fundamental obstacles for intelligent systems is the absence of an automated ability to relate events as the human intuition. With this solution, the role of an event modeler must be reviewed since it only differs from a process modeler on the level of granularity of the target entities under modeling.

15.3 Notes on the Implementation Architecture

These advantages turn natural the choice of an *event-driven architecture*[1] to support the object-centered process modeling landscape. Note that data-intensive systems have a natural fit with an event-driven solution as they are, by design, composed by loosely-coupled interacting entities (an event itself may not know about the consequences of its cause), responsive to data-changes, and more prone to unpredictable and asynchronous environments [25].

Fig.15.2 introduces a possible architecture. Where, as an independent system, CEP is a parallel running platform that analyses and processes events in an integrated fashion with the process monitoring tool using additional events produced by the object-centered workflow engine.

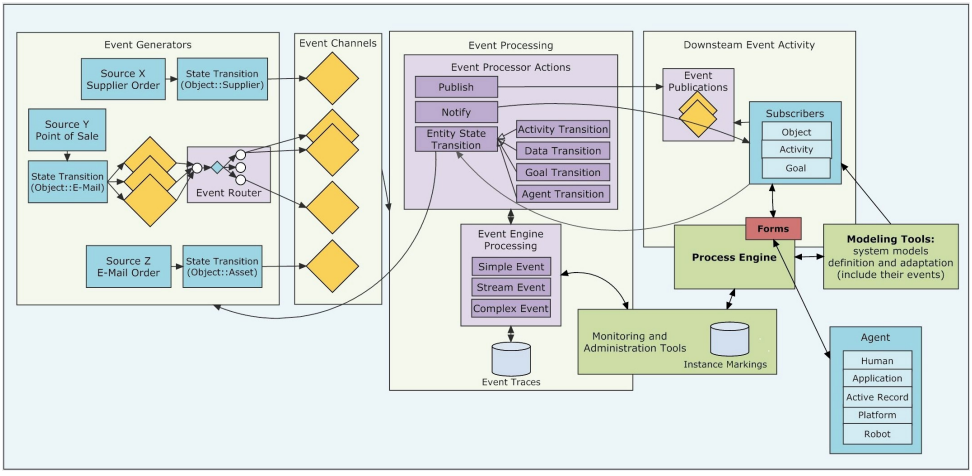


Figure 15.2: Architecture for the event-driven and object-centered modeling landscape

[1] software architecture pattern promoting the production, detection, consumption of, and reaction to events

16
Thesis Boundary

This thesis has been focused around five requirements that compromise the ability of data-intensive systems to evolve. Does, however, their satisfaction guarantee the applicability of an object-centered approach? To which degree they cover all the requirements for a successful data-centered approach?

This annex uses the *five* requirements to understand how the cover the overall modeling aspects. *Section 16.1* introduces the developed framework to structure process modeling aspects, and *section 16.2* exploits how each requirement relates with those aspects and, thus, fosters the evolution of data-intensive processes.

16.1 The Process Model Framework

The *Process Model Framework* is a formal guideline for the development, improvement and analysis of process models. This framework aims to structure a wide set of concerns in a consistent, coherent and usable way. It was developed through an inductive path based on both theoretical and empirical research, where concepts and relationships were firstly discovered and then organized into a theoretical explanatory scheme.

This framework is defined as a tuple $< M, D, A >$, where M is a set of modeling approaches, D is a set of modeling domains and A is a set of areas of concerns. Main contribution is on promoting an accepted ontology of concepts and leveraging the awareness of different aspects when deriving a modeling approach. In practice this framework serves three main purposes: *i)* to classify or retrieve requirements (*descriptive* side), *ii)* to guide the development of a process meta-model by turning natural the definition of principles (*prescriptive* side), and *iii)* to serve as a basis for evaluating existing approaches.

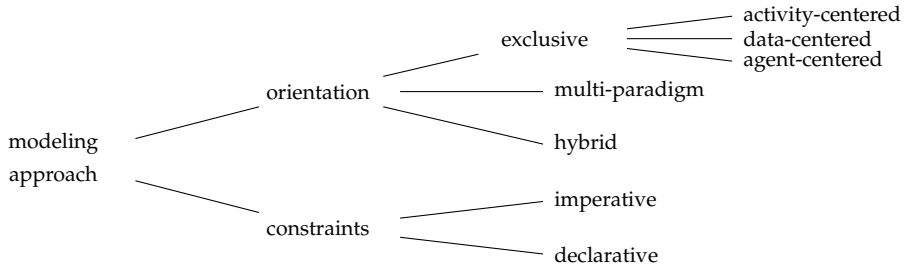
		M_1			...	M_n		
		D_1	...	D_n	...	D_1	...	D_n
A_1	A_{11}							
	...							
	A_{1n}							
...	...							
A_n	A_{11}							
	...							
	A_{1n}							

Table 16.1: Process Model Framework's skeleton

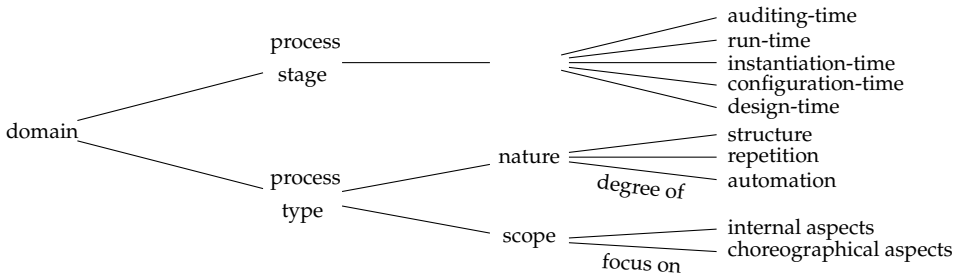
The potential resides on its simplicity as a result of the visual orthogonality and from the fact that the cells that result from the intersection of the three framework dimensions provide the right level of abstraction to extract requirements, define principles and evaluate approaches (see Table 16.1).

Modeling Approaches M: Processes exist in every kind of system and, therefore, the way they are modeled must be aligned with their system's properties and needs. Two dimensions underpin the M set: *process orientation* and *constraints specification*. Different orientation paradigms were introduced in section 2.5. Each one of them can, however, follow an *imperative* or *declarative* way to specify constraints. In imperative approaches the tasks, data or agents dependencies are explicitly defined [99][120]. In

declarative settings the focus is on what should be done, instead of how, by using constraints – either mandatory (strictly enforced) or optional (possibly violated) – to limit the execution options [113].



