



HAL
open science

A Grammatical Approach for Distributed Business Process Management using Structured and Cooperatively Edited Mobile Artifacts

Milliam Maxime Zekeng Ndadji

► **To cite this version:**

Milliam Maxime Zekeng Ndadji. A Grammatical Approach for Distributed Business Process Management using Structured and Cooperatively Edited Mobile Artifacts. Software Engineering [cs.SE]. Université de Dschang (Cameroun), 2021. English. NNT: . tel-03243900

HAL Id: tel-03243900

<https://hal.science/tel-03243900>

Submitted on 31 May 2021

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

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

REPUBLIQUE DU CAMEROUN
PAIX - TRAVAIL - PATRIE

UNIVERSITE DE DSCHANG

ÉCOLE DOCTORALE



REPUBLIC OF CAMEROON
PEACE - WORK - FATHERLAND

UNIVERSITY OF DSCHANG

POST GRADUATE SCHOOL

DSCHANG SCHOOL OF SCIENCES AND TECHNOLOGY

UNITE DE RECHERCHE EN INFORMATIQUE FONDAMENTALE, INGENIERIE ET APPLICATIONS (URIFIA)

TOPIC :

A GRAMMATICAL APPROACH FOR DISTRIBUTED BUSINESS PROCESS MANAGEMENT USING STRUCTURED AND COOPERATIVELY EDITED MOBILE ARTIFACTS

This thesis defended in public under the title “*a grammatical approach to peer-to-peer cooperative editing on a service-oriented architecture*” for the award of a
Doctorate/PhD degree in Computer Science

Option: Software Engineering
Speciality: Business Process Management

By

ZEKENG NDADJI Milliam Maxime

Registration Number: **CM04-10SCII755**
Master in Computer Science

Under the co-direction of

TCHOUBE TCHENDJI Maurice
Senior Lecturer, University of Dschang

TAYOU DJAMEGNI Clémentin
Full Professor, University of Dschang

This thesis was defended on 28 May 2021 in front of the examination panel consisting of

| Names, Grade and Affiliation | Role |
|---|------------|
| TIEUDJO Daniel, <i>Full Professor, University of Ngaoundéré</i> | President |
| NKENLIFACK Marcellin Julius A., <i>Associate Professor, University of Dschang</i> | Examinator |
| DJOTIO NDIE Thomas, <i>Associate Professor, University of Yaoundé I</i> | Examinator |
| KOUAMOU Georges-Edouard, <i>Associate Professor, University of Yaoundé I</i> | Examinator |
| TAYOU DJAMEGNI Clémentin, <i>Full Professor, University of Dschang</i> | Reporter |
| TCHOUBE TCHENDJI Maurice, <i>Senior Lecturer, University of Dschang</i> | Reporter |

Year 2021

DEDICATION

*I dedicate this work to all those who one day, saw the efforts they had put into a project (especially an intense love relationship) being wiped out without a logical explanation and who took it upon themselves, to ride the waves unleashed by these storms, came out ten times stronger and developed an incredible desire to live rather than die. Like your daily efforts, this work is in large part the result of the knowledge that suffering has brought to me and thus, it contributes to prove Friedrich Nietzsche's aphorism: "**what doesn't kill you, makes you stronger**".*

ACKNOWLEDGEMENTS

Along the paths I followed during this thesis, I have gained a unique experience and, I hope, the necessary maturity to aspire to the title of Doctor/PhD. Throughout these years of work, I have come to realise that producing a thesis is far from being a solitary labour. In order to go through this tunnel, one must be under the benevolent escort of the Almighty Lord and of souls of good faith. I have benefited from a multiform accompaniment of many persons, both physical and moral. I would like them to find in this section, the expression of my deep gratitude. I therefore thank very warmly:

- The LORD *our God, creator of heaven and earth*: in addition to my days, he offers me every day, the necessary grace and strength to continue to glorify him. Glory be to you, Holy Father!

- Dr TCHOUPÉ TCHENDJI MAURICE, *Senior Lecturer at the Department of Mathematics and Computer Science of the Faculty of Science of the University of Dschang*: he took the time to mentor me throughout this work, inculcating in me his sense of commitment and organisation. Sir, I reiterate that you are my model.

- Pr TAYOU DJAMEGNI CLÉMENTIN, *Head of the Computer Engineering Department of the Institute of Technology - Fotso Victor of Bandjoun*: it was under difficult conditions that he agreed to supervise this work and to reward us with his legendary positivism. I'm more than honoured to be one of your students, Sir.

- All the imminent members of my pre-hearing and defence juries: they agreed to objectively evaluate this work. Thank you for the chance you are giving me, gentlemen.

- Dr PARIGOT DIDIER, *Senior Researcher on Programming Languages at INRIA*: he accompanied me during this thesis by his reviews, his points of view, his

recommendations and his know-how in terms of software programming. I learned a lot from your multiple contributions, Sir.

- The *INRIA/LIRIMA FUCHSIA Associate Research Team*: they welcomed me when I needed it most and gave me a chance to express myself. Working with you guys is a lifelong dream.

- The *lecturers at the University of Dschang*: they contributed enormously to my education. As education is priceless, I can only express my gratitude.

- My *parents*, NDADJI EMMANUEL and MAFFOZEMTSOP MARIE: they gave me everything. I love you infinitely.

- My *brothers, sisters, friends and colleagues (Nani, Chance, Alex, Arnold, Brice, Ariège, Brel, Doris, Audrey, Fabrice, Rodrigue, Virginie, Emeric, Nestor, Preston, Yann, Max, Ange, Lionel, etc.)*: I live only by you and for you (uh, for me too ;-)).

- My dear BANGUKET T. NINA: she proofread and corrected the typos in all my English documents. Now I can write a correct sentence in English, thanks to you.

TABLE OF CONTENTS

| | |
|--|-----------|
| Dedication | i |
| Acknowledgements | ii |
| Table of Contents | iv |
| Abstract | viii |
| List of Acronyms | ix |
| List of Tables | x |
| List of Figures | xi |
| List of Algorithms | xiv |
| General Introduction | 1 |
| The Emergence of Business Process Management | 1 |
| The Mitigated Use of Business Process Management | 3 |
| Our Global Vision | 5 |
| The Challenge Addressed in this Thesis | 6 |
| A Synoptic View of our Methodology and Engineering | 8 |
| The Organisation of this Manuscript | 12 |
| Chapter 1 ► A State of the Art in Business Process Management: the Artifact-Centric Modelling | 14 |
| I.1 - Introduction | 14 |
| I.2 - Key Principles of Business Process Management | 16 |
| I.2.1 - Some Business Process Management Basic Concepts | 16 |
| I.2.1.1 - Some Definitions | 16 |
| I.2.1.2 - An Introductory Example of Business Process | 17 |
| I.2.1.3 - Workflow Typology | 19 |
| I.2.2 - Business Process Management Lifecycle and Key Activities | 22 |
| I.2.2.1 - Business Process Management Lifecycle | 22 |
| I.2.2.2 - The "Model" Activity | 23 |
| I.2.2.3 - The "Enact" Activity | 26 |
| I.3 - Peer to Peer Business Process Management | 29 |
| I.3.1 - The Advent of the Multiagent and Service-Oriented Concepts | 30 |

| | |
|--|-----------|
| I.3.1.1 - The Multiagent Concept | 30 |
| I.3.1.2 - The Service-Oriented Architecture | 32 |
| I.3.2 - Some Existing Distributed WfMS | 35 |
| I.3.2.1 - Some Partially Distributed WfMS | 36 |
| I.3.2.2 - Some Fully Distributed WfMS | 38 |
| I.4 - Artifact-Centric Business Process Management | 44 |
| I.4.1 - Artifact-Centric BPM Basic Concepts | 44 |
| I.4.1.1 - The Aim of Artifact-Centric BPM | 44 |
| I.4.1.2 - How the Artifact-Centric Approach to BPM Works | 45 |
| I.4.2 - Some Existing Artifact-Centric BPM Frameworks | 47 |
| I.4.2.1 - Some Purely Artifact-Centric BPM Frameworks | 47 |
| I.4.2.2 - A Guarded Attribute Grammars Based Framework to Data-Centric Case Management | 50 |
| I.5 - Summary | 54 |
| <i>Chapter II ► A Workflow for Structured Documents' Cooperative Editing : Key Principles and Algorithms</i> | 56 |
| II.1 - Introduction | 56 |
| II.2 - Basic Concepts on Cooperative Editing Workflows | 58 |
| II.2.1 - Real-Time Cooperative Editing Workflows | 58 |
| II.2.2 - Asynchronous Cooperative Editing Workflows | 59 |
| II.2.3 - Badouel and Tchoupé's Cooperative Editing Workflow | 60 |
| II.3 - Tree Automata for Extracting Consensus from Partial Replicas of a Structured Document | 62 |
| II.3.1 - Structured Cooperative Editing and Notion of Partial Replication | 62 |
| II.3.1.1 - Structured Document, Editing and Conformity | 62 |
| II.3.1.2 - Notions of View, Projection, Reverse Projection and Merging | 67 |
| II.3.2 - Reconciliation by Consensus | 69 |
| II.3.2.1 - Issue and Principle of the Solution of Reconciliation by Consensus | 69 |
| II.3.2.2 - Consensus Calculation | 70 |
| II.3.2.3 - Illustration | 77 |
| II.4 - A Software Architecture for Centralised Management of Structured Documents in a Cooperative Editing Workflow | 83 |
| II.4.1 - The Proposed Architecture | 84 |
| II.4.1.1 - Overall Operations | 84 |

| | |
|--|-----------|
| II.4.1.2 - Server Architecture | 84 |
| II.4.1.3 - Client Architecture | 86 |
| II.4.1.4 - The Middleware | 86 |
| II.4.2 - TinyCE v2 | 86 |
| II.4.2.1 - Presentation of TinyCE v2 | 86 |
| II.4.2.2 - Java-Haskell Cross-Fertilisation in TinyCE v2 | 87 |
| II.5 - Summary | 89 |
| | |
| Chapter III ► A Choreography-like Workflow Design and Distributed Execution Framework Based on Structured Mobile Artifacts' Cooperative Editing | 90 |
| III.1 - Introduction | 90 |
| III.2 - Overview of the Artifact-Centric Model Presented in this Thesis | 92 |
| III.2.1 - A Description of the Artifact-Centric Model Presented in this Thesis | 92 |
| III.2.2 - The Running Example: the Peer-Review Process | 94 |
| III.2.2.1 - Description of the Peer-Review Process | 94 |
| III.2.2.2 - Overview of the Peer-Review Process Artifact-Centric Execution using the Model Presented in this Thesis | 95 |
| III.3 - Modelling Artifacts | 96 |
| III.3.1 - Artifacts' Structure | 96 |
| III.3.2 - Target Artifacts and Grammatical Model of Workflow | 97 |
| III.3.3 - Artifact Type and Artifact Edition | 99 |
| III.3.3.1 - Modelling the Information Model of Processes with GMWf | 99 |
| III.3.3.2 - Artifact Type | 100 |
| III.3.3.3 - Artifact Edition | 101 |
| III.4 - Agent and choreography | 103 |
| III.4.1 - Relations between Agent, Actor and Choreography | 103 |
| III.4.2 - Structure of an Agent | 103 |
| III.4.2.1 - The Local Workflow Engine | 103 |
| III.4.2.2 - The Storage Device | 104 |
| III.4.2.3 - The Specialised Editor | 104 |
| III.4.3 - Concepts of Accreditation, Partial Replica of an Artifact and Local GMWf | 105 |
| III.4.3.1 - Concept of Accreditation | 105 |
| III.4.3.2 - Concept of Partial Replica of an Artifact | 107 |

| | |
|---|------------|
| III.4.3.3 - The Need of a Local GMWf | 110 |
| III.4.4 - The Artifact-Centric Choreography | 114 |
| III.4.4.1 - Initial Configuration of an Agent | 114 |
| III.4.4.2 - The Execution Choreography and Agent's Behaviour | 115 |
| III.4.4.3 - The Protocols | 115 |
| III.5 - Illustrating the Choreography on the Peer-Review Process | 123 |
| III.6 - Experimentation | 126 |
| III.6.1 - P2PTinyWfMS: an Experimental Prototype System | 127 |
| III.6.2 - Executing our Running Example under P2PTinyWfMS | 129 |
| III.7 - Related Works and Discussion | 130 |
| III.8 - Summary | 132 |
| General Conclusion | 134 |
| Recall of this Thesis' Challenge and of our Methodological Choices | 134 |
| A Critical Analysis of the Performed Work | 136 |
| Some Perspectives | 139 |
| Bibliography | 143 |
| <i>Appendix A</i> ► Implementation of Some Important Algorithms Presented in this Thesis | 155 |
| <i>Appendix B</i> ► List of Publications Issued from the Work Presented in this Thesis | 158 |

ABSTRACT

In this thesis, we focus on the proposal of distributed workflow systems dedicated to the automation of administrative business processes. We propose an approach to build such systems by relying on the concepts of multiagent systems, Peer to Peer (P2P) architecture, Service-Oriented Architecture (SOA) and structured documents (artifacts) cooperative edition. Indeed, we develop mathematical tools that allow any workflow systems designer, to express each administrative process in the form of an attributed grammar whose symbols represent tasks to be executed, productions specify a scheduling of these tasks, and instances (the derivation trees that conform to it) represent the different execution scenarios leading to business goal states. The obtained grammatical model is then introduced into a proposed P2P system which is in charge of carrying out the completely decentralised execution of the underlying process's instances. The said system orchestrates a process's instance execution as a choreography during which, various software agents driven by human agents (actors), coordinate themselves through artifacts that they collectively edit. The exchanged artifacts represent the system's memory: they provide information on already executed tasks, on those ready to be executed and on their executors. The software agents are autonomous and identical: they execute the same unique protocol each time they receive an artifact. This protocol allows them to identify the tasks they must immediately execute, to execute them, to update the artifact and to disseminate it if necessary, for the continuation of the execution. Moreover, actors potentially have only a partial perception of processes in which they are involved. In practice, this means that certain tasks can be carried out confidentially: this property makes it possible to offer automatic management of administrative processes that is a little closer to their non-computerised management.

Keywords: Administrative Workflows, Artifacts, Peer to Peer, Partial Replication, Business Process Management.

LIST OF ACRONYMS

| | |
|-------------|---|
| AST | Abstract Syntax Tree |
| BPM | Business Process Management |
| BPMN | Business Process Model and Notation |
| CDML | Component Description Meta Language |
| CSCW | Computer-Supported Cooperative Work |
| P2P | Peer to Peer |
| DS(E)L | Domain Specific (Embedded) Language |
| DTD | Document Type Definition |
| GMAWfP | a Grammatical Model of Administrative Workflow Process |
| GMWf | Grammatical Model of Workflow |
| LSAWfP | a Language for the Specification of Administrative Workflow Processes |
| (L)WfE | (Local) Workflow Engine |
| P2PTinyWfMS | a Peer-to-Peer Tiny Workflow Management System |
| SOA | Service Oriented Architecture |
| SON | Shared-Overlay Network |
| TinyCE | a Tiny Cooperative Editor |
| WfM(S) | Workflow Management (System) |
| WF-Net | Workflow Net |
| (WS-)BPEL | (Web Services) Business Process Execution Language |
| XML | eXtensible Markup Language |
| YAWL | Yet Another Workflow Language |

LIST OF TABLES

| | |
|--|-----|
| I - Exhaustive tasks list of a paper validation process in a scientific journal and their respective performers. | 19 |
| II - The transition schemas for the view $\{A, B\}$ | 79 |
| III - The tree automaton associated to the replica $tv1$ | 80 |
| IV - The transition schemas for the view $\{A, C\}$ | 80 |
| V - The tree automaton associated to the replica $tv2$ | 81 |
| VI - The consensual tree automaton. | 82 |
| VII - Accreditations of the different agents taking part in the peer-review process. | 106 |
| VIII - Local GMWf productions of all the agents involved in the peer-review process. | 114 |

LIST OF FIGURES

| | |
|--|----|
| 1 - A synoptic view of our methodology. | 8 |
| 2 - Classification of workflows according to whether they are human-oriented or system-oriented (source (Georgakopoulos, F. Hornick, and P. Sheth, 1995)). | 20 |
| 3 - The three phases of BPM's lifecycle (source (Van Der Aalst, 2013)). | 23 |
| 4 - The four key activities of BPM (source (Van Der Aalst, 2013)). | 24 |
| 5 - Four basic routing constructs (source (Van Der Aalst, 1998)). | 25 |
| 6 - Orchestration diagrams of the peer-review process. | 26 |
| 7 - Reference model of the Workflow Management Coalition (source (WfMC, 1995)). | 28 |
| 8 - The basic Service-Oriented Architecture (source (Papazoglou, 2003)). | 33 |
| 9 - Overview of a P2P application development process with SON (source (Lahcen and Parigot, 2012)). | 35 |
| 10 - An ADEPT environment (source (Jennings, 1996)). | 36 |
| 11 - The workflow execution process in EvE (source (Geppert and Tombros, 1998)). | 38 |
| 12 - The METEOR ₂ architecture (source (Das, Kochut, Miller, Sheth, and Worah, 1997)). | 39 |
| 13 - A P2P workflow architecture (source (Fakas and Karakostas, 2004)). | 41 |
| 14 - A high-level view of SwinDeW's architecture (source (Yan, Yang, and Raikundalia, 2006)). | 42 |
| 15 - Structure of a peer in SwinDeW (source (Yan et al., 2006)). | 43 |
| 16 - Graphical representation of the proclat-based framework (source (Van Der Aalst, Barthelmeß, Ellis, and Wainer, 2001)). | 48 |
| 17 - An example of interoperation using an artifact-centric hub (source (Hull, Narendra, and Nigam, 2009)). | 49 |
| 18 - Active workspace of a clinician (source (Badouel, H elou et, Kouamou, Morvan, and Fondze Jr, 2015)). | 52 |
| 19 - Artifact edition in AWGAG (source (Badouel et al., 2015)). | 53 |

| | |
|---|-----|
| 20 - A BPMN orchestration diagram sketching a cooperative editing workflow of a structured document according to Badouel and Tchoupé. | 61 |
| 21 - Example of an indexed tree. | 63 |
| 22 - Example of a tree that contains buds. | 64 |
| 23 - Example of projections made on a document and partial replicas obtained. | 67 |
| 24 - Example of documents in conflict. | 71 |
| 25 - Document resulting from the consensual merging of the documents in figure 24. | 72 |
| 26 - An edition with conflicts and corresponding consensus. | 78 |
| 27 - Linearisation of a tree ($tv1$): the Dyck symbols '(' and ')' (resp. '[' and ']') have been associated with the grammatical symbol A (resp. B). | 79 |
| 28 - Consensual trees generated from the automaton $\mathcal{A}_{(sc)}$ | 83 |
| 29 - A software architecture (three-tiers) for centralised management of structured documents' cooperative editing workflows. | 85 |
| 30 - Some screenshots showing the creation process of a cooperative editing workflow in TinyCE v2. | 87 |
| 31 - Some screenshots of TinyCE v2 showing the authentication window of a co-author (Auteur1) as well as those displaying the various local and remote workflows in which he is implicated. | 88 |
| 32 - An illustration of consensual merging in TinyCE v2. | 88 |
| 33 - An overview of the artifact-centric BPM model presented in this chapter. | 93 |
| 34 - An overview of the artifact-centric execution of the peer-review process using the model presented in this chapter. | 96 |
| 35 - Target artifacts of a peer-review process. | 97 |
| 36 - An intentional representation of an annotated artifact containing buds. | 101 |
| 37 - An example of artifact edition: the bud C_ω is extended in a subtree. | 102 |
| 38 - Simplified architecture of an agent. | 104 |
| 39 - Example of projections made on an artifact and partial replicas obtained. | 108 |
| 40 - Example of projection of a GMWf according to a given view. | 112 |
| 41 - Activity of an agent in the system. | 116 |
| 42 - Some scenarios to be managed during the expansion. | 121 |
| 43 - Beginning of the peer-review process on the editor in chief's site. | 124 |
| 44 - Continuation of the peer-review process execution on the associated editor's site; the latter receives the request formulated by the editor in chief. | 125 |
| 45 - Continuation of the peer-review process execution on the first referee's site: the request of the associated editor arrives at the first referee. | 126 |

| | |
|---|-----|
| 46 - Continuation of the peer-review process execution: the associated editor receives answers from referees, to requests that he has previously made. | 127 |
| 47 - Continuation and end of the peer-review process execution: the editor in chief receives a response containing referees' contributions, from the associated editor. | 128 |
| 48 - P2pTinyWfMS on the editor in chief's site: presentation of the GMWf (the tasks and their relations, the actors and their accreditations). . . . | 129 |
| 49 - P2pTinyWfMS on the associated editor's site: receipt of editor in chief's request, execution of tasks, expansion-pruning, and diffusion. . | 130 |
| 50 - P2pTinyWfMS on the editor in chief's site: reception of the associated editor's response, execution of tasks, expansion-pruning and end of the case. | 130 |