Modeling Domains *D***:** This dimension aims to properly separate and structure the aspects of a particular area of concern and modeling approach. Concerns can be structured according to the *process stage* where they occur. A simple and consistent space division defines design and run time. This framework adopts, however, [113] classification as it better isolates different concerns into distinctive five moments. Concerns also vary depending on the *type of process* under analysis. Type is defined by: *i)* the scope or interaction extent (either intra- or inter-systemic) and by *ii)* the nature of a process [120] that is defined by the process degree of automation[1], repetition[2] and structuring[3] [120].



Areas of Concern *A***:** Modeling approaches can be evaluated by the extent to which they provide constructs useful for representing and reasoning about the various aspects of a process. This dimension structures the intrinsic *qualities* and *support* ability of process models. The proposed taxonomy can be mapped in the perspectives proposed in [68] and extended in [28]. The realization of these concerns is a continuous playing of trade-offs.

Modeling **qualities** can be consistently divided according to:

- *expressivity* refers to the extent to which an approach models execution constraints, comprising the ability to specify simple and advanced[4] control-flow patterns based on participants and activities;
- *granular pliancy* is the ability to model system elements at an arbitrary level of granularity, to coherently bridge those nesting levels, and to assure the right level of atomicity and the use of a uniform methodology to avoid incomparability among process fragments;
- *flexibility* is the ability of process models to adapt without loss of identity [102][29], i.e., without the need to replace them completely [87]. Kumar [62] considers two types of flexibility: pre-designed (here captured as expressivity) and just-in-time responsive. Responsive flexibility determines the adaptation ability using dynamic changes, deferred decisions and deviation to specifications;
- *usability*, the ease process users can specify, configure and execute processes based on models,

[1]ability of a process to be supported by software systems

[2]ability of a process to sustain a set of execution paths

[3]ability of a process to completely prescribe activities and their execution constraints

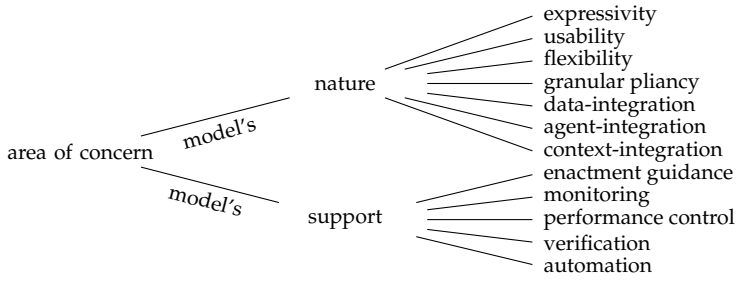
[4]include: i) vertical dynamic aggregation (ability to synchronize multiple instances from different process types [64]), and ii) state-based execution, which comprises horizontal dynamic aggregation (ability to turn activities available based on participants state conditions [64])

depends on metrics as model's simplicity and readability and affects process agents state and relations as fosters modeling learnability and communication [26];

- we can say that a model is *integrated* with other model if there are direct dependencies between those models that consistently and coherently define a third model that can govern those models evolution. Said this, *data-*, *agent-* and *context-integration* are defined as the ability to integrate data, agent and goal models with activity models through a governance or process model.

Modeling **support** refers to the ability to transfer the process modeling concerns from human to software agents. Since highly depends on how models are implemented in practice [120] and this work adopts only a system modeling perspective, they will not be totally covered. Aalst et. al [113] distinguishes between: *i)* design-time support, as the ability to guide modeling and assure their correctness and performance acceptability, and *ii)* run-time support, as the ability to guide monitoring and execution of process instances. This work proposes a consistent fifth-nary division:

- *automation area* depicts the space of modeling handled by non-human agents. A fully-automated environment is still utopian, nevertheless dynamic process derivation based on data models [77][117] or on goal-based models [41][79][24] are two directions that shorten the distance to higher levels of automation as, for instance, is observed in semantic process modeling [13]. In increasingly complex environments, these strategies can strongly simplify the evolution of models;
- *verification* assures that processes are correct during the modeling and enactment of processes (as a result of adaptations, deviations or deferred decisions) [114], and can be either syntactical (based on models' sound criteria) or semantical (grounded on domain knowledge) [113];
- *performance control* assures that support tasks are performed in useful time by obtaining and evaluating the acceptability of metrics as processes completion time, level of service, utilization of capacity and verification period [114] (quantitative aspects *vs.* qualitative or correctness);
- *enactment guidance* refers to the assistance degree provided to users during process modeling and enactment, since correct and efficient models may not behave as desired [113]. Domain specific languages can be used either to enforce or recommend effective adaptations and executions [39];
- *monitoring* is the ability to trace and audit in real-time a process. By promoting control and leveraging the knowledge of hidden process patterns [11], models trace enables the continuous improvement of data-intensive systems fostering self-awareness and learnability.



16.2 How Requirements foster Evolution of Data-intensive Systems?

To answer this question, there is the need to understand which modeling aspects promote the desired evolutionary landscape. In last section the process model framework was introduced as a matter to evaluate or derive modeling concerns spread among a set of domains and according to a modeling approach. Different approaches, thus, answer differently to modeling concerns in each domain. Since

meta-modeling is a science of trade-offs, flexibility and expressivity of models may not increase unless their usability and verification are penalized.

In fact, all the concerns affect evolution. Recovering the fact that evolution is triggered by integration, adaptability and agility, several conditions can be stated. First, system informational, instrumental and purposeful views must coherently be bridged and affect execution constraints by being an expressive source for automatically detectable changes. Thus, evolution depends on data, agent and context integration. Expressivity, flexibility and traceability are at the core of evolution notion as they define the system adaptability. Finally, usability, granular pliancy, automation ability, enactment guidance, and qualitative and quantitative compliance determine the agility of system adaptations.

By understanding how modeling concerns foster evolution, the requirements effects can now be easily structured. First, each requirement may intersect multiple areas of concern at different process stages. Let us focus on usability to understand how it is fostered by each requirement. Data-access affects usability since advanced ways to capture activities' data-context exist than explicit input and output parameters for every activity. Data-based reaction and coordination lead to simpler models. Data modeling affects usability as it promotes a near-complete derivation of process models from enriched data models. And, finally, data-based granularity promoting functional decomposition also increases the ability to work under abstractions, the process fragments.

All requirements affect not only modeling qualities but modeling support as well – e.g. data-based coordination poses new challenges for model verification. The summarized rationale used for usability can be extended for every modeling concern in order to study the extent to which these data-specific requirements foster evolution.

17

Implementation Notes

As the set of object-centered models can be considered a high-level domain-specific language to lower-level constructs provided by enriched *YAWL* models, syntax specification and model-to-model transformations were defined in order to proof the mapping correctness. The choice of using the *ASF+SDF* engine for this task was exploited in *chapter 10*. This chapter introduces the features of the developed application (*section 17.1*), and briefly covers some of its syntax and transformation implementation aspects (*section 17.2*).