LIST OF ALGORITHMS

| | |
|--|-----|
| 1 - A Haskell type for automata with exit states. | 73 |
| 2 - A Haskell function to check if two states of a given automaton are in conflict. | 74 |
| 3 - A Haskell function used to check if two given automata admit a consensus. | 74 |
| 4 - Consensus Listing. | 76 |
| 5 - Algorithm to project a given artifact according to a given view. | 109 |
| 6 - Algorithm to project a given GMWf according to a given view. | 111 |
| 7 - Merger protocol executed by an agent i | 117 |
| 8 - Replication protocol executed by an agent i | 117 |
| 9 - Execution protocol executed by an agent i | 118 |
| 10 - Expansion-Pruning protocol executed by an agent i | 119 |
| 11 - Algorithm to search a merging guide. | 120 |
| 12 - Three-way merging algorithm. | 120 |
| 13 - Diffusion protocol executed by an agent i | 123 |

GENERAL INTRODUCTION

CONTENTS

| | |
|--|----|
| The Emergence of Business Process Management | 1 |
| The Mitigated Use of Business Process Management | 3 |
| Our Global Vision | 5 |
| The Challenge Addressed in this Thesis | 6 |
| A Synoptic View of our Methodology and Engineering | 8 |
| The Organisation of this Manuscript | 12 |

The Emergence of Business Process Management

Business Process Management (BPM) has received considerable attention in recent years due to its potential for significantly increasing productivity and saving costs. It is defined by Wil M. P. Van Der Aalst (2013) as *"the discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes"*. BPM aims to improve business processes by focusing on their automation, their analysis, their involvement in decision-making operations (management) and the organisation of work. Hence, BPM is often accompanied by software to manage, control and support business processes: these software systems are called *Workflow Management Systems* (WfMS) (Georgakopoulos et al., 1995; IMA, 2002).

The BPM discipline emerged in the 1980s in a more restrictive form known as *Workflow Management* (WfM). Before WfM was developed, information systems were built from scratch; it means that, all components of such systems had to be programmed, including data storage and retrieval (Van Der Aalst, 1998). Software vendors soon realised that many information systems had similar data management requirements. This generic functionality was therefore outsourced to a data management system. Subsequently, the generic functionality related to user interaction

(forms, buttons, graphics, etc.) was outsourced to user interfaces generators. The trend to outsource recurring functionalities to generic tools has continued in different areas. It is in this context that WfM has been introduced. Precisely, a WfMS automatically manages the process-related aspects (Georgakopoulos et al., 1995; Van Der Aalst, 2013) of information systems. The aim of WfM in the design of information systems is to simplify as much as possible, the modelling and management of the business processes they automate: traditionally, this modelling and management are reduced to the specification of processes using simple graphical languages called *workflow languages*.

Across the years, WfM has evolved into BPM. While WfM focused primarily on the automation of business processes, i.e. it was not fundamentally interested in other issues such as the analysis, verification and maintenance of their specifications, BPM made it its foundation (Van Der Aalst, La Rosa, and Santoro, 2016). With this evolution, many tools and techniques have been developed and have allowed the BPM field to mature. Today, its relevance is recognised by practitioners (users, managers, analysts, consultants and software developers) and academics. The availability of many BPM systems (WfMS) and a series of BPM-related conferences is proof of this (Van Der Aalst, 2013).

With the evolution of WfM and the development of new concepts related to the implementation of collaborative software systems, namely *Peer-to-Peer (P2P) computing*, *multiagent* paradigm and *Service-Oriented Architecture (SOA)*, the way of designing and implementing WfMS has also evolved. There has been a shift from centralised systems implemented according to a reference architectural model (WfMC, 1995), to fully distributed systems offering decentralised workflow execution (Fakas and Karakostas, 2004; Meilin, Guangxin, Yong, and Shangguang, 1998; Yan et al., 2006). New paradigms of specification and management of business processes have been developed too. Among these, we find the *document-centric* (Badouel and Tchoupé Tchendji, 2008; Krishnan, Munaga, and Karlapalem, 2001; Marchetti, Tesconi, and Minutoli, 2005), the *email-based* (Burkhart, Werth, and Loos, 2012; Gazzé, La Polla, Marchetti, Tesconi, and Vivaldi, 2012), the *database-based* (Medina-Mora, Winograd, Flores, and Flores, 1993; Miao, Zhou, and Tao, 2011), the *artifact-centric* (Hull et al., 2009; Lohmann and Wolf, 2010; Nigam and Caswell, 2003), the *data-centric* (Badouel et al., 2015; Damaggio, Hull, and Vaculín, 2013) paradigms, etc. The *artifact-centric* paradigm has been the subject of several studies over the last two decades (Assaf, 2016; Assaf, Badr, and Amghar, 2017; Assaf, Badr, El Khoury, and Barbar, 2018; Boaz, Limonad, and Gupta, 2013; Deutsch, Hull, and Vianu, 2014; Estanol, Queralt, San-

cho, and Teniente, 2012; Hull et al., 2009; Lohmann and Nyolt, 2011; Lohmann and Wolf, 2010); it has been very successful because it has enabled the development of much more flexible workflow languages, that treat process data as first-class citizens, as opposed to the existing languages (BPMN - *Business Process Model and Notation* (Model, 2011) -, YAWL - *Yet Another Workflow Language* (Van Der Aalst, 1998; Van Der Aalst and Ter Hofstede, 2005) -, etc.), that were only concerned with process tasks scheduling and assignment to actors. All these concepts' evolution and this involvement of many technologies in workflow systems' engineering, has made the BPM field, one of the most attractive for software vendors and software engineering researchers.

The Mitigated Use of Business Process Management

The BPM discipline has quickly established itself as an indispensable solution to the process automation needs of large firms, which often involve production lines (Van Der Aalst et al., 2016). Indeed, the workflows encountered in this context are generally highly structured and their tasks are almost entirely automated (executed by machines); this greatly "simplifies" their management by BPM. Another field of application in which BPM tends to naturally impose itself, is that of science. Indeed, the management (storage, distribution, computation, analysis, etc.) of generally very voluminous scientific data, involves several human and material resources that are often distributed across organisations (Bell, Hey, and Szalay, 2009). With the help of cloud computing, BPM in this context, serves to organise these resources for a more efficient management of scientific data (Juve et al., 2013; Ludäscher et al., 2006). In these two application contexts, the systems' complexity due to the large amount of data to be managed and to the time-consuming and intensive computations requiring computer support, as well as the multiplicity and the distributivity of the involved resources, have somehow "imposed" the use of BPM.

Although in its current state BPM could be applied very successfully in the context of organisations with so-called administrative business processes (i.e. processes whose tasks are often semi-automated and therefore require the expertise of human agents - they are then more complex to automate using generic frameworks) (Boukhedouma, 2015), there is less enthusiasm for it (Dumas, 2015; Van Der Aalst et al., 2016); information systems using classical database management concepts and tailored to the use cases involved, are often preferred. In some cases, such information systems are inspired by some BPM concepts or they embed work-

flow engines to make limited use of them: this is the case of Enterprise Resource Planning (ERP) systems such as SAP S/4HANA¹ and Oracle Fusion Applications (OFA)² (Van Der Aalst, 2013; Van Der Aalst et al., 2016). Therefore, the application of "pure" BPM is still limited to specific industries such as banking and insurance. This mitigated use of BPM for the automation of quite common administrative processes can be explained by the following factors:

- *Building process management systems is considerably more "tricky" than building information systems using classical database management.* In database-based information systems, a specific number of processes whose behaviours are known in advance are designed and automated, whereas in process management, systems are designed and implemented to offer a generic management of an arbitrary set of processes with behavioural similarities. To quote Wil M. P. Van Der Aalst (2013), "*BPM is multifaceted, complex, and difficult to demarcate*"; it is therefore not accessible to everyone and requires from its practitioners : great capacities of reasoning, logical abstraction, generalisation and architectural design (Georgakopoulos et al., 1995).
- *Existing BPM solutions are too abstract and generic.* As a result, compared to traditional information systems, they are less suitable for certain applications such as the exclusive management of administrative processes, that have a variety of specifications depending on organisations (Börger, 2012; Van Der Aalst, 2013; Zur Muehlen and Recker, 2013).
- *There is no real consensus on BPM implementation techniques and tools.* BPM is composed by a multitude of paradigms and tools. Even if an effort of standardisation has been made in recent years, WfMS vendors seem to prefer the development of multiple proprietary solutions (Van Der Aalst, 2013); in our opinion, this has the effect of reducing BPM credibility.

Because they offer enormous benefits in terms of time saving, implementation cost and system complexity management, we believe that there is a need to better adapt workflow solutions for administrative processes' management, which seem to be the most frequently encountered (Dumas, Van Der Aalst, and Ter Hofstede, 2005; McCready, 1992; Van Der Aalst, 1998). Naturally, we are not the first to take an interest in this issue; the authors of (Dumas, 2015) and (Van Der Aalst et al., 2016) have established that the potential obstacle to the popularisation of workflow

1. Official website of SAP: <https://www.sap.com/>, visited the 19/03/2020.

2. Official website of Oracle ERP: <https://www.oracle.com/applications/erp/>, visited the 19/03/2020.

solutions, is the imbalance of research work on the different BPM aspects. Indeed, they believe that research has focused too much on BPM specific artifacts (such as process models) rather than on improving organisations' business processes: this does not meet the real use and needs of BPM practitioners. They therefore propose to address now the issue of business process improvement, in order to give even more reasons to organisations regarding the choice of BPM for their processes management. This perspective is shared by a plurality of researchers; hence their growing interest in the new field of *process mining*³ (Van Der Aalst, 2011).

Although the idea of improving business processes by analysing the data produced during their execution is interesting, we believe that it does not answer the question that we are struggling with: *how can we get information system users (organisations and software vendors) to systematically opt for BPM technology to automate their processes ?* Indeed, data analysis is posterior to the users' choice of a technology to produce these data. Moreover, the question of improving business processes seems really interesting for use cases where these processes are very complex; it is not always the case for administrative processes.

Our Global Vision

Because the benefits of BPM are now widely recognised and unanimously accepted, we believe that it would be more interesting to tackle the problem of its popularisation for administrative business processes automation, by improving its technology as it stands at present. Therefore, it would be a matter of:

1. Making more accessible, the automation of administrative business processes using BPM. We believe that if the implementation of workflow systems for administrative business processes is simplified, then more and more software vendors will choose it to produce organisations' information systems.
2. Adapt BPM technology so that it is less abstract and evasive, so that it is closer to that of classical information systems, and so that it can respond to specific problems of organisations.

We believe that one way to achieve these goals, is to apply the *domain-specific software engineering* (Bryant, Gray, and Mernik, 2010) to BPM and thus, to focus on a kind of *Domain-Specific BPM*: that is the application of BPM techniques in

3. Process mining is one of the leading techniques conducting workflows' event data automatic analysis, for possible improvement of its corresponding workflow model (Van Der Aalst, 2013).

specific domains of activity, thus respecting the constraints imposed by them, and offering a framework which best fits the needs of practitioners in these domains. In this thesis therefore, we must use BPM exclusively for the automation of administrative business processes: it is not a matter of reinventing the wheel (even if some new concepts are added) but rather of reproducing and adapting in the more constrained context of administrative business processes management, what is already being done in BPM, while introducing new concepts and making judicious choices to achieve the desired goals.

As far as we know, there are no other scientific studies that have focused exclusively on the automation of administrative processes. On the other hand, there is a growing effort among workflow solution providers to popularise administrative BPM. Precisely, they increasingly offer to organisations, cloud-based and flexible process management solutions such as Metatask⁴, Samepage⁵, Digital Business Transformation Suite⁶, Favro⁷, etc. Let us mention that there are studies focused on adapting BPM technologies to scientific data management exclusively (Domain-specific BPM): these gave birth to the field of scientific workflows (Bell et al., 2009; Juve et al., 2013; Ludäscher et al., 2006).

The Challenge Addressed in this Thesis

In this thesis, we are interested in the automation of administrative business processes (exclusively) using BPM technology. The idea is to use the most up-to-date paradigms already developed in the implementation of workflow systems, to produce a new approach for the specification and execution of administrative business processes. Nowadays, the most common paradigms and concepts used in the implementation of workflow management approaches include:

- *Artifact-centric paradigm*: it is the most successful BPM paradigm of the last two decades; it simplifies the specification of a business process, to the instantiation of a data structure called artifact, whose state evolves into a business goal state when executing process instances.
- *Cooperative edition of documents*: in the artifact-centric paradigm, an artifact can be seen as a structured document containing the execution state of a

4. Official website of Metatask: <https://metatask.io/>, visited the 29/03/2020.

5. Official website of Samepage: <https://www.samepage.io/>, visited the 29/03/2020.

6. Official website of Digital Business Transformation Suite: <https://www.interfacing.com/>, visited the 29/03/2020.

7. Official website of Favro: <https://learn.favro.com/>, visited the 29/03/2020.

process instance at a given time. Because the evolution of the state contained in it is the consequence of actions carried out by the different actors involved in the considered instance's execution, this execution can be assimilated to the cooperative edition of structured documents.

- *Peer to Peer computing*: workflow systems increasingly rely on P2P architectures; as opposed to centralised systems, they facilitate scalability and fault tolerance (Fakas and Karakostas, 2004; Tlili, 2011; Yan et al., 2006).
- *Multiagent paradigm*: the multiagent paradigm was developed to facilitate the creation of distributed systems, especially those based on P2P architectures. It is also increasingly used in workflow systems, where process tasks are now executed by agents that communicate through messages; these agents have a high degree of autonomy to allow better decentralisation of process management.
- *Service-Oriented Architecture*: in decentralised workflow management systems based on P2P architectures, the concept of SOA is generally used to define communication protocols between agents and to increase their autonomy by implementing a loose coupling between them.

We combine these concepts to produce a workflow solution that best suits the automation of administrative business processes. Because this project is much too voluminous to be addressed in the context of a single PhD thesis, we will focus herein, only on the fundamental aspects: administrative process modelling and their distributed execution. One could summarise the main objective of this thesis by saying that it focuses on:

The proposal of a new artifact-centric framework, facilitating the modelling of administrative business processes and the completely decentralised execution of the resulting workflows; this completely decentralised execution being provided by a P2P system conceived as a set of agents communicating asynchronously by service invocation so that, the execution of a given workflow instance is technically assimilated to the cooperative edition of (mobile) structured documents called artifacts.

This justifies the title of this thesis. We should however admit that, a title like "***yet another approach to facilitate administrative workflows design and distributed***

execution using structured and cooperatively edited mobile artifacts" would certainly have better reflected the work done⁸.

A Synoptic View of our Methodology and Engineering

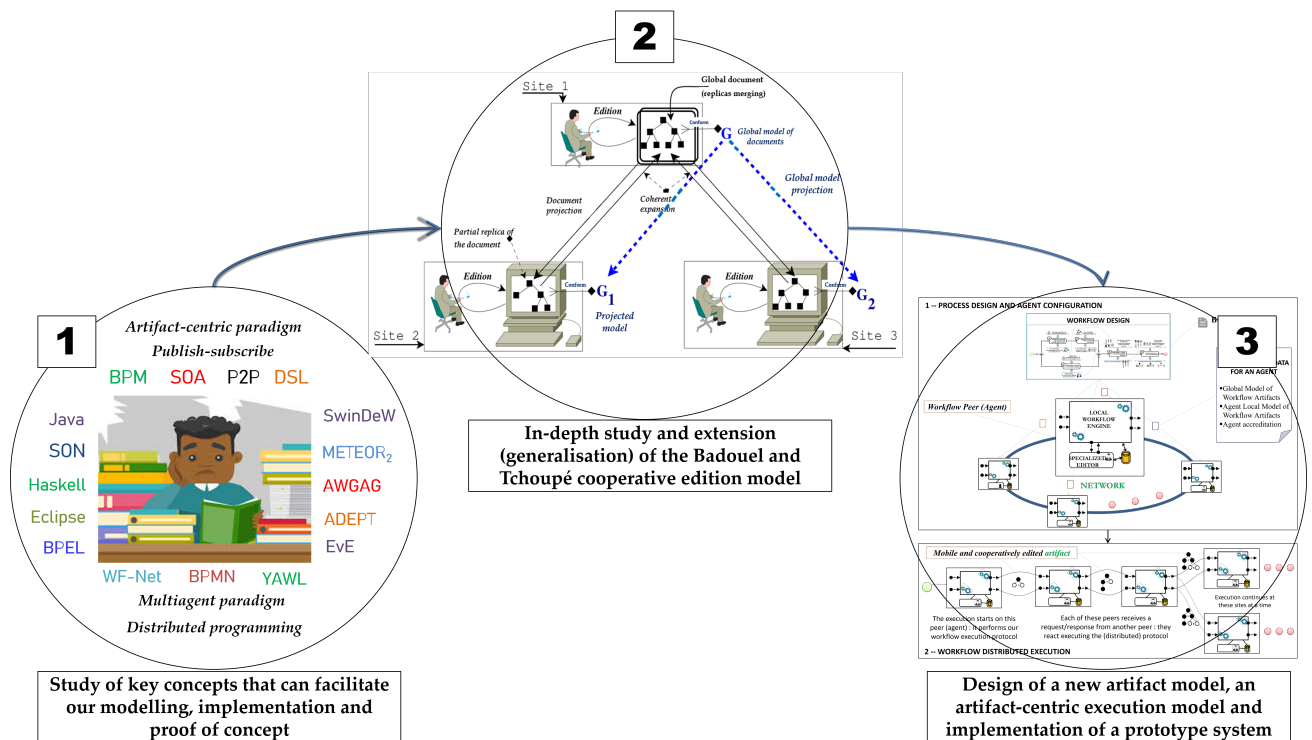


Figure 1 – A synoptic view of our methodology.

Figure 1 summarises the methodology and chronology we used to complete this thesis work. We started by studying a plethora of concepts including, the key concepts presented in chapter I. Then, we reinforced our knowledge on some other concepts, especially those related to the implementation of distributed systems (see fig. 1(1)). Having a better knowledge on BPM, especially on the artifact-centric paradigm, we undertook the design of a new artifacts' model; the search for such a model led us to an extensive study of structured cooperative editing workflows. In particular, we studied the structured documents asynchronous cooperative editing model proposed by Badouel and Tchoupé (2008) so well that, we extended it with three new contributions (see chapter II). Based on the document model manipulated

8. Although this is the title that best suits our work, academic constraints have "forced" us to keep the one currently in use (A Grammatical Approach to Peer-to-Peer Cooperative Editing on a Service-Oriented Architecture). Indeed, it is with this last one that we applied for a thesis in our doctoral school; at the time when we were certain of the judicious title for our work, the legal texts governing the PhD thesis in our institution, no longer allowed us to make changes to the title of our thesis.

by Badouel and Tchoupé, we derived an artifact model (an attributed grammar) that can be used to specify administrative business processes. We then designed a P2P system based on communicating agents using a service-oriented model, capable of executing in a completely decentralised manner, administrative processes specified using the proposed artifact model. To demonstrate the concreteness of our artifact-centric model, we finally implemented a fully functional prototype system allowing to experiment the proposed models on various administrative processes (see chapter III). As a result, we count at least six major contributions in this thesis:

1. An extension of the merge algorithm proposed by Badouel and Tchoupé for the cooperative edition of a structured document, so that it can be applied in the more general case where edition conflicts might appear. It should be noted that this contribution was initiated in our Master's work before being fully matured during our first year of thesis. Some of the elements presented here are therefore part of our Master's dissertation;
2. A generic system architecture that can be used to produce workflow systems for the cooperative editing of structured documents based on the Badouel and Tchoupé extended model;
3. A workflow system prototype referred to as TinyCE v2 (a Tiny Cooperative Editor version 2), coded in Java and Haskell following the proposed system architecture and a novel cross-fertilisation protocol. It allowed us to test all the proposed algorithms related to the merge of documents' replicas;
4. Another tree-based model of "business artifact" for administrative processes modelling, which makes it possible to better assimilate them to structured documents edited cooperatively;
5. A choreography-oriented artifact-centric workflow execution model in which geographically dispersed agents execute the same and unique update (editing of artifact upon receipt) and diffusion (dissemination of updates) protocol;
6. A prototype of a distributed system referred to as P2PTinyWfMS (a Peer-to-Peer Tiny Workflow Management System), allowing to fully experiment the artifact-centric approach investigated in this thesis.

More concretely, in the asynchronous cooperative editing workflow as perceived by Badouel and Tchoupé, each of the co-authors receives in the different

phases of the editing process, a copy of the edited document to insert his contribution. Since the collectively edited document is structured, it may in some cases, be preferable for reasons of confidentiality for example, that a co-author has access only to certain information; meaning that, he only has access to certain parts of the document, belonging to certain given types (*sorts*⁹) of the document model. Thus, the replica t_{q_i} edited by co-author c_i from the site i may be only a *partial replica* of the (global¹⁰) document t , obtained via a *projection operation*, which conveniently eliminates from the global document t , parts which are not accessible to the co-author in question. Badouel and Tchoupé call *view* of a co-author, the set of *sorts* that he can access.

When asynchronous local editions are done on partial replicas, it can be assumed that each co-author has on his site a local document model guiding him in his edition. This local model can help to ensure that for any update $t_{q_i}^{maj}$ of a partial replica t_{q_i} (conform to the considered model), there is at least one document t_c conform to the global model so that $t_{q_i}^{maj}$ is a partial replica of t_c : for this purpose, the local model should be coherent towards the global one¹¹. Thus, because of the edition's asynchronism, the only inconsistencies that we can have when the synchronisation time arrives are those from the concurrent edition of the same node¹² (in the point of view of the global document) by several co-authors: the partial replicas concerned are said to be in conflict. In its first contribution, this thesis proposes an approach of detection and resolution of such conflicts by *consensus*, using a tree automaton said of consensus, to represent all the documents that are the consensus of competing editions realised on different partial replicas.

A structured document t is intentionally represented by a tree that possibly contains buds¹³. Intuitively, synchronise or merge consensually the updates $t_{q_1}^{maj}, \dots, t_{q_n}^{maj}$ of n partial replicas of a document t , consists in finding a document t_c conform to the global model, integrating all the nodes of $t_{q_i}^{maj}$ not in conflict and in which, all the conflicting nodes are replaced by buds. Consensus documents are therefore the largest prefixes without conflicts in merged documents. Technically, the process

9. A *sort* is a datum used to define the structuring rules (syntax) in a document model. Example: a *non-terminal symbol* in a context free grammar, an *ELEMENT* in a *Document Type Definition* (DTD).

10. We designate by *global document* or simply *document* when there is no ambiguity, the document including all parts.

11. Intuitively, a local model of document is *coherent* towards a global model if any partial document t_{q_i} that is conform to it, is the partial replica of at least one (global) document t_c conform to the global model.

12. Manipulated documents are structured, they can be intentionally represented by a tree. Intuitively, a node is an identifiable part in the document (a section, a subsection, an image, a table, ...): it is the instance of a sort.

13. A *bud* is a leaf node of a tree indicating that an edition must be done at that level in the tree. Edit a bud consists to replace it by a subtree using the productions of the grammar of the document.

for obtaining the documents forming part of the consensus is: (1) For each update $t_{q_i}^{maj}$ of a partial replica t_{q_i} , we associate a tree automaton with *exit states* $\mathcal{A}^{(i)}$ recognising the trees (conform to the global model) for which $t_{q_i}^{maj}$ is a projection. (2) The consensual automaton $\mathcal{A}_{(sc)}$ generating the consensus documents is obtained by performing a *synchronous product* of the automata $\mathcal{A}^{(i)}$ with a commutative and associative operator noted \otimes that we define. It is such that: $\mathcal{A}_{(sc)} = \otimes \mathcal{A}^{(i)}$. (3) It only remains to generate the set of trees (or those most representative) accepted by the automaton $\mathcal{A}_{(sc)}$, to obtain the consensus documents.

The concept of structured document as perceived by Badouel and Tchoupé can be adapted to correspond to that of artifact in the sense of artifact-centric workflow systems. In this sense, a structured document can be seen as an artifact (annotated tree) that can be exchanged between the different agents (actors) involved in the execution of a given particular business process case; during its life, it is edited appropriately to make the system converge towards the achievement of one of the considered process's business goals. Its buds materialise the tasks to be executed or which are being executed and, an attributed grammar called the *Grammatical Model of Workflow* (GMWf) is used as *artifact type*. The sorts of a given GMWf represent the process tasks and each of its productions represents a scheduling of a subset of these tasks. When a task is executed on a given site, the corresponding bud in the artifact is closed accordingly; then, one of the GMWf's production having the considered task as left hand side is chosen by the local stakeholder to expand the bud into a subtree highlighting in the form of new buds, the new tasks to be executed. To enrich the notion of access to different parts of artifacts, we add to GMWf, organisational information called *accreditations* (similar to views) that offer a simple mechanism for modelling the generally different perceptions that actors have on processes and their data. The couple (GMWf, accreditations) is then the proposed model of "business artifact".

The execution of an administrative process's instance modelled using the couple (GMWf, accreditations), is a choreography in which the agents are reactive autonomous software components, communicating in peer-to-peer mode and driven by human agents (actors) in charge of executing tasks. An agent's reaction to the reception of a message (an artifact) consists in the execution of a five-step protocol clearly described in this thesis. This protocol allows it to: (1) *merge* the received artifact with the one it hosts locally in order to consider all updates, (2) *project* the artifact resulting from the merger in order to hide the parts to which the local actor may not have access and highlight the tasks to be locally executed, (3) make the local actor *execute* the revealed tasks and thus edit the potentially partial replica

of the artifact obtained after the projection, (4) integrate the new updates into the artifact through an operation called *expansion-pruning* and finally, (5) *diffuse* the updated artifact to other sites for further execution of the process if necessary. The agents' operational capabilities allow that, for the execution of a given process, an artifact created by one of them (initially reduced to an open node), moves from site to site to indicate tasks that are ready to be executed at the appropriate time and to provide necessary data (created by other agents) for that execution; the mobile artifact, cooperatively edited by agents, thus "grows" as it transits through the distributed system so formed.

The Organisation of this Manuscript

The rest of this thesis manuscript consists of four chapters and two appendices organised as follows:

Chapter I - A State of the Art in Business Process Management: the Artifact-Centric Modelling: we present some basic concepts related to the field of BPM, P2P, SOA, as well as the multiagent and artifact-centric paradigms. We also do a survey of some P2P and artifact-centric workflow management systems.

Chapter II - A Workflow for Structured Documents' Cooperative Editing : Key Principles and Algorithms: after a brief presentation of Badouel and Tchoupé's asynchronous cooperative edition model, we present our algorithm for reconciling partial replicas of a structured document as well as a generic architecture of administrative workflow management systems.

Chapter III - A Choreography-like Workflow Design and Distributed Execution Framework Based on Structured Mobile Artifacts Cooperative Editing: we propose a new choreography-like approach to address administrative workflow design and their distributed execution. We also present a fully functional prototype system built according to the proposed approach.

General Conclusion: we summarise our work and present some possible venues for its further development.

Appendix A - Implementations of Some Important Algorithms Presented in this Thesis: we present an implementation of all this thesis' key algorithms in the Haskell language.

Appendix B - List of Scientific Communications Issued from the Work Presented in this Thesis: we list the various scientific papers we produced during this thesis.

I

CHAPTER

A STATE OF THE ART IN BUSINESS PROCESS MANAGEMENT: THE ARTIFACT-CENTRIC MODELLING

CONTENTS

| | |
|---|----|
| I.1 - Introduction | 14 |
| I.2 - Key Principles of Business Process Management | 16 |
| I.3 - Peer to Peer Business Process Management | 29 |
| I.4 - Artifact-Centric Business Process Management | 44 |
| I.5 - Summary | 54 |

I.1. Introduction

The work we are doing in this thesis falls within the domain of Computer-Supported Cooperative Work (CSCW); it is a sub-domain of Software Engineering. Software engineering can be seen as a field of engineering that enables the design, the implementation and the maintenance of quality software systems (Barais, 2005). Research in the field of software engineering has led to the implementation of new design and even programming paradigms (e.g. *object-oriented programming* (Meyer, 2000), *component-based programming* (Heineman and Council, 2001), etc.), new methods and processes (*Object Modeling Technique - OMT* - (Rumbaugh et al., 1991), *Unified Modeling Language - UML* - (Booch, Rumbaugh, Jacobson, et al., 2000), etc.), new technologies (*workflow*, etc.), etc. The importance of software and the ever-increasing complexity of systems keep the field of software engineering among the most important in computer science.

Since the start of the 1980s, workflow technology knows an ever-growing success near companies and researchers in the field of computer-aided production. This success can be justified by the fact that, workflow enables firms to reduce their production costs as well as to quickly and easily develop new products and services: their competitiveness is therefore increased. Workflow technology offers methods and tools (notations, management systems, etc.) for the specification, optimisation, automation and monitoring of business processes (Georgakopoulos et al., 1995; Van Der Aalst, 2015). Workflow technology tools are logically orchestrated within complex systems called *Workflow Management Systems* (WfMS) (IMA, 2002; WfMC, 1995). The purpose of WfMS is not only to automate at best workflows, but also to provide an appropriate framework for facilitating collaboration between actors involved in the execution of a given business process.

The search for Workflow Management (WfM) / Business Process Management¹ (BPM) techniques has been densely conducted over the past two decades and a clear interest has been given to the *artifact-centric* paradigm (Nigam and Caswell, 2003) proposed by International Business Machines Corporation (IBM). This one, revisited in several works (Assaf, 2016; Assaf et al., 2017, 2018; Boaz et al., 2013; Deutsch et al., 2014; Estanol et al., 2012; Hull et al., 2009; Lohmann and Nyolt, 2011; Lohmann and Wolf, 2010), proposes a new approach to BPM by focusing on both automated processes (tasks and their sequencing) and data manipulated through the concept of "*business artifact*" (*artifact-centric modelling*).

In this chapter which serves as a state of the art, we present some key notions related to BPM in general and to the artifact-centric paradigm in particular, in order to make it easier to better apprehend the concepts handled in this manuscript. In Section I.2, we define some basic concepts such as *business process*, *workflow*, *workflow management*, etc., then we present some workflow classification approaches as well as an overview of their standardised automation using BPM. In section I.3, we conduct a review of Peer to Peer (P2P) approaches to BPM. In section I.4, we focus on the artifact-centric paradigm to BPM; we present the two general approaches to its implementation (*orchestration* and *choreography*) as well as some frameworks implementing it from the literature. Section I.5 is dedicated to a summary of the explored concepts and a smooth transition to the next chapter.

1. BPM can be considered as an extension of classical Workflow Management (WfM) systems and approaches (Van Der Aalst, 2015).

I.2. Key Principles of Business Process Management

I.2.1. Some Business Process Management Basic Concepts

Research in the CSCW field focuses on the role of computers in collaborative work (Schimdt and Bannon, 1992). These have given rise to numerous softwares called *CSCW systems* or *groupware*. CSCW systems communicate through networks and provide functionalities facilitating exchanges, coordination, collaboration and co-decision between the actors of a given collaborative work; they thus defy the space and time constraints to which collaborative work is subjected. Indeed, with the help of such systems, actors can either operate on the same site and thus manipulate the same objects (*centralised approach*), or they can operate on geographically distant sites (*distributed approach*); in this case, the objects they manipulate are replicated on the different sites and synchronised at the appropriate time (Grudin, 1994; Johansen, 1988; Penichet, Marin, Gallud, Lozano, and Tesoriero, 2007). In the same vein, they can act at the same time (*synchronous approach*) or at completely different times and sometimes independently of the actions carried out by others (*asynchronous approach*) (Grudin, 1994; Johansen, 1988; Penichet et al., 2007). CSCW systems are often referred to as *workflow systems*. However, it should be noted that workflow is an extension and a generalisation of CSCW to business processes' automation.

I.2.1.1. Some Definitions

A *business process* can be informally defined as a set of tasks ordered following a specific pattern and whose execution produces a service or a particular business goal (Georgakopoulos et al., 1995). When such a process is managed electronically, it is called *workflow*. The purpose of workflow is to streamline, coordinate and control business processes in an organised, distributed and computerised environment. The peer-review validation of an article in a scientific journal is a common example of business process. Descriptions of it can be found in (Badouel, H elou et, Kouamou, and Morvan, 2014; Rowland, 2002; Van Der Aalst et al., 2001). As in most literature works, most of the time, we will use the terms "business process" and "workflow" as synonyms in the rest of this manuscript.

The *Workflow Management Coalition*² (WfMC) (WfMC, 1995) defines *Workflow Management* (WfM) as the modelling and computer management of all the

2. The growing reputation of workflow led to the creation, in 1993, of the *Workflow Management Coalition* (WfMC) as the organisation responsible for developing standards in this field. Official website of the WfMC: <https://www.wfmc.org/>.

tasks and different actors involved in executing a business process. WfM is achieved using *Workflow Management Systems (WfMS)*: these are complex systems with the aim of automating at best workflows by providing an appropriate framework to facilitate collaboration between actors involved in business processes' execution (Dumas, La Rosa, Mendling, and Reijers, 2018; Georgakopoulos et al., 1995; Van Der Aalst, 2013, 2015). WfMS are composed of logically orchestrated tools to specify, to optimise, to automate and to monitor business processes (Dumas et al., 2018; WfMC, 1995). Technically, the management of a process according to WfM is done in two phases (Divitini, Hanachi, and Sibertin-Blanc, 2001):

1. the *process modelling phase*: the process is studied and then specified using a language (usually graphical) called *workflow language*. The resulting specification is called *workflow model*;
2. the *process instantiation and execution phase*: the workflow model is introduced into a WfMS which then instantiates and orchestrates the execution of the underlying process.

Since WfMS are pre-engineered standalone systems, WfM simplifies business processes' automation to their specifications in *workflow languages*.

WfM primarily focuses on business processes' automation. It is not fundamentally concerned with other issues such as the analysis, the verification and the management (maintenance) of workflow (models) unlike BPM, which made these its foundation (Van Der Aalst et al., 2016). BPM is the discipline that combines knowledge from information technology and knowledge from management sciences and applies this to operational business processes (Van Der Aalst, 2013). BPM can be seen as an extension of WfM as it primarily supports WfM and provide additional tools to improve business processes. For this, we have chosen to use the expression BPM rather than WfM (which tends to disappear) in the context of this work. It should be noted however that our contributions (chapter II and III) could be perfectly presented as part of restricted WfM (we are not interested in the management and improvement of workflow models).