17.1 The Application

To interact with the developed application the user just needs to specify one or more models and apply a set of transformations (from the available and valid range) to it, which will return a transformed model. For instance, if the reader wants to specify an object-centered process model for the GF case, it might be a good option to simply start by defining a data model using UML notation. Using this input, a transformation to generate an object model skeleton is available. Then, the reader just need to add data-dependencies, life-cycles and points of synchronization. After this specification several transformations are available, from the default generation of rule-set models or data-contexts until the related activity-models generation. Assuming that activity-models are generated, the reader can now use both object and activity models' documents as input and apply the transformation responsible to generate process models. Finally, using these object-centered process models new transformations can be applied in order to derive a proclets net enriched with multi-colored places conditions.

The command available for the application of transformations is the following:
COMMANDLINE> ACTION=var_1 FILE=var_2 [FILE2=var_3] , with var_1=dm|om-1|om-2|om-3|am-4|am-5|am-6|pm-8|pm-9|yawl-plain|yawl-proclets|om-sound|am-sound |pm-sound|format being the type of transformation to apply, var_2 being the path to the input model to transform, and, optionally, var_3 being the path to a second input model to transform that is used when an object-centered process model is derived from two documents containing, respectively, object and activity models.

Not only transformations are available, but also soundness verifications and a pretty-printer. In fact, their use may not be so regular since both are internally applied after each transformation.

To support these transformations some steps need to be performed. First, an optional *merging* of models. Second, the *parsing* of the input files using a parse table adequated to the type of the given file (e.g. an object model needs to be parsed by a table derived from the object models syntax). Third, the *rewriting* of the parsed tree according to the algebraic equations file for the chosen transformation. This is the core step. Fourth, *soundness* is verified according to the input file – in case of failure, the input model is rewritten as a simple message containing the sound failing clause. Fifth, *formatting* adds constructs for the unparsed tree being printed in an usable manner. Finally, the parsed tree is *unparsed*.

The main modules of the application, for a possible further exploitation, are organized as follows:

syntax – object-centered models syntax;
transformations – the proposed transformations (DM->OM,OM->AM,OM,AM->PM, etc.);
format – pretty-printer;
soundness – soundness check (our language "typechecker");
migration – to support run-time adaptations/changes to the object-centered models;
mapping – mapping of the target language to enriched yawl models;
bin – contain the parsers, unparsers, trees' transformers and additional auxiliary executables;
build – contain the language specific executables: ASF tables for parsing and algebraic equations;
examples – contain a set of input models to apply the defined transformations;
output – contain the intermediary and final outputs;

17.2 Models Syntax and Transformations

The length of syntax definitions and, mainly, of algebraic equations do not allow their full explanation. However, two examples are provided for the reader understand the basics on how to capture the syntax of a model using SDF (*Alg. 17.1*) and on how to specify rewriting rules using ASF (*Alg. 17.2*).

The *ASF+SDF Meta-Environment*, illustrated in Fig.17.1, is a very useful graphical interface for the edition of SDF and ASF models, for the dynamic parsing and checking of the correctness of the specification modules, and for generating parsers for the syntax modules and rewriter engines for the equations modules. It may be used in alternative to the developed commands used in a shell-based interaction.

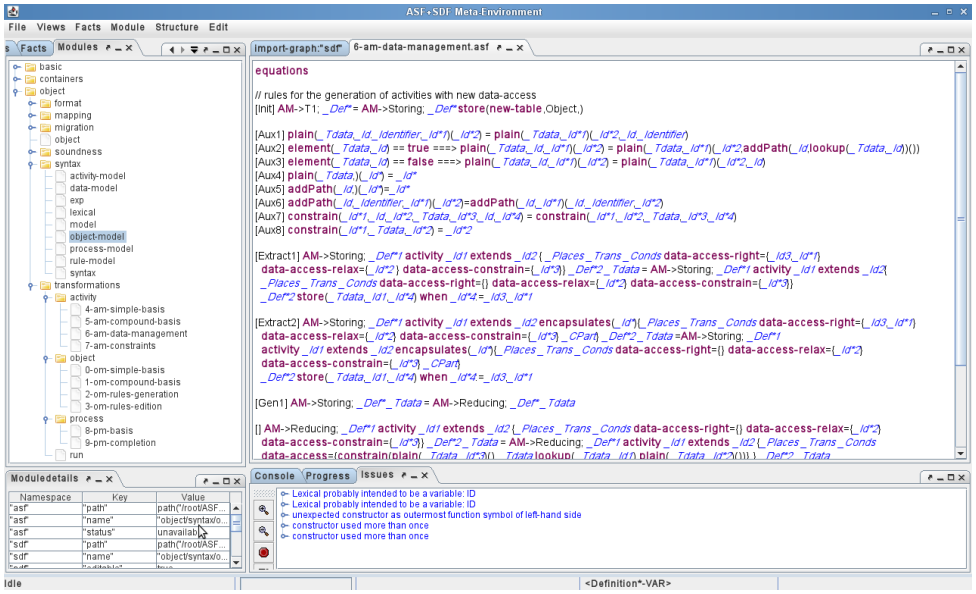


Figure 17.1: Extract of the algebraic equations used for the data-contexts definition

To obtain the application or for further clarifications please contact the author of this thesis.