1.2.1.2. An Introductory Example of Business Process

BPM is an important technology because it simplifies the automation of business processes which are the foundation of how companies and organisations operate. Business processes can be found everywhere. The examples are diverse and include the following:

- The design and development of a software by a team (especially when members are geographically dispersed) (Imine, 2006);
- The simultaneous writing of a scientific paper or the documentation of a product by several researchers (cooperative editing) (Imine, 2006);
- The follow-up of a medical file (Chaâbane, Bouzguenda, Bouaziz, and Gargouri, 2007);
- The student registration process in a faculty;
- The withdrawal of a large sum of money from a bank teller;
- The procedure for taking holidays in a government institution;
- The procedure for claiming damages from an insurance company;
- The peer-review process (Badouel et al., 2014; Rowland, 2002; Van Der Aalst et al., 2001).

The peer-review process presents all the characteristics of the type of processes (*administrative processes*) studied in this manuscript. Then, we will use it as an illustrative example along the whole manuscript (running example). Our description of this process is inspired by those made in (Badouel et al., 2014; Rowland, 2002; Van Der Aalst et al., 2001):

Example 1 *The peer-review process (running example):*

The process is triggered when the editor in chief receives a paper for validation submitted by one of the authors who participated in its drafting.

- *After receipt, the editor in chief performs a pre-validation after which, he can accept or reject the submission for various reasons (subject of minor interest, submission not within the journal scope, non-compliant format, etc.);*
- *If the submission is rejected, he writes a report then notifies the corresponding author and the process ends; in the other case, he chooses an associated editor and sends him the paper for the continuation of the validation;*
- *The associated editor prepares the manuscript, forms a referees committee (two members in our case) and then triggers the peer-review evaluation process;*
- *Each referee reads, seriously evaluates the paper and sends back a message and a report to the associated editor;*

- *After receiving reports from all the referees, the associated editor takes a decision and informs the editor in chief who sends the final decision to the corresponding author.*

From this description, it is easy to identify all the tasks to be executed, their sequencing, actors involved and the tasks assigned to them. For this case, four actors are involved: an editor in chief (*EC*) who is responsible for initiating the process, an associated editor (*AE*) and two referees (*R1* and *R2*). A summary of tasks assignment is presented in table I. We have associated symbols with tasks so that we can easily manipulate them in diagrams.

Table I – Exhaustive tasks list of a paper validation process in a scientific journal and their respective performers.

| Tasks | Associated Symbols | Executors |
|---|------------------------------|------------------------------|
| Receipt, pre-validation of a submitted paper and possible choice of an associated editor to lead peer-review evaluation | <i>A</i> | <i>EC</i> |
| Drafting of a pre-validation report informing on the reasons for the immediate rejection of the paper | <i>B</i> | <i>EC</i> |
| Sending the final decision (acceptance or rejection of the paper) to the author | <i>D</i> | <i>EC</i> |
| Study, eventually formatting of the paper for the examination by a committee | <i>C</i> | <i>AE</i> |
| Constitution of the reading committee (selection of referees) and triggering the peer-review evaluation | <i>E</i> | <i>AE</i> |
| Decision making (paper accepted or rejected) from referees evaluations | <i>F</i> | <i>AE</i> |
| Evaluation of the manuscript by the first (resp. second) referee | <i>G1</i> (resp. <i>G2</i>) | <i>R1</i> (resp. <i>R2</i>) |
| Drafting of the after evaluation report by the first (resp. second) referee | <i>H1</i> (resp. <i>H2</i>) | <i>R1</i> (resp. <i>R2</i>) |
| Writing the message according to evaluation by the first (resp. second) referee | <i>I1</i> (resp. <i>I2</i>) | <i>R1</i> (resp. <i>R2</i>) |

1.2.1.3. Workflow Typology

The authors of (Georgakopoulos et al., 1995) conduct a very interesting study on the classification of workflows in which, they report the lack of a commonly accepted approach to categorising workflows. There are therefore several approaches to workflow classification in the literature.

The classification of workflows according to the nature and behaviour of automated processes is one of the most commonly found in the literature. According to it, workflows are divided into three groups: *production* workflows, *administrative* workflows and *ad-hoc* workflows (McCready, 1992; Van Der Aalst, 1998). Production workflows are those that automate highly structured processes that undergo very little (or no) change over time: all the scenarios are known in advance and most of the tasks are carried out by systems. This is the case for processes in industrial production lines. Administrative workflows apply to variable processes for which all cases are known; that is, tasks are predictable and their sequencing rules are simple and clearly defined. In these, changes are more frequent than with production workflows and human actors are more involved in the execution of tasks. In particular, this type of workflow brings considerable added value to public administration organisations whose business is focused on administrative routines (Boukhedouma, 2015). Our running example, the peer-review process, is an administrative process. In the work presented in this manuscript, we are interested in this type of workflows. These are opposite of ad-hoc workflows, which automate occasional processes for which it is not always possible to define the set of rules in advance. Processes are therefore only partially specified and may undergo many updates over time.

The workflows' classification made in (Georgakopoulos et al., 1995) is orthogonal to the above-mentioned one (they can be used together); it is more concerned with tasks' automation degree. The authors classify workflows based on a measurement system, represented by a continuum ranging from *human-oriented* workflows to *system-oriented* workflows as shown in figure 2. The first type (human-oriented

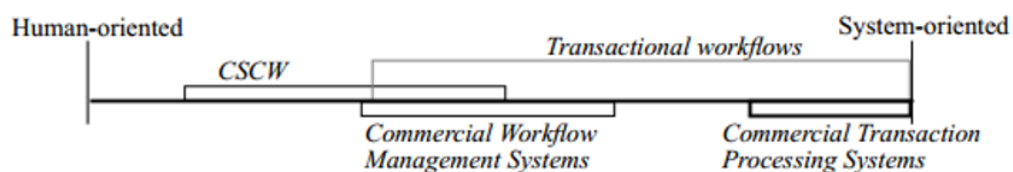


Figure 2 – Classification of workflows according to whether they are human-oriented or system-oriented (source (Georgakopoulos et al., 1995)).

workflows) includes workflows in which humans collaborate to perform tasks and to coordinate themselves; in these, humans are responsible for ensuring the validity and consistency of the exchanged data and of the workflow's results. The second type of workflows (system-oriented workflows) refers to those in which the use of computer systems to perform tasks is unavoidable because, they involve complex

data and computationally-intensive operations. According to this classification system, human-oriented workflows are the ones we are interested in.

In (Dumas et al., 2005), the authors refine the two above classification frameworks. Concerning the refinement of the one that classifies workflows according to the nature and behaviour of automated processes (McCready, 1992), their classification framework distinguishes *unframed*, *ad hoc framed*, *loosely framed*, and *tightly framed* workflows. A workflow is said to be unframed if there is no explicit workflow model associated with it; its execution is strongly conducted by its actors. When actors play a crucial role (no longer limited to the simple execution of tasks, but also including the explicit choice of the control flow, the adjustment of control and data flows, etc.) in the execution of a workflow, it is said to be *user-centric* (Badouel et al., 2015). This is the case for workflows being automated by groupware³. In the case of ad hoc framed workflows, workflow models are defined a priori but, they frequently change. A workflow is said to be loosely framed when it is defined by a workflow model describing the "right way of doing things", while allowing its actual executions to deviate from this way; this is the preferred type of workflow handled by *Case Management Systems* (Van Der Aalst, 2013) (see sec. I.4.2.2). Finally, a tightly framed process is one which consistently follows a defined process model.

Concerning the classification framework of (Georgakopoulos et al., 1995), authors of (Dumas et al., 2005) refine it and consider three types of workflows: *Person-to-Person*, *Person-to-Application*, and *Application-to-Application* workflows. Person-to-Person workflows are those for whom all the tasks require human intervention. Application-to-Application workflows are their opposite; in these, all the tasks are executed by software systems. Person-to-Application workflows are in the middle; they involve both human-oriented tasks and system-oriented tasks. Practically, most of workflows are of this category.

Nowadays, some scientific works require increasingly complex and data-intensive simulations and analysis. Scientific data management is therefore a major challenge (Bell et al., 2009) with a high level of complexity. Workflow technologies are increasingly used to manage this complexity (Juve et al., 2013). These are responsible for scheduling computational tasks on distributed resources, managing dependencies between tasks and staging data sets in and out of runtime sites. The resulting workflows are called *scientific workflows* and are usually based on a middleware infrastructure (*Grid* or *Cloud*). Ideally, the scientist should be able to

3. Groupware systems are computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment (Ellis, Gibbs, and Rein, 1991).

integrate almost any scientific data resource into such a workflow during analysis, inspect and visualise the data on-the-fly as it is computed, make parameter changes as needed and re-run only the affected components, and capture sufficient meta-data in the final products so that, scientific workflow executions help to explain the results and make them reproducible. Thus, a scientific workflow system becomes a scientific problem-solving environment, adapted to an increasingly distributed and service-oriented infrastructure (Grid or Cloud) (Ludäscher et al., 2006).

There are many other types of workflows in the literature. We can mention on the fly, *service-oriented* workflows (Piccinelli, Finkelstein, and Williams, 2003; Yongyi, Youjie, and Hui, 2009), *structured* workflows (Eder and Gruber, 2002; Kiepuszewski, Ter Hofstede, and Bussler, 2000; Liu and Kumar, 2005), etc. We do not present them here because they are not of great interest to the work we are doing for this thesis. We invite the interested reader to take a look at the few works mentioned above.

1.2.2. Business Process Management Lifecycle and Key Activities

A high-level view of the BPM discipline reveals that, its lifecycle consists of three phases on which it is possible to iterate indefinitely: the *(re)design*, *implement/configure*, and *run & adjust* phases (Van Der Aalst, 2013) (see fig. 3). During its lifecycle, four key activities namely *model*, *enact*, *analyse*, and *manage* (see fig. 4) are carried out (Van Der Aalst, 2013). In this section, we examine what is done during these different activities; we mainly focus on the *"model"* and the *"enact"* activities: they are the only ones common to BPM and WfM and thus, they are of relevant interest for the work presented in this manuscript.

1.2.2.1. Business Process Management Lifecycle

The automation of a given process using BPM starts with its modelling using one or more workflow languages (Dumas et al., 2018). This *"model"* activity is initiated during the *"(re)design"* phase of the BPM lifecycle. The workflow models obtained during this activity can be analysed (the *"analyse"* activity) either by simulations or by using model checking⁴ algorithms (to verify models' soundness): this type of analysis is said to be *model-based*. As shown in figure 3, the *(re)design* phase is followed by the *"implement/configure"* phase in which, the workflow models obtained in the previous phase are converted, if necessary, into executable workflow models and then, used to configure the process execution

4. Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model (Baier and Katoen, 2008).

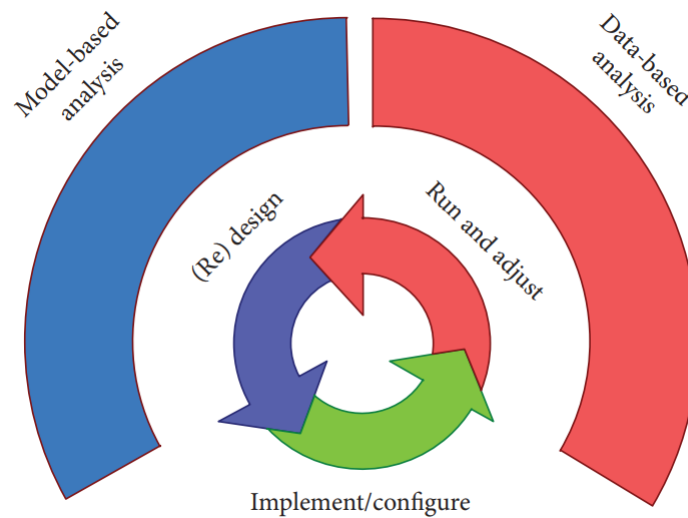


Figure 3 – The three phases of BPM's lifecycle (source (Van Der Aalst, 2013)).

environment (the WfMS): this is where the "model" activity ends. After the "implementation/configure" phase, comes the run & adjust phase. During this last phase, the workflow is instantiated, executed and managed (adjusted) according to the scenarios foreseen when modelling the underlying process and when designing the host WfMS: these are the purposes of the "enact" and "manage" activities. Moreover, when a workflow instance is running, produced and logged data can be analysed (to discover possible bottlenecks, waste, and deviations) for possible improvement of its corresponding workflow model: this other type of analysis/monitoring is said to be *data-based*; during the last decade, *process mining* (Van Der Aalst, 2011) has emerged as one of the leading techniques conducting *data-based analysis*. If enough possible improvements to the workflow model are detected, the cycle can restart to apply them.

1.2.2.2. The "Model" Activity

Basic concepts

Process modelling is a crucial activity in WfM/BPM. As mentioned above (sec. 1.2.2.1), it is done using dedicated languages called *workflow languages*. Several workflow languages have already been developed. Among the most well-known are the BPMN standard (Model, 2011) based on statecharts, the UML activity diagrams language (Booch et al., 2000), the WF-Net (*Workflow Net*) language (Van Der Aalst, Jörg, and Oberweis, 2003) which uses a formalism derived from that of Petri nets, the YAWL language (Van Der Aalst and Ter Hofstede, 2005) which is an extension of WF-Net and so forth. Some of these languages (BPM, UML activity diagrams) are *informal* (i.e. they do not have a well-defined semantics and

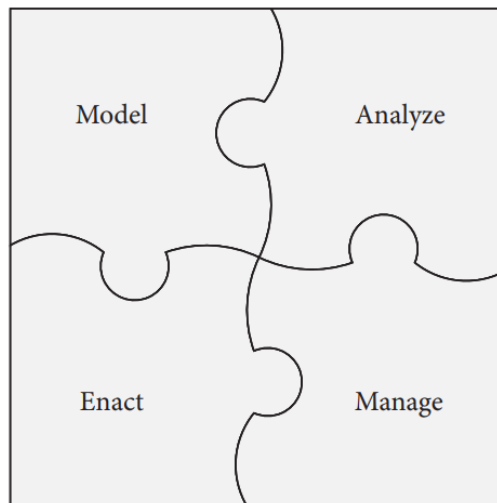


Figure 4 – The four key activities of BPM (source (Van Der Aalst, 2013)).

do not allow for analysis (Van Der Aalst, 2013; Zur Muehlen and Recker, 2013)) while others (WF-Net, YAWL) are based on powerful mathematical (*formal*) tools (Petri nets). Nevertheless, they all allow to express in a diagram (called a *workflow model*), the tasks that make up a given process and the control flow between them. More precisely, workflow languages allow to describe the behaviour of processes through the representation (among others) (Grigori, 2001) of :

- Tasks that make up the main part of the process;
- Information and resources relating to the various tasks;
- Sequences or dependencies between those tasks;
- Trigger and termination events for the tasks.

Tasks are the base of any workflow; a *task* is the smallest unit of hierarchical decomposition of a process. A task represents any work that is performed within a process. It consumes time, one or more resources, requires one or more input objects and produces one or more output objects. You can find examples of tasks in our running example (sec. 1.2.1.2).

From a workflow point of view, the term *resource* refers to a system or a human who can execute a task. It is also known as *actor*, *participant*, *stakeholder*, *agent* or *user* depending on the context. Resources can be grouped according to various characteristics, to form either a *role* or an *organisational unit* (Grigori, 2001). A *role* is a group of resources with the same functional capabilities, while an *organisational unit* is a set of resources (or class of resources) that belong to the same structure (department, team, service, cell, etc.).

Routing patterns

To achieve its objectives, any workflow language must, for a given process, allow to express at least its tasks and their *routing (control flow)*. The task control flow is commonly referred to as the *lifecycle (process) model* of the process under study (Divitini et al., 2001; Hull et al., 2009). There are a number of routing patterns identified in the literature as basic ones: these are *sequential*, *parallel*, *alternative* or *conditional* and *iterative* routings (see fig. 5) (Van Der Aalst, 1998).

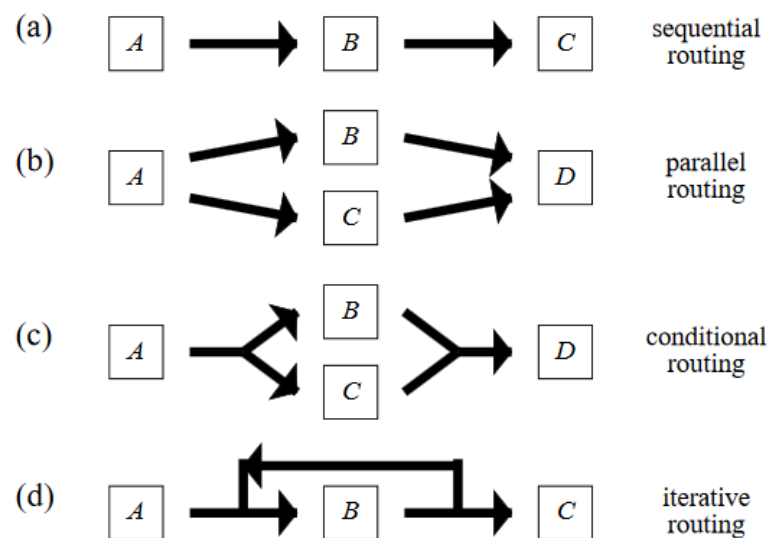


Figure 5 – Four basic routing constructs (source (Van Der Aalst, 1998)).

- Sequential routing expresses the fact that tasks must be executed one after the other (task *A* before tasks *B* and *C* in figure 5(a));
- Parallel routing is used to specify the potentially concurrent execution of certain tasks. Tasks *B* and *C* in figure 5(b) can be executed at the same time; in this case, tasks *A* and *D* are considered as *gateways*: *A* is said to be an *AND-Split* gateway while *D* is an *AND-Join* gateway;
- With alternative routing, one can model a *decision*: i.e. the choice to execute one task rather than another at a given time. In figure 5(c), tasks *B* and *C* cannot be both executed; for this case, *A* is called an *OR-Split* gateway and *D* is an *OR-Join* gateway;
- In some cases, it is necessary to execute a task multiple times. In figure 5(d) task *B* is executed one or more times.

The search for more advanced and expressive routing patterns has been the subject of many studies (Börger, 2012; Van Der Aalst and Ter Hofstede, 2012). The interested reader is invited to consult the few references mentioned above to find out more.

When a given workflow language only allows to specify the routing of the processes' tasks, when it is not interested in modelling the consumed and produced data during tasks execution, and when it only gives a secondary role to the processes' users, it is said to be *process-centric*. This is the case for all the previously mentioned languages (BPMN, WF-Net, UML activity diagrams and YAWL). This type of workflow language is often referred to as "*traditional workflow language*".