Algoritm 17.1: Syntax specification of a *simplified* proclets net model

```
module object/mapping/yawl-syntax/proclets
imports
    object/mapping/yawl-syntax/yawl
    object/syntax/model
    object/syntax/exp
hiddens
    context-free start-symbols Application
exports
    sorts ProcletApplication ProcletBasedProcess Proclet ProcletBody Definition ProcletsComposition
    context-free syntax
      ProcletBasedProcess* -> ProcletApplication{cons("App")}
      AppStatusType ";" Top ProcletsComposition Channels
        Performatives NamingService Definition* -> ProcletBasedProcess{cons("ProcletsProc")}
      "Proclets" -> AppType{cons("ProcletsApp1")}
      "Proclets" "->" Id -> AppType{cons("ProcletsApp2")}
      Proclet -> Definition{cons("Proclet")}
      "Proclet" Id "{" ProcletBody "}" -> Proclet{cons("Proclet")}
      "Proclet" Id "(" {TaskInstance ","}* ")" "{" ProcletBody "}" -> Proclet{cons("CProclet")}
      LifeCycle WData InPorts OutPorts -> ProcletBody{cons("PBody")}
      "proclets" "=" "{" TREE "}" -> ProcletsComposition{cons("PHierarchy")}
    sorts LifeCycle Performatives Performative Channels
    context-free syntax
      "life-cycle" "=" "{" Init Final NetConds NetTasks Flows "}" -> LifeCycle{cons("LifeCycle")}
      "channels" "=" "{" {Channel ","}* "}" -> Channels{cons("Channels")}
      "performatives" "=" "{" {Performative ","}* "}" -> Performatives{cons("Perfs")}
      Id "(" ChannelId "," InPortsId "," OutPortsId "," Type ")" -> Performative{cons("Performative")}
    sorts Port InPorts OutPorts InPortsId OutPortsId Cardinality Multiplicity
    context-free syntax
      Id Cond Cardinality Multiplicity -> Port{cons("Port")}
      "in-ports" "=" "{" {Port ","}* "}" -> InPorts{cons("InPorts")}
      "out-ports" "=" "{" {Port ","}* "}" -> OutPorts{cons("OutPorts")}
      "in-ports" "=" "{" {Id ","}*"}" -> InPortsId{cons("InPorts")}
      "out-ports" "=" "{" {Id ","}*"}" -> OutPortsId{cons("OutPorts")}
      "cardinality" "=" "{" Card Card "(" Time ")" "}" -> Cardinality{cons("Card")}
      "multiplicity" "=" "{" Card Card "(" Time ")" "}" -> Multiplicity{cons("Mult")}
    sorts NamingService ProcletInfo ProcletId Endpoint
    context-free syntax
      "ns" "=" "{" {ProcletInfo ","}* "}" -> NamingService{cons("NS")}
      ProcletId "=" Endpoint -> ProcletInfo{cons("Info")}
      Id -> ProcletId{cons("Id")}
      "http://localhost:8080/" Id -> Endpoint{cons("Id")}
    sorts Channel QualityParameter Type Cond ChannelId
    context-free syntax
      Id "(" {QualityParameter ","}* ";" InPortsId ";" OutPortsId ")" -> Channel{cons("Channel")}
      "type" "=" Id -> QualityParameter{cons("Type")}
      "reliability" "=" Id -> QualityParameter{cons("Reliability")}
      "security" "=" Id -> QualityParameter{cons("Security")}
      "synchronization" "=" Id -> QualityParameter{cons("Synchro")}
      "closure" "=" Id -> QualityParameter{cons("Closure")}
      "formality" "=" Id -> QualityParameter{cons("Formality")}
      "channel" "=" Id -> ChannelId{cons("Channel")}
      "type" "=" Id -> Type{cons("Type")}
      "cond" "=" Id -> Cond{cons("Cond")}
```

Algoritm 17.2: Brief extract of the algebraic equations for the mapping in plain YAWL models

```
equations
// PART1: Auxiliary functions [...]
// PART2: Creating the tasks' join and split operators based on rules
[] fillYAWLTasks(,_C*)(_Task*)=_Task*
[] fillYAWLTasks(_Id,_Id*,_C*)(_Task*)=fillYAWLTasks(_Id*,_C*)(_Id (join(_Id,_C*),split(_Id,_C*)),clean(),1 1 static, _Task*)
[] split(_Id,_C*1,(_Id,_Id1)._Id0,_C*2,(_Id,_Id2)._Id0,_C*3)=and
[] split(_Id,_C*1,(_Id,_Id1)._Id2,_C*2,(_Id,_Id3)._Id4,_C*3)=or
[] split(_Id,_C*)=normal
[] join(_Id,_C*1,(_Id1,_Id)._Id0,_C*2,(_Id2,_Id)._Id0,_C*3)=and
[] join(_Id,_C*1,(_Id1,_Id)._Id2,_C*2,(_Id3,_Id)._Id4,_C*3)=or
[] join(_Id,_C*)=normal
// PART3: Integrating object-places and activity-places into one unique life-cycle
[] PM->T3; _Def*1 process _Identifier extends _Id1 encapsulates (_Id*) {
     _Vars tasks={_Id*4} init=_Id2 final=_Id3 places={_Id*1} transitions={_T*1} conds={_C*1}
     ort-init=_Id4 ort-final=_Id5 ort-places={_Id*2} ort-transitions={_T*2} ort-conds={_C*2}      sub-processes={_Frame*} _Data}
   _Def*2 = PM->T3; _Def*1 process _Identifier extends _Id1 encapsulates (_Id*) {
     _Vars tasks={_Id*4} init=_Id2 final=_Id3 places={init,addP(obj,_Id*1)(),addP(act,_Id*2)(),completed}
     transitions={addT(obj,_T*1)(),addT(act,_T*2)(),(init,_Id2.obj),(init,_Id4.act),(_Id3.obj,completed),(_Id5.act,completed)}
     conds={addC(obj,_C*1)(),addC(act,_C*2)(),(init,_Id2.obj)._Identifier.wnet.R0,(init,_Id4.act)._Identifier.wnet.R0,
       (_Id3.obj,completed)._Identifier.wnet.R1,(_Id5.act,completed)._Identifier.wnet.R1}
     sub-processes={addF(obj,_Frame*)()} _Data} _Def*2
   rule _Identifier.wnet.R0{domain=_Identifier.init} range={_Id6,_Id7} conds={true}
     actions={marking(_Id6):=marking(_Id6)+1,marking(_Id7):=marking(_Id7)+1,marking(_Identifier.init):=0}}
   rule _Identifier.wnet.R1{domain=_Identifier.completed} range={_Id8,_Id9} conds={true}
     actions={marking(_Id8):=marking(_Id8)+1,marking(_Id9):=marking(_Id9)+1,marking(_Identifier.completed):=0}}
   when _Id6:=add(_Identifier,_Id2.obj), _Id7:=add(_Identifier,_Id4.act),_Id8:=add(_Id.,_Id3.obj), _Id9:=add(_Identifier,_Id5.act)
%% [...]
//PART4: storage composition [...]
//PART5: mapping an object-centered process to a yawl process basis
[] PM->T5; _Def*1 process _Id extends _Id1 {
     tasks={_Id*0} init=_Id2 final=_Id3 places={_Id*1} transitions={_T*1} conds={_C*1} data-access={_Id*2}} _Def*2 _Tproc
   = PM->T5; _Def*1 WNet _Id { init=_Id2 final=_Id3 conditions={_Id*1} tasks={fillYAWLTasks(_Id*0,_C*1)()}
     flows={flow(_C*1)()} data-access={_Id*2}} _Def*2 _Tproc
%% [...]
//PART6: composition of the WNet tree
[] search([_Id,_Id*],[_Id*1,_Id,_Id*2]) = _Id
[] search([_Id,_Id*],[_Id*2]) = search([_Id*],[_Id*2])
[] findTop(_Tproc) = search(keys(_Tproc),values(_Tproc))
[] extractNodes([_Id,_Id*],_Tproc)(_N*) = extractNodes([_Id*],_Tproc)(_N*,_Id;lookup(_Tproc,_Id))
[] extractNodes([],_Tproc)(_N*)=_N*
[] PM->T5; _Def* _Tproc = WNet; top=Object workflows={ buildTree( extractNodes(keys(_Tproc),_Tproc)() )((Object))}
   _Def* _Tproc when _Id:=findTop(_Tproc)
//PART7: completion of removal and instantiation patterns [...]
//PATY 8: Generating tasks for the support of control-flows [...]
[] or-out(_Id,_Flow*)==true, and-out(_Id,_Flow*)==false ===> pattern-out(_Id,_Flow*)=or
[] or-out(_Id,_Flow*)==false,and-out(_Id,_Flow*)==true ===> pattern-out(_Id,_Flow*)=and
[] or-out(_Id,_Flow*)==false,and-out(_Id,_Flow*)==false ===> pattern-out(_Id,_Flow*)=normal
[] or-out(_Id,_Flow*)==true ,and-out(_Id,_Flow*)==true ===> pattern-out(_Id,_Flow*)=compound
[] pattern-out(_Id,_Flow*)==or ===> addInitTasks(_Id,_Flow*) = _Id.out.or (normal,or),clean(),1 1 static
[] pattern-out(_Id,_Flow*)==and ===> addInitTasks(_Id,_Flow*) = _Id.out.and (normal,and),clean(),1 1 static
[] pattern-out(_Id,_Flow*)==normal===> addInitTasks(_Id,_Flow*) =
[] pattern-out(_Id,_Flow*)==compound ===> addInitTasks(_Id,_Flow*) = _Id.out.compound (normal,and),clean(),1 1 static,
   _Id.out.and (normal,and),clean(),1 1 static, _Id.out.or (normal,or),clean(),1 1 static
[] isUnique(_Id2,_Flow*1,_Flow*2,_Flow*)==true ===> linkOutCompound(_Id,_Flow*1,(_Id1,_Id)._Id2,_Flow*2)(_Flow*) =
   linkOutCompound(_Id,_Flow*1,_Flow*2)((_Id1,_Id.in.and)._Id2,_Flow*)
[] isUnique(_Id2,_Flow*1,_Flow*2,_Flow*)==false ===> linkOutCompound(_Id,_Flow*1,(_Id1,_Id)._Id2,_Flow*2)(_Flow*) =
   linkOutCompound(_Id,_Flow*1,_Flow*2)((_Id1,_Id.in.or)._Id2,_Flow*)
[] linkOutCompound(_Id,_Flow*1)(_Flow*)=_Flow*1,_Flow*
[] addFlows(_Id.out.or_TBody, _Task*)(_Flow*)=addFlows(_Task*)(linkIn(_Id,_Id.out.or,_Flow*)(),(_Id,_Id.out.or)._Id.out.or)
[] addFlows(_Id.out.and_TBody, _Task*)(_Flow*)=addFlows(_Task*)(linkIn(_Id,_Id.out.and,_Flow*)(),(_Id,_Id.out.and)._Id.out.and)
[] addFlows(_Id.out.compound_TBody, _Task1, _Task2, _Task*)(_Flow*) = addFlows(_Task*)( linkInCompound(_Id,_Flow*)(),
   (_Id,_Id.out.compound)._Id.out.compound, (_Id.out.compound,_Id.out.and)._Id.out.and, (_Id.out.compound,_Id.out.or)._Id.out.or)
%% [...]
```