Examples of workflow models

Figure 6 shows the orchestration diagrams corresponding to the graphical description of the peer-review process (see its textual description in sec. I.2.1.2) using the process-centric notations BPMN and WF-Net. The graphical notations equivalent to sequential flow, {And, Or}-Splits and {And, Or}-Joins are well represented. Each diagram resumes the *main scenarios* of this process.

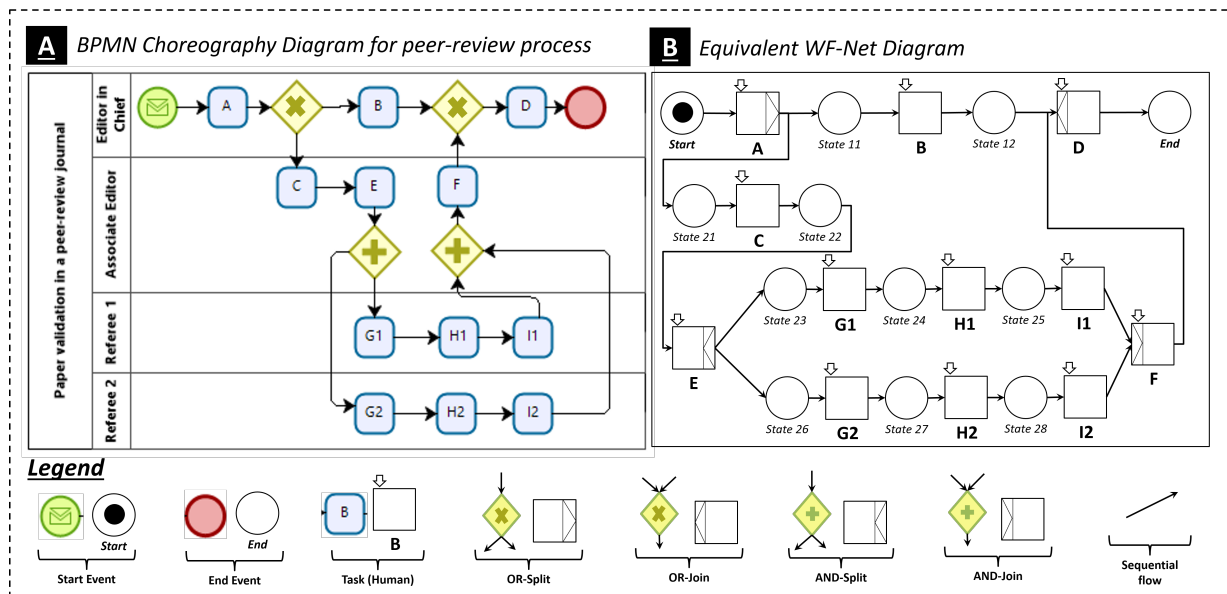


Figure 6 – Orchestration diagrams of the peer-review process.

I.2.2.3. The "Enact" Activity

Overview

The "enact" activity takes as input, the workflow models (specifications) obtained during the model activity. If these models are executable (i.e. they have been coded in more technical languages taking into account implementation de-

tails) then they are directly introduced into a WfMS suitably installed at the different workflow execution sites; otherwise, they are first converted into executable models then, they are introduced into the WfMS. There are several languages for producing executable workflow models. These are usually proprietary and provided by WfMS designers. Of these languages, (*Web Services*) *Business Process Execution Language* ((WS-)BPEL) is the standard⁵ and is well compatible with BPMN (Leymann, 2010; Ouyang, Dumas, Ter Hofstede, and Van Der Aalst, 2006; White, 2005).

Once the WfMS is properly configured using workflow models, it can create workflow instances and properly orchestrate their execution. To do this, WfMS must coordinate (according to workflow models) the execution of a set of tools and applications offering various services. In the 1990s, the WfMC developed and proposed an architectural *reference model* for the implementation of WfMS (WfMC, 1995) (see fig. 7). The latter structures and describes precisely, the expected functionalities of a WfMS.

The reference model

The WfMC reference model is a centralised architectural model in which the main component is called *workflow enactment service*. The workflow enactment service is responsible for controlling the executions of workflow instances. It is composed of several *workflow engines*. A given workflow engine handles some parts of workflows and also manages some of their resources (Dumas et al., 2018; Van Der Aalst, 2013). According to the reference model, WfMS must provide tools to facilitate their configurations using workflow models: therein, these tools are referred to as *process definition tools*. Process definition tools are connected to the WfMS core (the workflow enactment service) via *Interface 1*. In order to execute tasks, users use *workflow client applications* that communicate with the WfMS via *Interface 2*. When necessary, a given workflow engine invokes other applications via *Interface 3*. The *administration and monitoring tools* connected via *Interface 5*, are used to monitor and control the workflows. Finally, the WfMS can be connected to other WfMS using *Interface 4*. A considerable effort has been made to standardise the five interfaces shown in figure 7. These efforts led to the production of languages (exchange formats) such as *Workflow Process Definition Language* (WPDL), *XML*⁶ *Process Definition Language* (XPDL) and BPEL.

5. BPEL is standardised by the OASIS consortium. OASIS website: <https://www.oasis-open.org/>. BPEL Specification (PDF version): <https://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf>.

6. XML: eXtensible Markup Language.

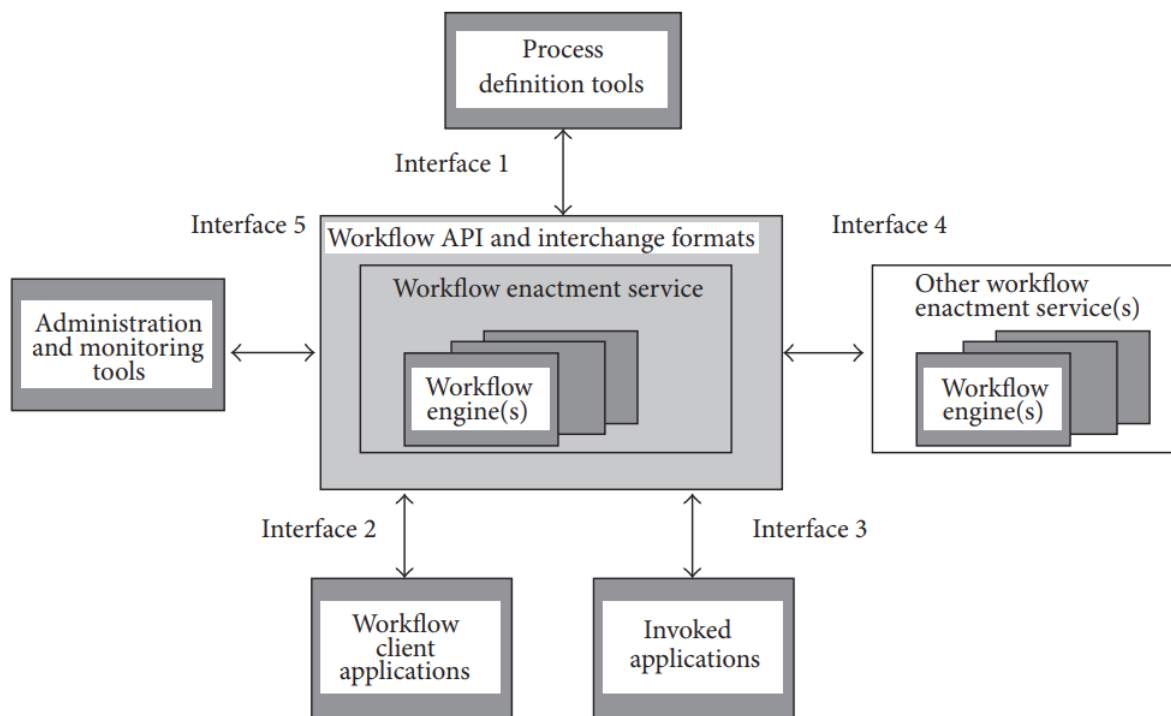


Figure 7 – Reference model of the Workflow Management Coalition (source (WfMC, 1995)).

The reference model has been very successful. Firstly, because to this day, it perfectly orchestrates the different tools used for the design and execution of workflows. Secondly, because it has served as the basic model for a very large number of WfMS in the industry. Examples include ActionWorkflow (Medina-Mora et al., 1993), FlowMark (Leymann and Roller, 1994), Staffware (Brown, 2005), In-Concert (Sarin, 1996), etc. Because the reference model is a centralised approach (client-server architecture), it has the advantage of facilitating a good mastery of the technologies used in the production of WfMS. Also, the implementation of (generally lightweight) client applications and the overall maintenance of WfMS (which is limited to the maintenance of the central server) are much simpler (Kanzow, 2004). However, systems based on a client-server architecture show some limitations because of the centralisation of workflow management. Their main weaknesses are: *the (non) fault tolerance*, *the (difficult) scalability* and the strong dependency of the system vis a vis the central server, which stores data, controls and thus, represents *a possible point of congestion* (Fakas and Karakostas, 2004; Yan et al., 2006). Concretely (Yan et al., 2006),

1. The client-server architecture allows centralised coordination of workflows with little use of the computing potential on the client side. Workflow systems based on such an architecture are very cumbersome. In application areas where several workflow instances need to be executed in parallel, the

centralised server can be overloaded with heavy computations and intensive communications when the system load increases, thus becoming a potential bottleneck.

2. Client-server systems are vulnerable to server failures. The centralised server is commonly viewed as a single point of congestion in the system. Its malfunction can cause the entire system to shut down.
3. The limited scalability of the client-server architecture prevents the WfMS based on it, from dealing with the ever-changing work environment. This also raises difficulties in system configuration, as any changes to the system, such as the admission of new actors, may require changes and updates to the centralised workflow server, which is very impractical and inefficient. Therefore, these WfMS are particularly unsuitable for application areas where workflow actors are required to join and leave frequently.
4. An important and crucial element of any workflow system is to allow actors to maintain their autonomy and control. However, workflow actors in a client-server-based WfMS are exclusively controlled by centralised servers. A serious problem is that, a large number of actors working on the "lightweight client side" may not be able to exercise their control, decision-making and problem-solving capabilities.

Knowing that various actors involved in a given business process are very often spread over remote sites, the reference model does not seem to be very suitable for efficiently implementing cooperation among them, as would systems based on a distributed architectural model be. In order to meet the shortcomings of the reference model, several works (Fakas and Karakostas, 2004; Imine, 2006; Kanzow, 2004; Lahcen and Parigot, 2012; Yan et al., 2006) have focused on the production of distributed WfMS built on top of peer-to-peer (P2P) architectures. This approach has also been successful since, systems such as ADEPT (Jennings, 1996) and METEOR₂ (Miller, Palaniswami, Sheth, Kochut, and Singh, 1998) have been designed over years (Kanzow, 2004).

I.3. Peer to Peer Business Process Management

Knowing that workflows are naturally distributed, they can sometimes involve resources from different organisations. Within each organisation, WfMS must

therefore be built with a strong emphasis on (sometimes inter-organisational) cooperation; this differs from the idea in which classical information systems have often been built. However, even if WfMS must be *interoperable* to facilitate cooperation, they must also ensure the *autonomy* and the *confidentiality* of actors and organisations involved in workflow execution; because, though organisations are aware of the need and necessity to participate in cooperation, they wish to protect their expertise in order to ensure sufficient confidentiality on their local data and local processes (Boukhedouma, 2015). The main challenge for WfMS designers over the last two decades, has therefore been to build WfMS capable of both ensuring the agility of organisations and fostering the interconnection of business processes, while preserving their autonomy and the confidentiality of their local processes and data.

The production of fully distributed WfMS proved to be an effective solution to this challenge (Meilin et al., 1998). This has been made more feasible with the advent of new concepts such as the *Multiagent* paradigm and the *Service-Oriented Architecture* (SOA). In this section, we take a look at how the distributed workflow management approach works, and some of the decentralised WfMS that have been developed for this purpose

1.3.1. The Advent of the Multiagent and Service-Oriented Concepts

1.3.1.1. The Multiagent Concept

The *agent* and *multiagents systems* concepts emerged in the 1980s. These concepts have generated lots of excitement in different research communities mainly because, they form the basis of a new paradigm for designing and implementing software systems that operates in distributed and open environments⁷, such as the internet (Sycara, 1998). One of the best-known and most famous definitions of the agent concept was formulated by Jacques Ferber (1997) and states that : *an agent is a physical or logical entity capable of acting upon itself and its environment, which has a partial representation of that environment, which, in a multiagent system, can communicate with other agents, and whose behaviour is the consequence of its observations, knowledge and interactions with other agents.*

7. "An open system is one in which the structure of the system itself is capable of dynamically changing. The characteristics of such a system are that its components are not known in advance; can change over time; and can consist of highly heterogeneous agents implemented by different people, at different times, with different software tools and techniques. Perhaps the best-known example of a highly open software environment is the internet. The internet can be viewed as a large, distributed information resource, with nodes on the network designed and implemented by different organisations and individuals." (Katia P. Sycara, (1998))

Multiagent systems have brought a new way to look at distributed systems and have provided a path to more robust intelligent applications (DeLoach, Wood, and Sparkman, 2001). The challenge of the multiagent concept is to build distributed systems in which the nodes (agents), endowed with great autonomy, high reactivity and communicating using an asynchronous messaging system (they therefore possess cooperation and deliberation/decision capabilities (Kanzow, 2004)), can appear and disappear at any time without paralysing the system. A multiagent system is characterised as follows (Sycara, 1998):

1. Each agent has incomplete information or capabilities for solving problems and thus, has a limited viewpoint;
2. There is no system global control;
3. Data are decentralised;
4. Computation is asynchronous.

Such properties for a multiagent system provide it with several capabilities that have mainly attracted researchers and professionals to the multiagent paradigm. Among these capabilities we can distinguish the following (Sycara, 1998):

- The capability to solve problems that are too large and difficult to handle by a centralised agent/server, because of resource limitations, or the sheer risk of having one centralised entity that could be a performance bottleneck or could fail at critical times;
- The capability to allow for the interconnection and interoperation of multiple existing legacy systems;
- The capability to provide solutions to problems that can naturally be regarded as a society of autonomous interacting components-agents;
- The capability to provide solutions that efficiently use information sources that are spatially distributed;
- The capability to provide solutions in situations where expertise is distributed.

The above mentioned capabilities of multiagent systems are in line with the desired capabilities of a distributed WfMS. This has motivated the growing use of concepts developed for multiagent systems, in the production of such WfMS. However, the vocabulary used by the designers of distributed WfMS is not always identical to the multiagent jargon, and it is sometimes necessary to abstract the

proposed systems to exhibit the multiagent concepts involved in their design and implementation.

1.3.1.2. The Service-Oriented Architecture

Basic concepts

Service-Oriented Architecture (SOA) has spread rapidly as a result of its growing success, and has been widely accepted as a supporting architecture for information systems because of its pivotal concept of *service* (Boukhedouma, 2015). Service is the essential concept of SOA and can be defined as "*self-describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications. Services perform functions, which can be anything from simple requests to complicated business processes. Services allow organizations to expose their core competencies programmatically over a network using standard languages and protocols, and be implemented via a self-describing interface based on open standards*" (Mike P. Papazoglou, (2003)). For MacKenzie et al. (2006), the term service combines the following related ideas :

- The offer to perform work for another;
- The capability to perform work for another;
- The specification of the offered work.

From a purely technological point of view, a service is a software component represented by two separate elements: its interface, which allows defining the access modalities to the service (name of the service and the parameters of the public operations defining the signatures of the operations) and its implementation (Boukhedouma, 2015). Services are offered by *service providers* (see fig. 8); these are organisations that procure the service implementations, supply their service interfaces and provide related technical and business support (Papazoglou, 2003). Service interfaces are available for their searching, their binding, and their invocation by *service consumers* (see fig. 8) (MacKenzie et al., 2006). Clients of services (service consumers) can be other solutions or applications within an enterprise or clients outside the enterprise. Service providers must therefore provide a distributed computing infrastructure for both intra and cross-enterprise application integration and collaboration. To satisfy these requirements, provided services should be (Papazoglou, 2003):

- *Technology neutral*: they must be able to be invoked by clients coded with various technologies having a few standards as a common denominator;

- *Loosely coupled*: they must not require neither knowledge nor internal structures or conventions (context) at the service consumer or service provider side;
- *Transparent from the point of view of their location*: one should be able to locate and invoke the services irrespective of their real location. To do so, the use of a *service registry* where services interfaces and location information are stored, is required (see fig. 8).

SOA is an architectural style, a logical way of designing a software system to provide services to either end-user applications or other services distributed in a network through published and discoverable interfaces. Basically, SOA defines an interaction between software agents as an exchange of messages between service consumers (clients) and service providers (see fig. 8). In SOA, the exchange of

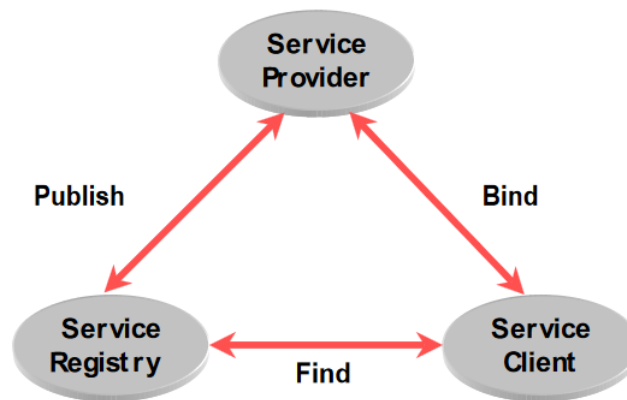


Figure 8 – The basic Service-Oriented Architecture (source (Papazoglou, 2003)).

messages between agents can be *synchronous* or *asynchronous*.

In the synchronous model, the service consumer invokes a service and expects a result. The invoked service is then designed to immediately return a result and is the only service involved. This model operates similarly to remote procedure call technologies such as Remote Method Invocation (RMI) but with a much loosely coupling between the service consumer and its provider.

In the asynchronous model (which is generally a particular form of *publish/subscribe*⁸), a given service consumer *A* expresses its desire to be aware of the execution state of a given service *b*, published (provided) by a service provider *B*, by subscribing to it. During this subscription, it provides information about one or

8. Publish/Subscribe is a communication paradigm well adapted to the loosely coupled nature of distributed interaction in large-scale applications; with systems based on its interaction scheme, subscribers register their interest in an event, or a pattern of events, and are subsequently asynchronously notified of events generated by publishers (Eugster, Felber, Guerraoui, and Kermarrec, 2003).

more services (a_i) that it also provides and that must be invoked when the execution state of b has changed. Several services are thus involved, and each agent is generally both a service provider and a service consumer.