18

GF Case Annexes

This chapter illustrates the object-centered modeling steps. The following illustrations do not dispense the careful reading of the thesis content, as due to the sake of simplicity they just cover a subset of all the proposed constructs. These illustrations have a quality that enables their amplification when using the original *pdf* version. If the reader wants to obtain such *pdf* version, please contact the author.

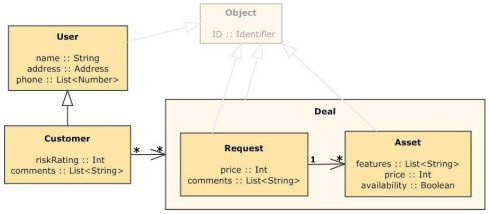


Figure 18.1: Simple data model

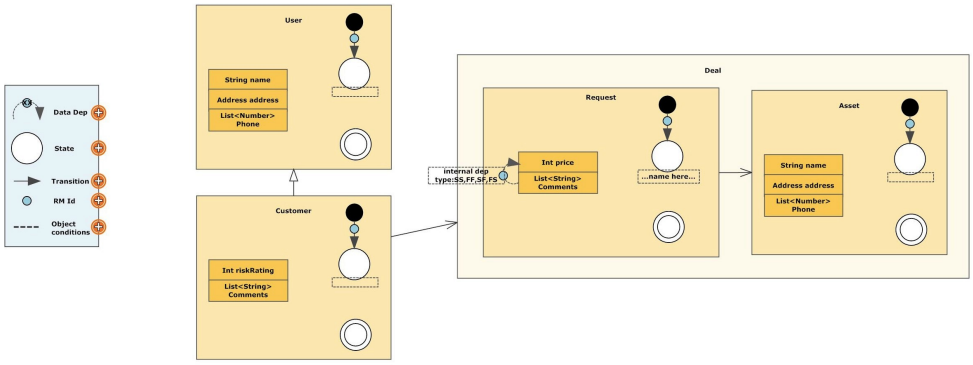


Figure 18.2: Automatic generation of object models skeleton

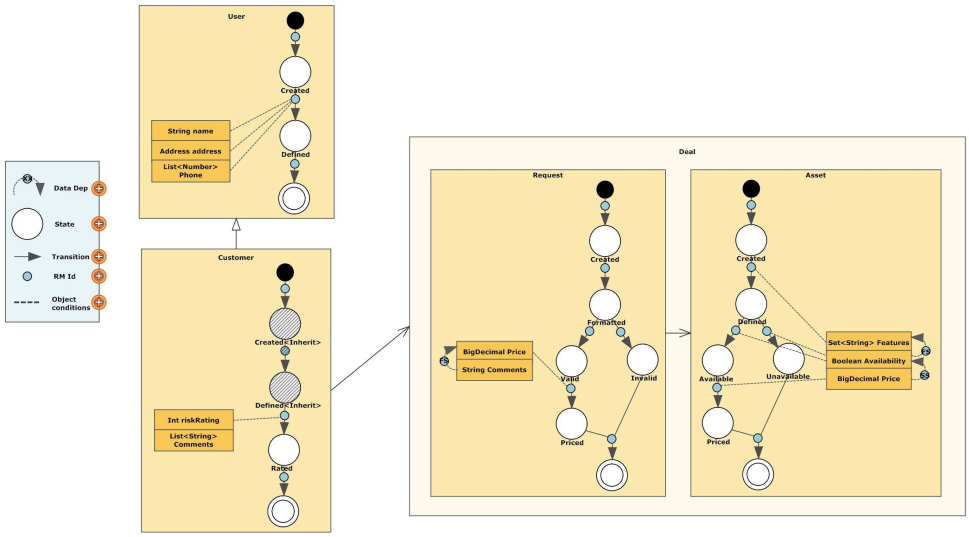


Figure 18.3: Manual enrichment of objects with data-dependencies and life-cycles

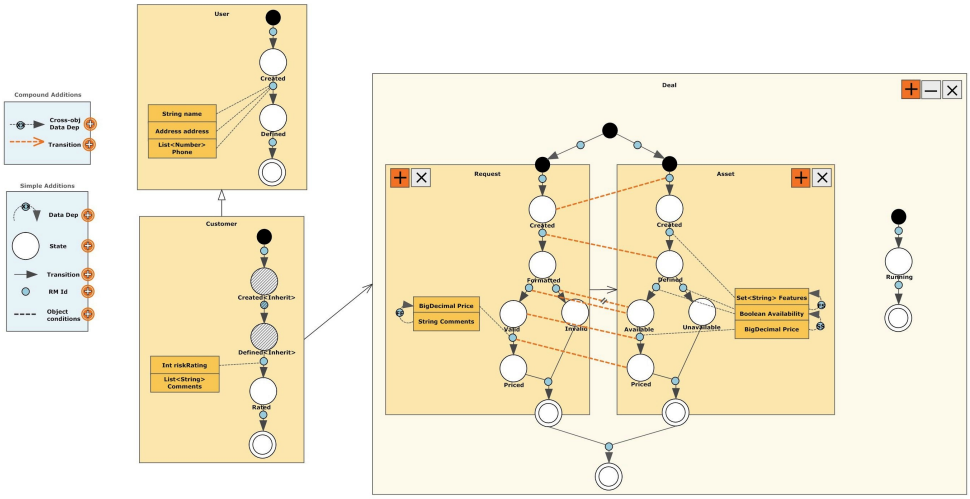


Figure 18.4: Dynamic life-cycle generation and manual definition of synchronization points between object instances

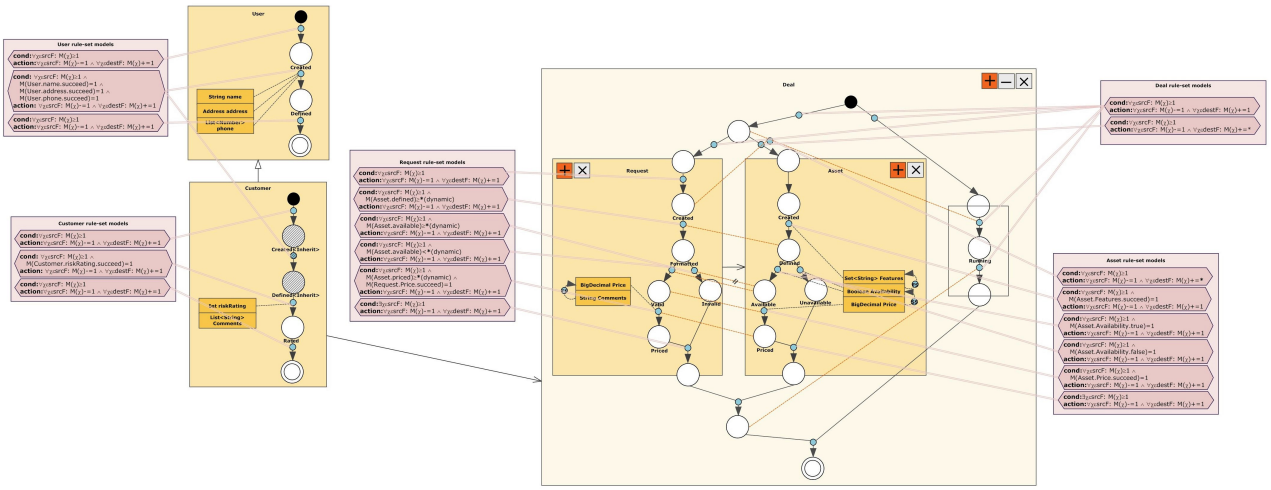


Figure 18.5: Dynamic generation of rule-set models and of an object model sound net

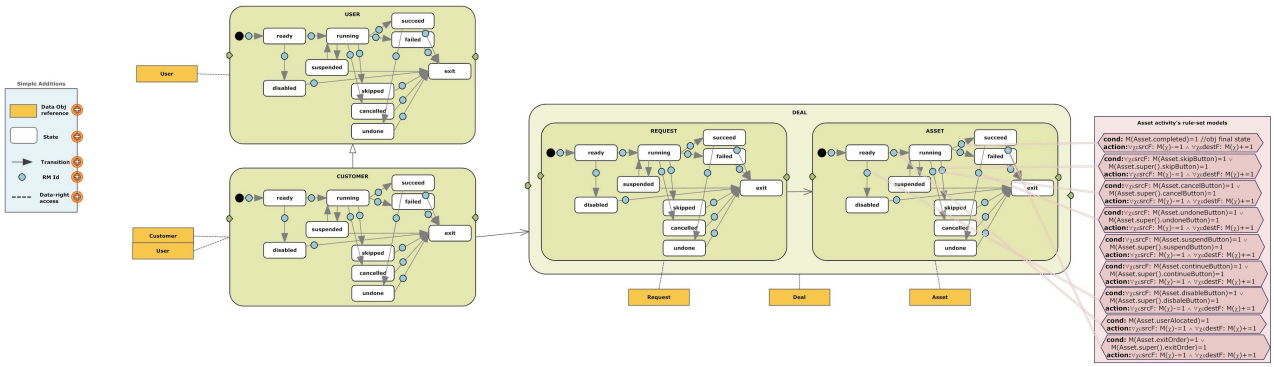


Figure 18.6: Dynamic derivation of activity execution behavior and default data-contexts (using default visibility criteria)

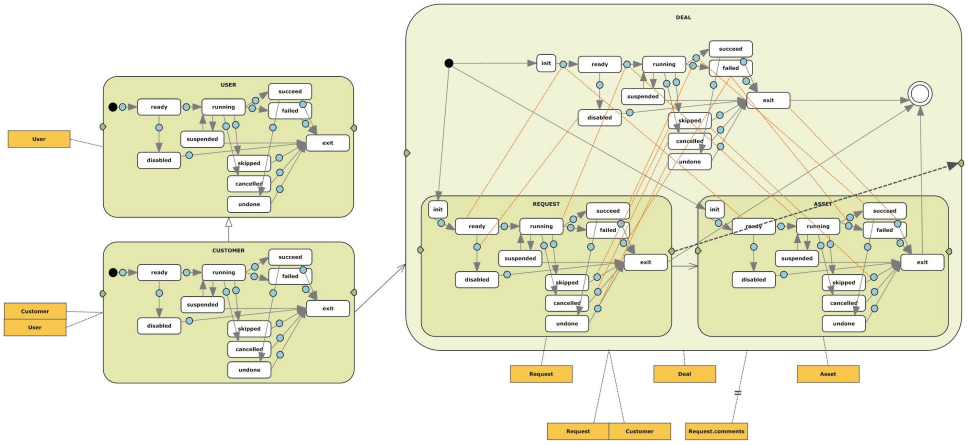


Figure 18.7: Dynamic derivation of compound execution effects and manual definition of local orderings and of data-contexts

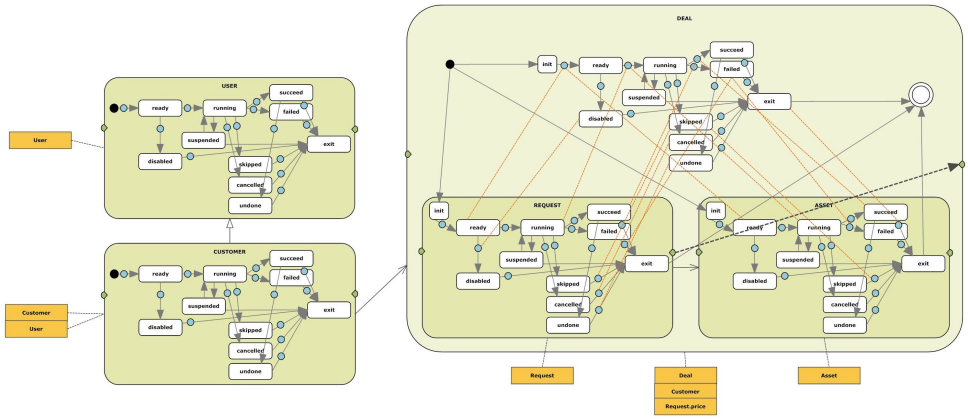


Figure 18.8: Automatic derivation of the data-accessible objects (data-contexts can also be defined using object models)

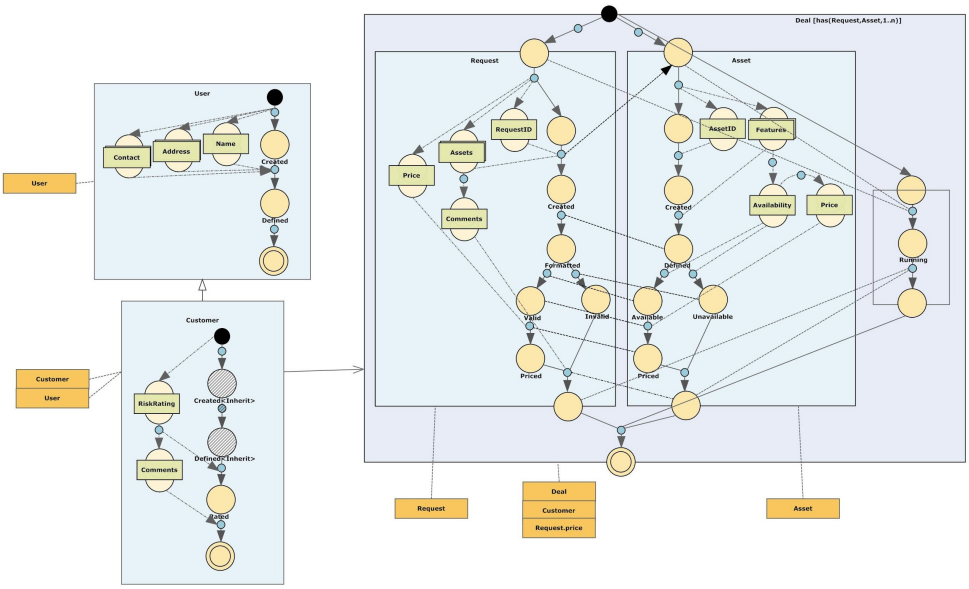


Figure 18.9: Dynamic derivation of object-centered process models based on object models