SOA has been designed to facilitate the implementation of distributed applications based on Peer-to-Peer architectures (nodes/agents communicate directly without going through a central server) and in which, the skills of each agent are exposed, discoverable and invocable by the others but, the technique and technology used by each agent is confidential. This setting completes the concept of agents to answer correctly to the challenges of distributed WfMS: hence the very increasing use of the concept of service in workflow systems. Actually, some currents of thought claim that SOA was designed to facilitate the automation of business processes and thus, the design of distributed WfMS. This is the case of Hurwitz et al. (2009) who define SOA as : "*a software architecture for building applications that implement business processes or services by using a set of loosely coupled black-box components orchestrated to deliver a well-defined level of service*".

Shared-data Overlay Network

As the use of SOA in P2P applications escalates, there is a proliferation of tools to facilitate the design and implementation of these new applications (Kaur et al., 2013). *Shared-data Overlay Network* (SON) (Lahcen and Parigot, 2012) is one of those tools. SON is a middleware offering several *Domain Specific Languages* (DSL) to facilitate the implementation of P2P systems whose components communicate in an asynchronous manner by services invocations. SON combines the powerful concepts of *Component-Based Software Engineering*⁹, *Service-Oriented Computing* and *P2P Computing* in its engineering.

By using SON middleware, the P2P application designer (software developer) is able not only to specify applications in component-based service model, but also to perform an effective code generation. In fact (see fig. 9), the software developer defines using a dedicated DSL called *Component Description Meta Language* (CDML), for each component, a set of services (input, internal and output). Then, he only implements the code of the components, i.e., the methods that implement the defined services. Afterwards, a code generation tool, called *Component Generator*, generates a set of Java source files that implement the so-called *container of the component*. These Java files are compiled together with the implementation

9. Component-Based Software Engineering is an emerging paradigm of software development whose goal is, composing applications with plug & play software components on the frameworks; so, to realise software reuse by changing both software architecture and software process (Aoyama, 1998).

code to generate a standalone and ready-to-use component. Thus, software developers are assisted and have greater ease in developing component and service-based P2P applications. These facilities allow them to focus more on the business logic and to defer to SON, the management of the runtime requirements (e.g., communication mechanisms, instantiation and connection of components, service discovery, etc.). We use SON to implement prototypes of some of the models pre-

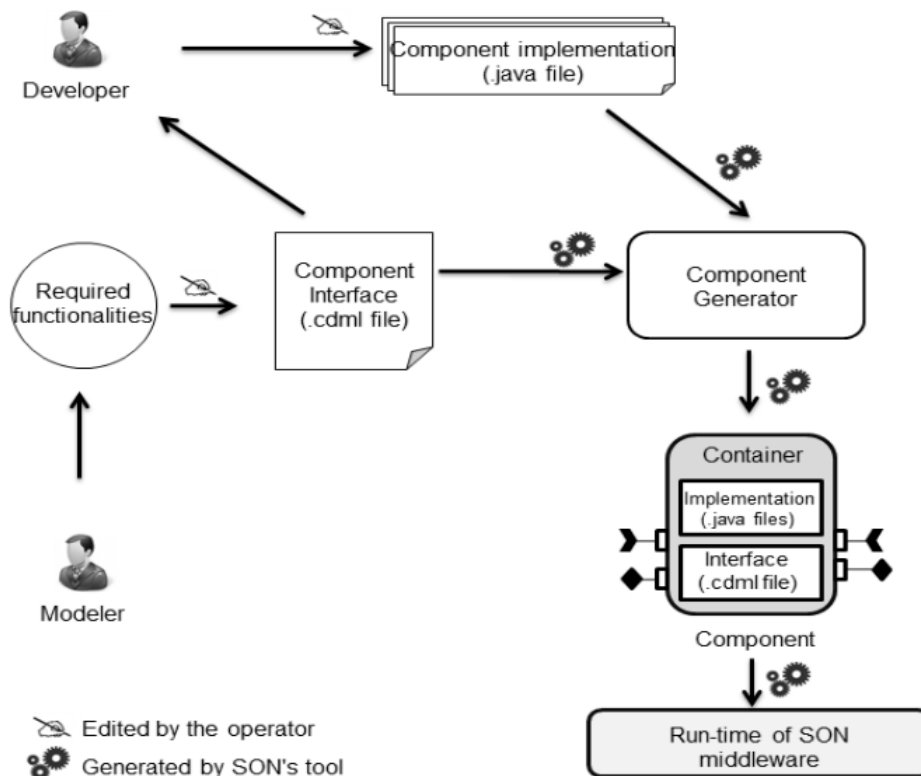


Figure 9 – Overview of a P2P application development process with SON (source (Lahcen and Parigot, 2012)).

sented in this manuscript.

I.3.2. Some Existing Distributed WfMS

In this section, we briefly present some existing approaches to distributed workflow management. As mentioned in (Kanzow, 2004), these approaches have the following characteristics :

- They are based on distributed entities that can communicate;
- These entities act autonomously, locally and thus, influence the further execution of the process (through their local actions, they choose the next actions to be executed);

- Each entity has a confidential local state;
- Each entity has only a partial view of the system's overall state at a given time;
- Workflow execution results from the automated interaction between the different entities.

The approaches to distributed workflow management presented here, can be divided into two categories :

1. The first category contains those in which, data and controls are partially distributed and the WfMS is based on a client-server architecture;
2. The second one is concerned with those in which WfMS, data and controls are fully distributed.

I.3.2.1. Some Partially Distributed WfMS

ADEPT (Advanced Decision Environment for Process Tasks) (Jennings, 1996)

The ADEPT project was designed to automate flexible workflows at British Telecom¹⁰. Its main goal is to allocate resources to business processes using agents. According to its logic, workflow tasks are executed by agents acting as cooperating actors in a system supervised by one or more statically or dynamically assigned servers.

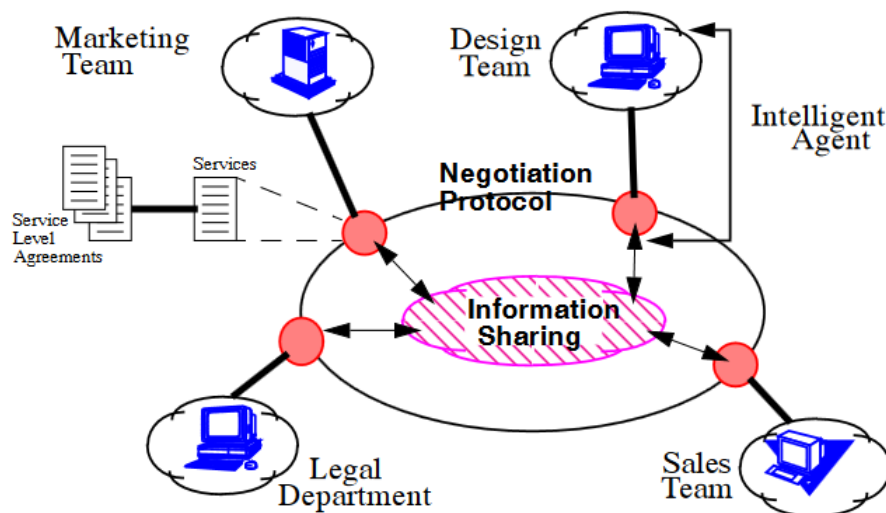


Figure 10 – An ADEPT environment (source (Jennings, 1996)).

10. Nowadays British Telecom is renamed BT Group and remains the leader in the fixed telephony sector (source Wikipedia - https://fr.wikipedia.org/wiki/BT_Group - visited the 07/03/2020).

Each agent is capable of providing one or more services. Services can be atomic (reduced to the execution of a single task) or composite (resulting from the combination of several other services, using operators that define the execution constraints - parallel, sequential, etc. -). If an agent needs the services of another agent, they must enter into an agreement called *Service Level Agreement*¹¹. To facilitate the negotiation of agreements between agents, ADEPT provides a negotiation protocol and a service description language. Technically, the service description language allows agents to expose their services so that, they can be discovered by other agents which can then initiate negotiations for the use of those services, through the negotiation protocol.

Figure 10 shows the architecture of ADEPT on an example of a workflow in which, four agents (marketing team, design team, sales team and legal department) collaborate to achieve business goals. Other publications on the ADEPT project may be useful for its understanding (Dadam, Reichert, and Kuhn, 2000; Reichert and Dadam, 1998; Reichert, Rinderle, and Dadam, 2003).

EvE (an Event-driven Distributed Workflow Execution Engine) (Geppert and Tombros, 1998)

According to the EvE approach, the distributed execution of workflows is done by event communication between agents (called *brokers*) in charge of executing tasks. These agents perform tasks and create events in response to the occurrence of other events. EvE is based on a multi-server architecture in which, each server manages an entire cluster (a local network). However, this multi-server architecture is made transparent for the different agents thanks to *adapters*; they can communicate independently of their respective domains. EvE provides amongst others the following services:

- Agents managed by servers and distributed across the network, capable of detecting events and executing tasks assigned to them; thereby, generating new events that are notified to other agents thanks to an inter-server communication mechanism that has been set up;
- A data warehouse in which information about agents, runtime data and *Event - Condition - Action*¹² (ECA) event handling rules are stored. The informa-

11. A Service Level Agreement is a formal contract used to guarantee that consumers' service quality expectation can be achieved (Wu and Buyya, 2012).

12. ECA is a paradigm that specifies the desired behaviour for reactive systems (i.e. systems that maintain ongoing interactions with their environments (Manna and Pnueli, 2012)). In such a system centered around the ECA paradigm, when an event occurs, a condition is evaluated (by a querying mechanism) and the system takes corresponding action (Almeida, Luntz, and Tilbury, 2005).

tion stored in the warehouse can be updated dynamically without the need to restart the system;

- Logging services for failure analysis and recovery. EvE supports exception notification, alerts and has the ability to resume execution after temporary disconnections;

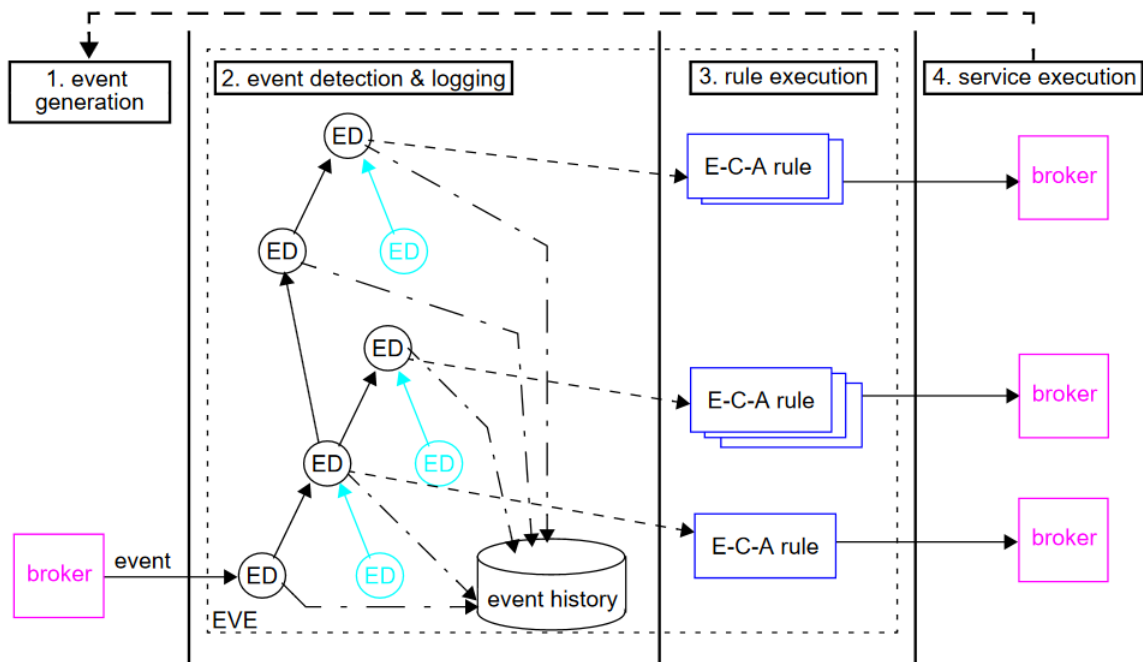


Figure 11 – The workflow execution process in EvE (source (Geppert and Tombros, 1998)).

The execution of a workflow starts as soon as an event is generated by a broker. The local EVE-server (its manager) then performs event detection and ECA-rule execution. Within the execution of each rule, task assignment determines responsible brokers, which are then notified and subsequently react as defined by their ECA-rules. Particularly, brokers can generate new events, which again are handled by EVE-servers, and so on transitively (see fig. 11).

I.3.2.2. Some Fully Distributed WfMS

METEOR₂ (Managing End-To-End Operations 2) (Das et al., 1997; Miller et al., 1998)

The METEOR₂ project is a continuation of the METEOR (Krishnakumar and Sheth, 1995) effort. It is intended to reliably support coordination of users and automated tasks in real-world multi-enterprise heterogeneous computing environments. Key capabilities of the METEOR₂ WfMS include a comprehensive toolkit for building workflows and supporting high-level process modelling, detailed work-

flow specification and automatic code generation for its workflow enactment systems.

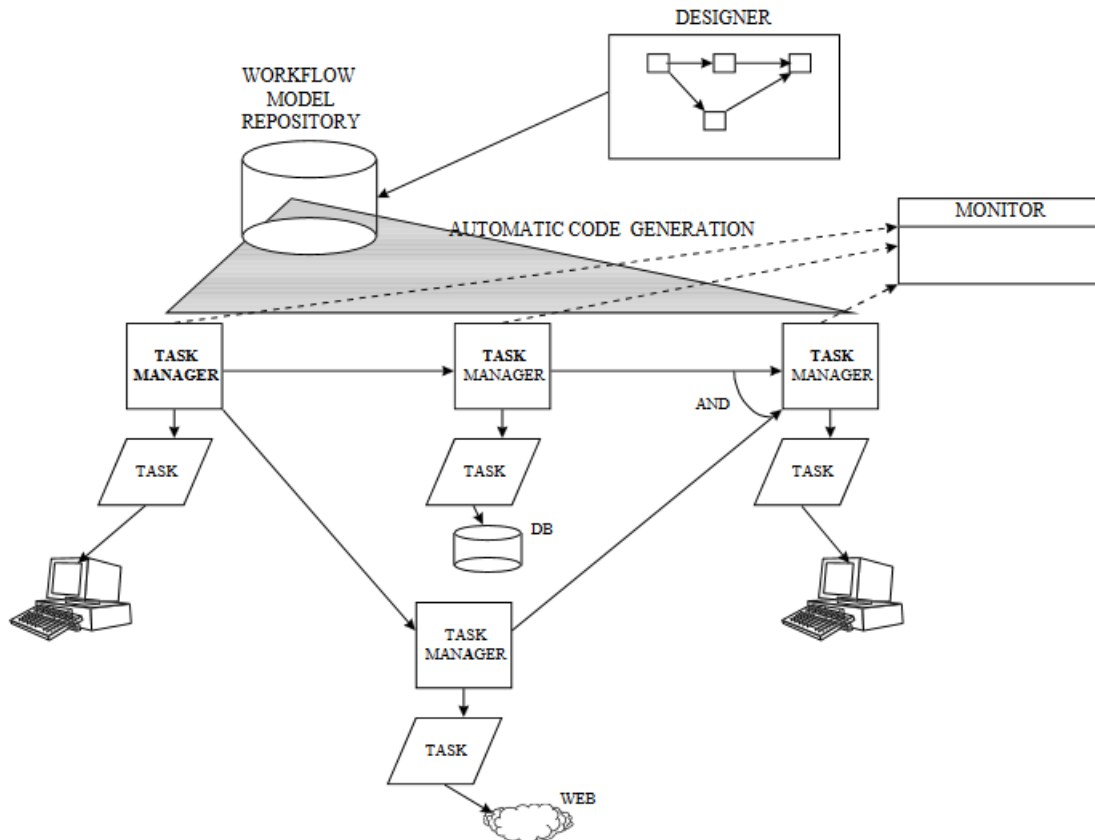


Figure 12 – The METEOR₂ architecture (source (Das et al., 1997)).

METEOR₂ introduces concepts to represent each workflow as a set of tasks, task managers, processing entities and interfaces, in order to execute them in a completely distributed manner. Figure 12 shows the various modules in METEOR₂ and their interaction. As can be seen in the picture, METEOR₂ includes a workflow designer that is used to create workflow models in a dedicated language. Once created, workflow models are stored in a workflow model repository. METEOR₂ also includes a workflow code generator that can read a stored workflow model and generate a convenient specific distributed workflow application. The generated application called the *runtime system*, consists of a set of communicating agents called *task managers* and their associated tasks, web-based user interfaces, a distributed recovery mechanism, a distributed scheduler and various monitoring components. All these workflow component are *Common Object Request Broker Architecture*¹³ (CORBA) objects and thus, they possess communication capabilities.

13. CORBA is a standard middleware for distributed object systems. In its paradigm, a client application wishing to perform an operation on a server object, sends a request. The request is received by an Object Request Broker (ORB), responsible for all of the mechanisms required to find the object implementation

The "Web Workflow Peer" Approach (Fakas and Karakostas, 2004)

The approach proposed by Fakas and Karakostas is based on the concepts of *Web Workflow Peer Directory* (WWPD) and *Web Workflow Peer* (WWP). WWPD is an active directory system that maintains a list of all peers (WWP) that are available to participate in web workflow processes. It allows peers to register with the system and offer their services and resources to other peers. During the execution of workflow processes, the WWPD assists WWP to locate other WWP and use their services and resources. In their setting, key functionalities and data are distributed among WWP. The architecture is completely decentralised as no central workflow engine is employed to coordinate the process execution. The WWP encapsulates the necessary knowledge to perform the activities that are assigned to it and also to delegate some of the process execution to other WWP. The only centralised feature is the WWPD.

A WWP is a processing unit with an interface that is exposed on the Web and which can be accessed using Internet protocols. Its interface describes different types of processing capabilities, each corresponding to a workflow activity. When combined, such activities form a workflow process. A WWP that initiates and administers the process is called the *Administrator Peer*. Other WWP delegated to carry out workflow activities are called the *Participating Peers* (see fig. 13). In practice, all peers are capable of becoming administrators in different workflow process instances. WWP use mobile documents called *Workflow Process Description* as communication medium. Segments of those documents move from site to site and conveys structural information about the running workflow instance.