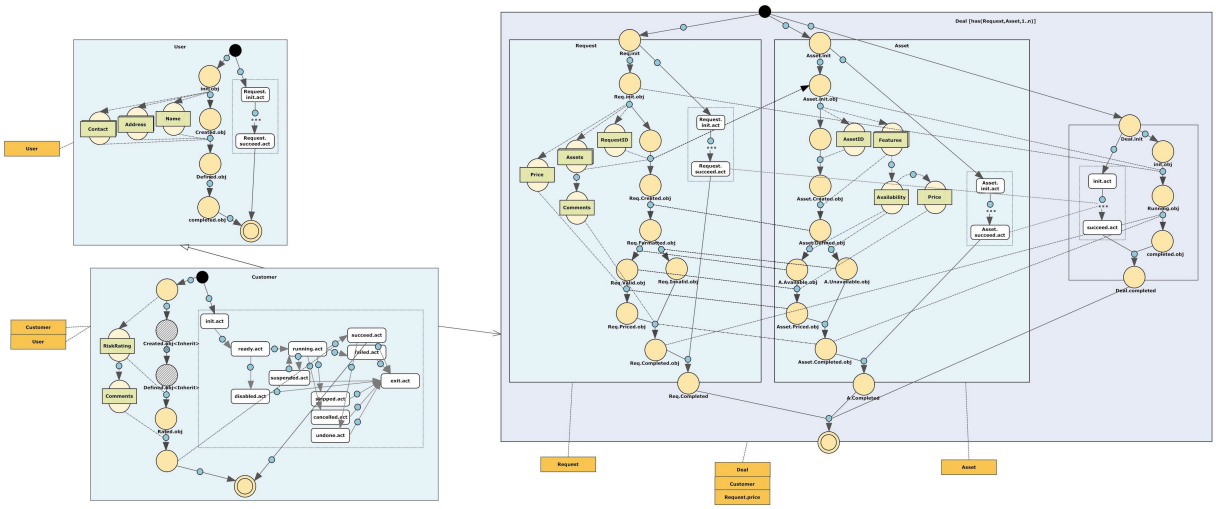


Figure 18.10: Automatic completion of object-centered process models

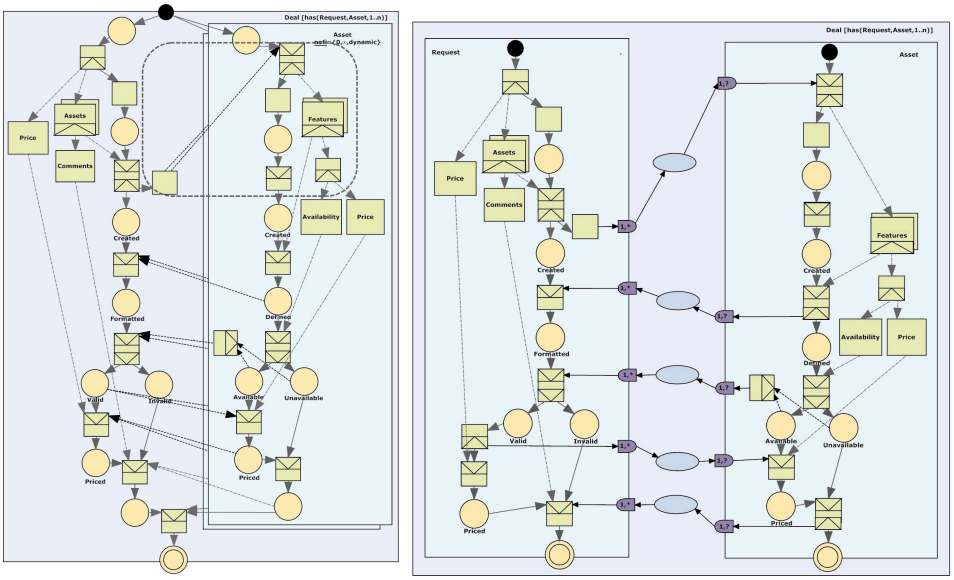


Figure 18.11: The mapping of object-centered process models into plain YAWL models and into a proclets' net

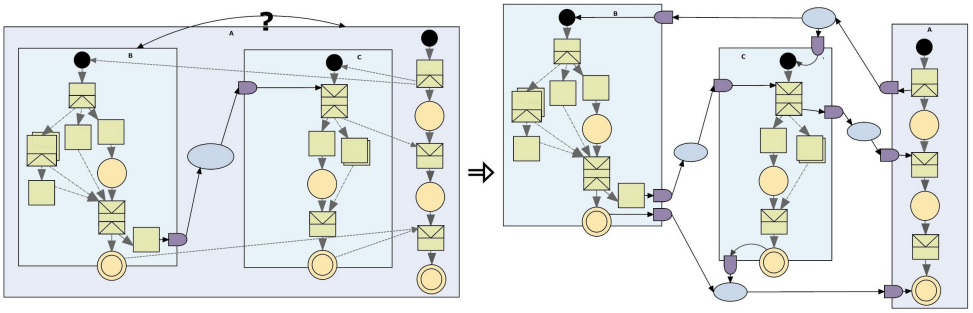


Figure 18.12: Mediator strategy to enable the composition mechanisms within proclets

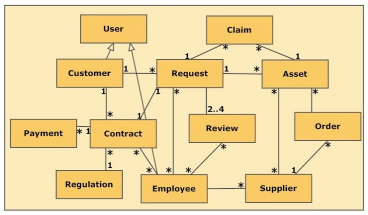


Figure 18.13: Data model for the GF case

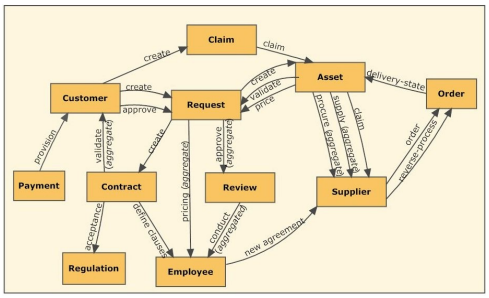


Figure 18.14: Object placeholders defining GF points-of-synchronization

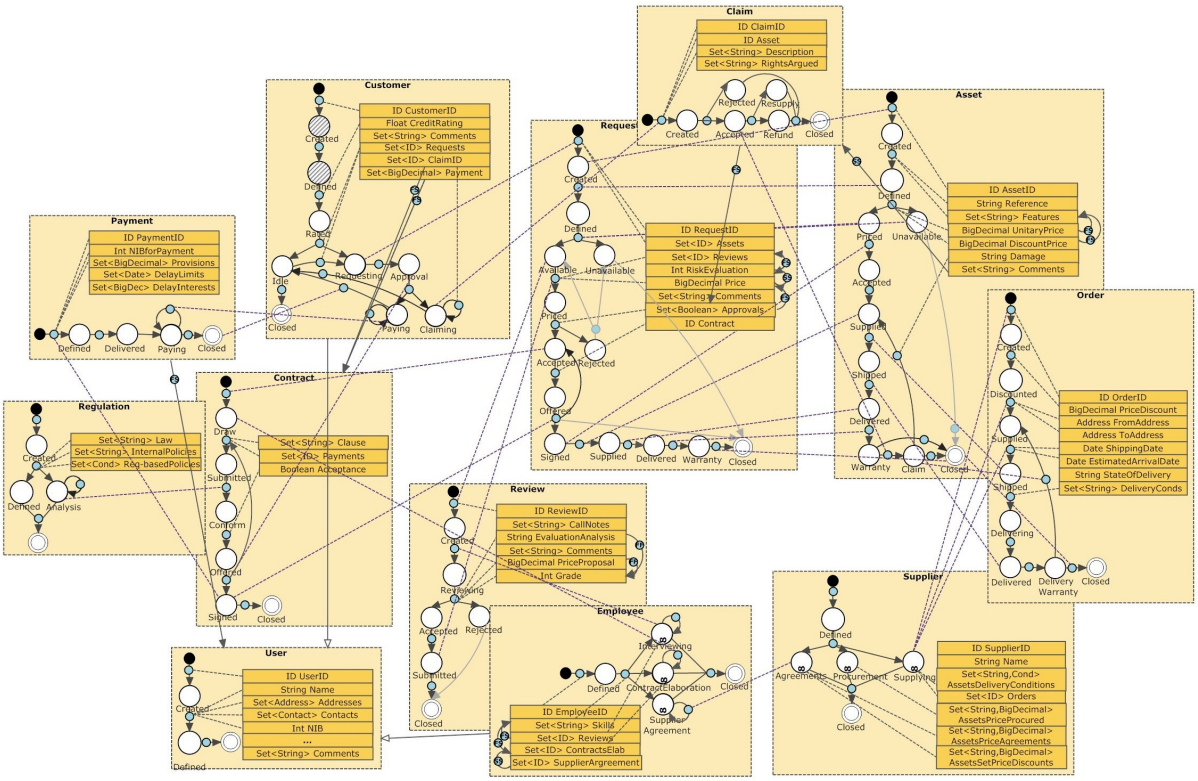


Figure 18.15: GF object model (textual definition or views must be used to restrict complex graphical dependencies)