Workflow process administration is achieved by employing a notification mechanism. For instance, at the completion of an activity the WWP notifies the Administrator Peer so that, an updated status of the process instance is maintained. Similarly, upon expiration of an activity deadline, the Administrator Peer notifies the WWP responsible for the expired activity. As far as we know, there is still no real workflow system based on this promising architecture.

SwinDeW (Swinburne Decentralised Workflow) (Yan et al., 2006)

Combining workflow and P2P concepts, SwinDeW (Yan et al., 2006) has been designed as a special P2P system, which provides workflow management support

for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request to the server object. A server object accessible by CORBA is referred to as a CORBA object (Houlding, 2004).

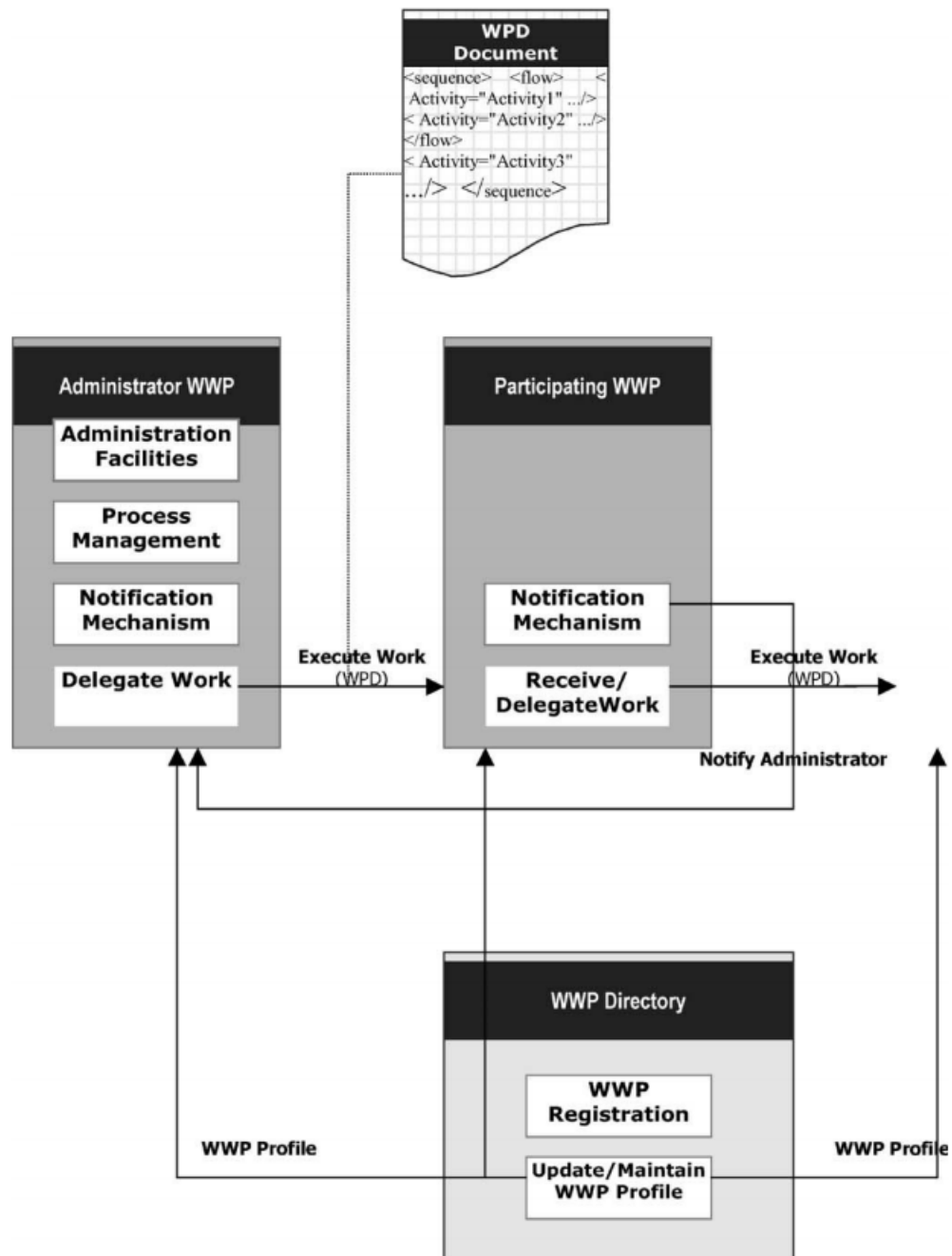


Figure 13 – A P2P workflow architecture (source (*Fakas and Karakostas, 2004*)).

in a truly decentralised way. SwinDeW adopts a flat, flexible and loosely coupled structure with an intentional absence of both a centralised device for data storage, and a centralised control engine for coordination. SwinDeW offers several distributed protocols, especially for the definition, instantiation and execution of processes.

The SwinDeW system is defined as four layers (see fig. 14). The top layer is the application layer; it defines application-related functions to fulfil workflows.

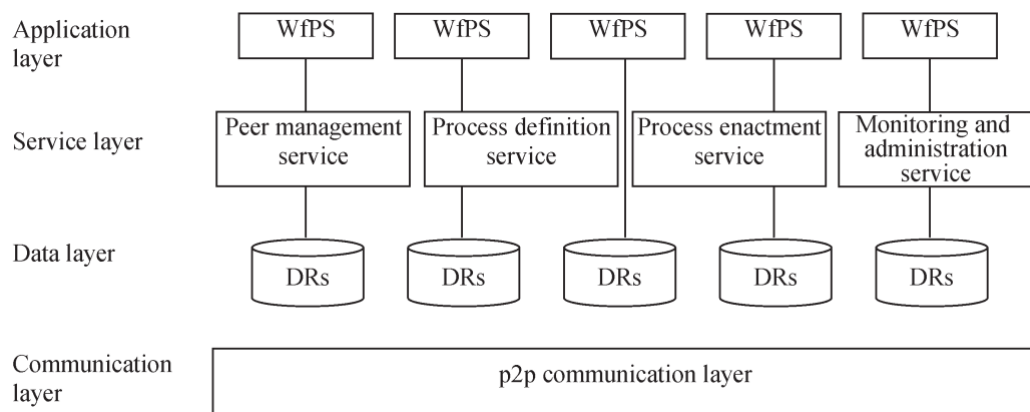


Figure 14 – A high-level view of SwinDeW's architecture (source (Yan et al., 2006)).

Workflow Participant Software (WfPS) is an application that provides interfaces to interact with a workflow participant and other WfPS, requesting services and responding to requests. Core services of the workflow system are provided at the service layer, which include the peer management service, the process definition service, the process enactment service, and the monitoring and administration service. The data layer consists of distributed Data Repositories (DR) that store workflow-related information. Finally, the monitoring and administration service provides supervisory capabilities and status monitoring.

In SwinDeW, a peer is given by a WfPS and a set of DR (see fig. 15). Each peer resides on a physical machine, enabling direct communication with other peers in order to carry out the workflow. A peer is a self-managing entity that is associated with and operates on behalf of a workflow participant. From the functional perspective, the WfPS of a peer consists of three software components :

1. A user component which serves as a "bridge" between the associated workflow participant and the workflow environment;
2. A task component that is in charge of the execution of tasks conducted by the associated participant;
3. A flow component which helps to fit an individual task into the workflow. It deals with data dependency and control dependency among tasks by handling incoming and outgoing messages.

A peer (agent) consists also in a set of four DR : the peer repository, the resource and tool repository, the task repository, and the process repository.

1. A given peer repository stores an organisational model that represents organisational entities and their relationships. This repository represents a user's view of the completely defined organisational model;
2. A resource and tool repository stores part of the resource model, which represents non human resources such as machines, external hardware, tools, etc.
3. A task repository stores a set of active task instances, which represent the work allocated to the associated workflow participant in the context of process instances;
4. A process repository stores a partial process definition distributed to the considered peer.

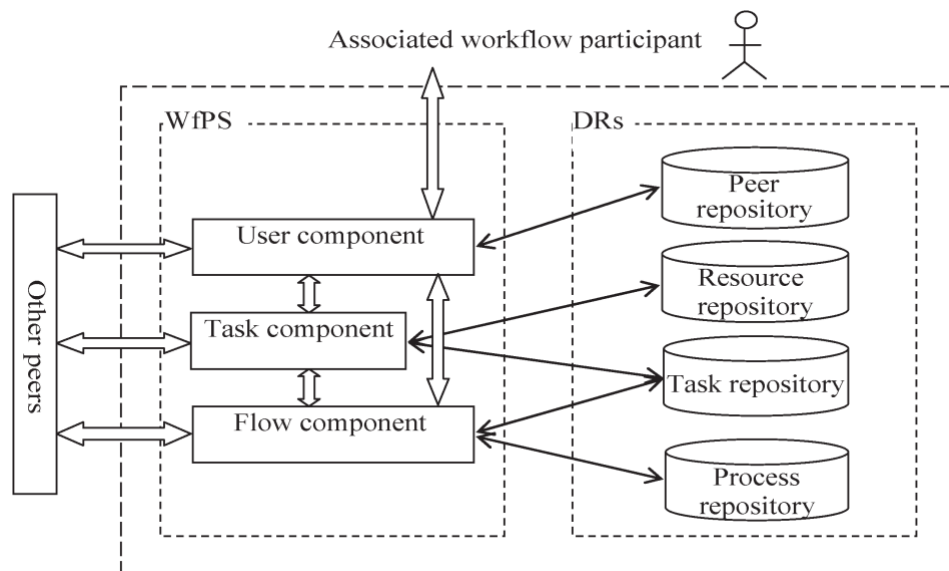


Figure 15 – Structure of a peer in SwinDeW (source (Yan et al., 2006)).

Workflow processes in SwinDeW are defined by a definition peer, which is associated with an authorised participant such as a process engineer. The resulting workflow models are stored in a distributed manner, in the process repositories of various peers. To avoid the distribution of too large workflow models, SwinDeW uses a "*know what you should know*" policy to partition these models and thus, to configure each peer only with the partitions of the models that are of interest to it.

In SwinDeW, a workflow instance is executed under the management of the workflow system. Once such an instance is created, a peer network is also constructed for carrying out this process instance. Various task instances are scheduled to enact at different sites, step by step. The execution of a task depends on the satisfaction of two conditions : the *information condition*, which defines the

start condition of a task from the data dependency perspective (a task can be executed only after essential input data are available), and the *control condition*, which indicates the start condition of a task from the control dependency perspective (a task can be executed only after some relevant work has been logically completed). Peers collaborate with one another through direct message exchange, to properly schedule the execution of various task instances. There are two kinds of messages flowing between peers: *information messages* and *control messages*, which are structured in XML format. When a peer receives messages from its predecessor peers directly, it evaluates the information and control conditions of the task instance independently, starts working when both the conditions are satisfied, and notifies its successor peers directly by delivering information messages and control messages after the task instance is completed. The successor peers repeat the same procedure until the completion of the whole process instance. This approach is then fully distributed; moreover, it has an implementation.

I.4. Artifact-Centric Business Process Management

Emerged in the early 2000s, the *artifact-centric* paradigm of BPM is one of those that has been much studied over the last two decades. This paradigm has been pioneered by IBM (Nigam and Caswell, 2003) and revisited in several works such as (Assaf, 2016; Assaf et al., 2017, 2018; Boaz et al., 2013; Deutsch et al., 2014; Estanol et al., 2012; Hull et al., 2009; Lohmann and Nyolt, 2011; Lohmann and Wolf, 2010); it proposes a new approach to BPM by focusing on both automated processes (tasks and their sequencing) and data manipulated through the concept of "*business artifact*" (*artifact-centric modelling*). In this section we present the key concepts of the artifact-centric paradigm as well as some artifact-centric frameworks from the literature.

I.4.1. Artifact-Centric BPM Basic Concepts

I.4.1.1. The Aim of Artifact-Centric BPM

In order to be able to better model workflows, process modelling should include a specification of the order in which tasks are executed (control flow), the way data are processed (data flow), and how different branches in distributed and inter-organisational business processes and services are invoked and coordinated (message flow) (Lohmann and Nyolt, 2011). These three conceptual models of workflows are also known as the *process*, the *informational* and the *organisational*

models (Divitini et al., 2001). Traditional approaches (BPMN, YAWL, BPEL, etc.) to BPM are process-centric (they are also said to be *imperative*): they generally offer two different views on business processes:

1. Collaboration diagrams (sometimes called interconnected models) that emphasise the local control flow of each participant of the process;
2. Choreography diagrams (interaction models) that describe the process from the point of view of the messages that are exchanged among the participants.

Traditional approaches thus express workflow models by means of diagrams which define how a workflow is supposed to operate, but give little importance (or none at all) to the information produced as a consequence of the process execution: data are treated as second-class citizens.

To precisely remedy this, researchers have developed the artifact-centric (Nigam and Caswell, 2003) approach to the design and execution of business processes. Artifact-centric models do not specify processes as a sequence of tasks to be executed or messages to be exchanged (i.e. imperatively), but from the point of view of the data objects (called *business artifacts* or simply *artifacts*) that are manipulated throughout the course of the process (i.e. declaratively) (Lohmann and Nyolt, 2011). They rely on the assumption that any business needs to record details of what it produces in terms of concrete information. Artifacts are proposed as a means to record this information. They model key business-relevant entities which are updated by a set of services (specified by pre and postconditions) that implement business process tasks. This approach has been successfully applied in practice and it provides a simple and robust structure for workflow modelling (Estanol et al., 2012).

1.4.1.2. How the Artifact-Centric Approach to BPM Works

According to the artifact-centric paradigm, BPM takes place in two main phases guided by the concept of artifact. In order to automate a given process, the designer must first of all focus on artifact modelling; i.e. he must provide data structures capable of storing and logically conveying the information produced during the execution of workflows. Then, these artifacts models will be introduced into a WfMS supporting artifact-centric execution of workflows for the enactment.

What is an artifact ?

(Nigam and Caswell, 2003) define an artifact as "*a concrete, identifiable, self-describing chunk of information that can be used by a business person to actually*

run a business". Artifacts are business-relevant objects that are created, evolve, and (typically) archived as they pass through the workflow (Hull et al., 2009); they represent key conceptual objects of workflow that evolve as they move through an enterprise. Artifacts are modelled through *artifacts types* (or *models*). An expected characteristic of artifacts is that they should be self-describing: this requirement allows a business person to be able to look at an artifact and determine if he or she can work on it. Toward this end, an artifact type should include both:

1. an *information model* (or "data schema"), for holding information about the artifact as it moves through the process, from creation to archival storage; and
2. a *lifecycle model* (or "lifecycle schema"), which describes how and when tasks (activities or services) might be invoked on the artifacts as they move through the process.

A prototypical example of an artifact type is the "air courier package delivery", whose information model can hold data about a package including sender, receiver, steps occurring in transport and billing activity; and whose lifecycle model would specify the possible ways that the delivery service and billing might be carried out (Hull et al., 2009; Hull, Su, and Vaculin, 2013).

Several approaches of modelling the lifecycle of artifacts have been studied in the literature. The most commonly used approach is that in which, some form of *finite state machines* (automata) (Hull et al., 2009) are used to specify lifecycles. Other variants presenting the lifecycle of an artifact by a Petri net (Lohmann and Wolf, 2010), logical formulae depicting legal successors of a state (Damaggio, Deutsch, and Vianu, 2012) have also been proposed.

The artifact-centric execution

Artifact-centric models can be executed by artifact-centric WfMS. This new type of WfMS put stress on how artifacts are created, updated and exchanged between various actors. In these, artifacts are considered as *adaptive documents* that conveys all the information concerning a particular execution case of a given process, from its inception in the system to its termination. In particular, this information provides details on the case's execution status as well as on its lifecycle (a representation of the possible evolutions of this status). To do this, during the execution of a given process, the actions carried out by each of the actors (agents) have the effect of updating (*editing*) the artifacts involved in that execution. If the process is cooperative, the artifact representing it will be updated by several

agents: it is said to be cooperatively edited and thus, the execution of a given business process according to the artifact-centric approach, can be assimilated to the *cooperative editing* of documents.

Two major trends in the artifact-centric modelling approach have been developed: *orchestration* and *choreography* (Hull et al., 2009).

1. Orchestration suggests the creation of centralised systems (usually called *artifact hubs*), coordinated by an *orchestrator* whose role is to facilitate interaction between actors while ensuring that business goals are met.
2. Choreography-oriented approaches get rid of the orchestrator, and model actors as autonomous agents coordinating with artifacts and communicating in a P2P manner, to accomplish business goals. In these, each agent focuses on achieving a local business goal and the achievement of the global business goal is the result of aggregating results from different local business goals.

Compared to choreography, orchestration reduces the agents' autonomy by making the orchestrator the main controller of interactions. Also, the orchestrated approach does not scale well. A limitation of the choreography-oriented approach is the lack of a single synchronisation point from which, it is possible to know the process's (actual) global execution state. Despite this, we agree with (Lohmann and Wolf, 2010) that, this completely decentralised approach is the one that best fits the modelling of the intrinsically distributed nature of business processes.

1.4.2. Some Existing Artifact-Centric BPM Frameworks

There are several frameworks in the literature, that implement artifact-centric concepts. We briefly present some of them in this section. We begin by presenting purely artifact-centric approaches; then we look at an even more flexible model, recently developed as a *data-centric* solution for case management.

1.4.2.1. Some Purely Artifact-Centric BPM Frameworks

Proclets (Van Der Aalst et al., 2001)

The concept of *proclets* was introduced to specify business processes in which, objects' lifecycles can be modelled at different levels of granularity and cardinality. A proclat can be seen as a lightweight workflow process equipped with a knowledge base that contains information on previous interactions; it is thus equipped with an explicit lifecycle or active documents (i.e., documents aware of tasks and processes): in this setting then, a proclat is both an agent and an artifact. Proclats

can find each other using a *naming service*, and communicate with each other to exchange messages through *channels* (see fig. 16).

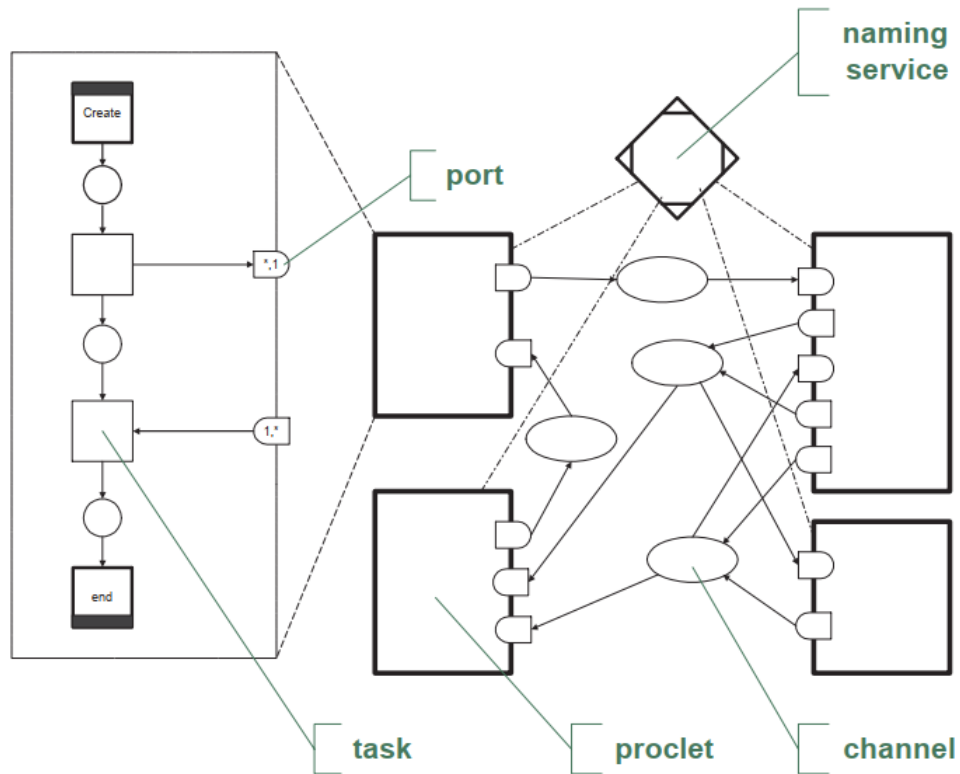


Figure 16 – Graphical representation of the procelet-based framework (source (Van Der Aalst et al., 2001)).

In the procelet-based framework, the lifecycle of procelet instance is described by a *procelet class* used as artifact type. Like an ordinary workflow model, a procelet class describes the order in which tasks can/need to be executed for individual instances of the class. Procelet classes are specified using a graphical language based on a sub-class of Petri nets so-called *class of sound WF-nets*.

Procleets are well-suited to deal with settings in which several instances of data objects are involved. Procleets are considered to be distributed and autonomous enough to decide how to interact with the other procleets: thus, the procelet-based framework does not model procleets' locations. Moreover, the execution model of procleets is similar to choreography; the interoperation of procleets is not managed or facilitated by a centralised hub.

Artifact hosting (Hull et al., 2009)

Hull et al. extend the artifact-centric model proposed by Nigam and Caswell (2003), to provide an interoperation framework in which data (artifacts) are hosted on central infrastructures named *artifact-centric hubs*. Data hosted in artifact-

centric hubs can be read and written by agents. This model is between choreography and orchestration because, agents are all connected to the hub but are not coordinated by a particular orchestrator. Unlike traditional orchestration schemes, the hub enables the participating agent to be pro-active, and serves primarily as a shared resource for coordinating activities. Participating agents can access information about the running artifact instances, can progress those instances along their lifecycles, and can subscribe to events in order to be alerted about significant steps in the progress of artifacts through their lifecycles. Security mechanisms are proposed for controlling access to data hosted in the hub.

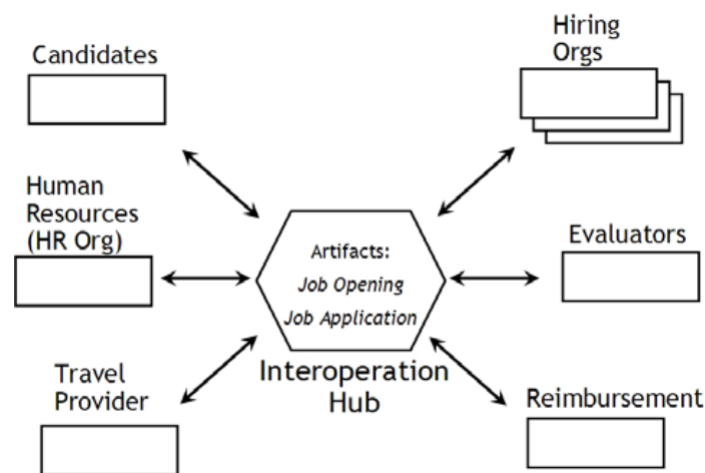


Figure 17 – An example of interoperation using an artifact-centric hub (source (Hull et al., 2009)).

Figure 17 illustrates an example of six groups of agents (potentially organisations) coordinating using artifacts that are managed in a centralised hub. These are the agents related to the resources (*Candidates*, the *Human Resources Organisation*, the *Hiring Organisations*, the *Evaluators*, the *Travel Provider* and the *Reimbursement*) that carry out the employee hiring process in a given enterprise.

Artifact-centric choreographies (Lohmann and Wolf, 2010)

Lohmann and Wolf (2010) provide a more choreography-like framework for artifact-centric interoperation. They abandon the fact of having a single artifact hub (Hull et al., 2009) and they introduce the idea of having several agents which operate on artifacts. Some of those artifacts are *mobile* (their location may change over time); thus, the authors provide a systematic approach for modelling artifact location and its impact on the accessibility of actions using a Petri net. Their model was designed with the conviction that by making explicit who is accessing an artifact and where the artifact is located, one will be able to automatically generate an interaction model that can serve as a contract between agents, and which make

sure that global goal states specified on artifacts are reached. They thus propose an approach to automatically derive such an interaction model.

Declarative choreographies (Sun, Xu, and Su, 2012)

In (Sun et al., 2012), the authors are also interested in choreographies. More precisely, they develop a language allowing to model (in a declarative manner) the collaboration between several actors (the choreography) and a distributed algorithm allowing the execution of the choreographies specified in their language. Their choreography language has four distinct features :

1. Each type of actor is an artifact schema with a selected sub-part of its information model visible to choreography specification.
2. Correlations between actor types and instances are explicitly specified, along with cardinality constraints on correlated instances (e.g. each Order instance may correlate with exactly one Payment instance and multiple Vendor instances).
3. Messages can include data; data in both messages and artifacts can be used in choreography constraints.
4. The language is declarative and uses logic rules based on a mix of first-order logic and a set of binary temporal operators from DecSerFlow¹⁴ (Van Der Aalst and Pesic, 2006).

In particular, Skolem¹⁵ notations are used to both reference correlated actor instances and to manipulate dependencies among messages.

1.4.2.2. A Guarded Attribute Grammars Based Framework to Data-Centric Case Management

What is case management ?

Highly important processes in organisations that have a tremendous impact on the success and add the most value, involve a high degree of knowledge work: they are driven by users' decisions (*user-centric*) making it difficult to specify them

14. DecSerFlow: *a Declarative Service Flow Language* is a graphical, extendible language for expressing process models in a declarative way; it captures what is the high-level process behaviour without expressing how it is procedurally executed, hence giving a concise and easily interpretable feedback to the business manager (Lamma, Riguzzi, Storari, Mello, and Montali, 2007).

15. Thoralf Albert Skolem (1887-1963) is a Norwegian mathematician and logician. He is particularly known for his work in mathematical logic and set theory which now bears his name, such as the Löwenheim-Skolem theorem or the notion of skolemisation (source, Wikipedia: https://en.wikipedia.org/wiki/Thoralf_Skolem, visited the 02/04/2020).

into a set of activities with precedence relations at design-time (they are said to be *knowledge-intensive*). Because knowledge-intensive processes are subject of frequent exceptions, traditional BPM solutions are not able to support them sufficiently (Hauder, Pigat, and Matthes, 2014). *Adaptive Case Management* (ACM) is gaining interest among researchers and practitioners as an emerging paradigm to master situations in which adaptations have to be made at run-time (unpredictable situations) by so called knowledge workers. In contrast to traditional BPM, the ACM paradigm is not dictating knowledge workers a predefined course of action, but provides them with the required information at the right time (they are *data-centric*) and authorises them to make decisions on their own (Hauder et al., 2014).

The notion of *case* in the ACM context, is closely related to the concept of artifact. Both involve the notion of a conceptual entity that progresses through time, according to some set of guidelines or lifecycle schema, and both taking advantage of a growing set of data accumulated over the case instance lifecycle (Hull et al., 2013). ACM can be seen as an extension of the artifact-centric paradigm in which, the flexibility of workflow models (types of artifacts) is highly valued; therefore, both users and data are treated as first-class citizens.

There is a growing research interest in the ACM paradigm and several models have already emerged. Guard-Stage-Milestone (GSM) (Damaggio et al., 2013; Hull et al., 2011), a declarative model of the lifecycle of artifacts, was recently introduced and has been adopted as a basis of *Case Management Model and Notation* (CMMN), the OMG standard for ACM. The GSM model defines Guards, Stages and Milestones to control the enabling, enactment and completion of (possibly hierarchical) activities; it then allows for dynamic creation of subtasks (the stages), and handles data attributes. However, interaction with users are modelled as incoming messages from the environment, or as events from low-level (atomic) stages. In this way, users do not explicitly contribute to the choice of a flow for a process.

Recently, Badouel et al. (2014; 2015) have proposed a more user-centric and data-driven ACM model (AWGAG) based on the concepts of Active Workspaces (AW) and Guarded Attribute Grammars (GAG). We are particularly interested in this model because, it incorporates concepts that we will manipulate throughout this manuscript. These include the concepts of *grammars* as artifact types, *structured documents* (trees) as artifacts, *artifact editing*, etc.

AWGAG (Badouel et al., 2015)

The AWGAG model of collaborative systems is centered on the notion of user’s workspace. It assume that the workspace of a user is given by a map. It is a tree used to visualise and organise tasks in which, the user is involved together with information used for their resolution. The workspace of a given user may, in fact, consist of several maps where each map is associated with a particular service offered by the user. In short, one can assume that a user offers a unique service so that any workspace can be identified with its graphical representation as a map.

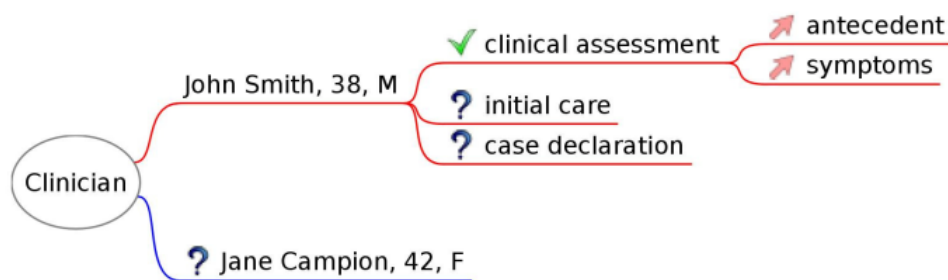


Figure 18 – Active workspace of a clinician (source (Badouel et al., 2015)).

As an example, figure 18 shows a map that represents the workspace of a clinician acting in the context of a disease surveillance system. The service provided by the clinician is identifying the symptoms of influenza in a patient, clinically examining the patient, eventually placing him under therapeutic care, declaring the suspect cases to the disease surveillance center, and monitoring the patient based on subsequent requests from the epidemiologist or the biologist.

Each call to this service, namely when a new patient comes to the clinician, creates a new tree rooted at the central node of the map. This tree is an artifact that represents a structured document for recording information about the patient all along being taken over in the system. Initially, the artifact is reduced to a single (open) node that bears information about the name, age and sex of the patient. An open node, graphically identified by a question mark, represents a pending task that requires the clinician’s attention. In this example the initial task of a given artifact is to clinically examine the patient. This task is refined into three subtasks: clinical assessment, initial care, and case declaration.

In the AWGAG model, a task is interpreted as a problem to be solved, that can be completed by refining it into sub-tasks using business rules. A business rule is modelled by a production $P : s_0 \rightarrow s_1 \dots s_n$ expressing that task s_0 can be reduced to subtasks s_1 to s_n . For instance, the production

$$patient \rightarrow clinicalAssessment \ initialCare \ caseDeclaration$$

states that, a task of sort *patient*, the axiom of the grammar associated with the service provided by the clinician, can be refined by three subtasks whose sorts are respectively *clinicalAssessment*, *initialCare*, and *caseDeclaration*. If several productions with the same left-hand side s_0 exist, then the choice of a particular production corresponds to a decision made by the user. In the example, the clinician has to decide whether the case under investigation has to be declared to the disease surveillance center or not. This decision can be reflected by the following two productions:

$$suspectCase : caseDeclaration \rightarrow followUp$$

$$benignCase : caseDeclaration \rightarrow$$

If the case is reported as suspect, then the clinician will have to follow up the case according to further requests of the biologist or the epidemiologist. On the contrary (i.e. the clinician has described the case as benign), the case is closed with no follow up actions.

AWGAG model considers artifacts as trees whose nodes are sorted and whose productions are taken into a grammar (GAG). The lifecycle of an artifact is implicitly given by the set of productions of the underlying GAG:

1. The artifact initially associated with a case, is reduced to a single open node.
2. An open node X of sort s can be refined by choosing a production $P : s \rightarrow s_1 \dots s_n$ that fits its sort. The open node X becomes a closed node under the decision of applying production P to it. In doing so, task s associated with X is replaced by n subtasks s_1 to s_n , and new open nodes X_1 of sort s_1 to X_n of sort s_n , are created accordingly: the artifact is said to be edited (see fig. 19).
3. The case reaches completion when its associated artifact is closed, i.e. it no longer contains open nodes.

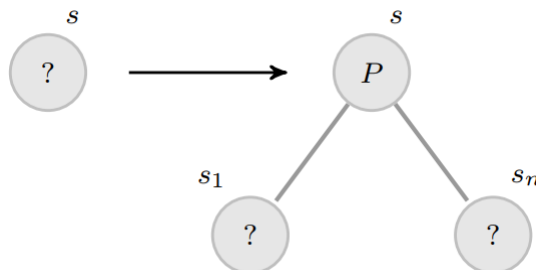


Figure 19 – Artifact edition in AWGAG (source (Badouel et al., 2015)).

Additional information are attached to open nodes using *attributes*, to model the interactions and data exchanged between the various tasks associated with them. For that, each sort comes equipped with a set of *inherited* attributes and a set of *synthesised* attributes where: inherited attributes represents input data, i.e. necessary data for the associated task to be executed, while synthesised attributes represents output data, i.e. data that are produced after the task being executed. This formalism puts emphasis on a declarative (logical) decomposition of tasks to avoid over-constrained schedules. Indeed, business rules do not prescribe any ordering on task executions. Ordering of tasks depend on the exchanged data and are therefore determined at runtime. In this way, the AWGAG model allows as much concurrency as possible in the execution of the current pending tasks.

Furthermore, a given AWGAG model is flexible and can incrementally be designed: one can initially let the designer manually develop large parts of the map, and progressively improve the automation of the process by refining the classification of the nodes, and introducing new business rules when recurrent patterns of activities are detected. The AWGAG model presents great properties such as *distribution* and *soundness*; these properties and more details are discussed in (Badouel et al., 2015). Several implementations and extensions of the AWGAG model are currently being carried out.

1.5. Summary

In this chapter, we have provided an overview of the basic concepts related to BPM. To this end, we have provided clear and concise definitions of numerous notions. We presented the concept of WfMS, how it works and the current challenges in the production of such systems. Knowing that the main current challenges in the production of WfMS are to provide them with the flexibility to better cope with the distributed nature of the workflows they manage, and extend their expressiveness so that they can address as first-class citizens, other perspectives of workflows such as data and users, we presented new paradigms to BPM and a few approaches to P2P BPM.

In addition, we focused on the artifact-centric paradigm for BPM and established that, according to it, the execution of a given workflow can be seen as the cooperative editing of one or more documents called artifacts. In these cases, it is preferable that the manipulated documents are structured; they can be exchanged between the different actors in the workflows' execution (they are said to be mo-

ble) to serve as a support to help their coordination, but also, indirectly, to be the fruit of their cooperation: this is the research axis that we followed in this thesis.

To make our task easier, it would be a good idea to look at structured documents' cooperative editing workflows: it is the subject of the next chapter. Since Badouel and Tchoupé have theorised an asynchronous cooperative editing model for structured documents of which, some of the concepts were brilliantly taken up in the development of the AWGAG model, it is their editing model that will be the main subject of our study.

II

CHAPTER

A WORKFLOW FOR STRUCTURED DOCUMENTS' COOPERATIVE EDITING : KEY PRINCIPLES AND ALGORITHMS

CONTENTS

| | |
|---|----|
| II.1 - Introduction | 56 |
| II.2 - Basic Concepts on Cooperative Editing Workflows | 58 |
| II.3 - Tree Automata for Extracting Consensus from Partial Replicas of a Structured Document | 62 |
| II.4 - A Software Architecture for Centralised Management of Structured Documents in a Cooperative Editing Workflow | 83 |
| II.5 - Summary | 89 |

II.1. Introduction

The purpose of this chapter is to introduce the main concepts related to asynchronous cooperative editing of structured documents. In an asynchronous cooperative editing workflow, several authors located on geographically distant sites coordinate to edit asynchronously the same structured document. In such editing workflows (see fig. 20), the desynchronised editing phases in which each co-author edits on his site, his copy of the document, alternate with the synchronisation-redistribution phases in which, the different contributions (local replicas) are merged (on a dedicated site) into a single document, which is then redistributed to the various co-authors for the continuation of the edition. This pattern is repeated until the document is completely edited.

Badouel and Tchoupé (2008) have theorised an asynchronous cooperative editing workflow in which, stakeholders (several subsystems - sites - distributed across a network) work by editing and exchanging (partial) replicas of documents representing their perceptions (views) at any given time. Therefore, each subsystem (actor) has a partial view of the edited document at any given time, and the current (global) document is given by the merging of different (partial) documents from the various subsystems. In their model, collaborations between actors can be divided into three sequential phases (see fig. 20):

- The *distribution phase* where global structured document (an artifact) is replicated to each subsystem;
- The *editing phase* in which local processes of subsystems are executed, inducing an update of the local replica of the global document;
- The *synchronisation phase* in which the various local documents updated are merged into a global document.

In this chapter, we are mainly interested in the model proposed by Badouel and Tchoupé. However, we are not just doing a systematic review of literature of their model. We subtly introduce three contributions which further validate their model, and polish the path towards our goal of producing a completely decentralised model for the automation of administrative workflows :

1. First of all, we extend the merge algorithm proposed by them so that, it can be applied in the more general case where conflicts might appear. To this end, we propose a consensus reconciliation algorithm that generates conflict-free maximum prefixes of the documents resulting from the merging of several conflicting replicates.
2. Second, we propose a generic system architecture that can be used to produce workflow systems for the cooperative editing of structured documents based on their model.
3. Finally, we propose a cooperative editing system prototype based on the proposed architecture and coded by cross-fertilisation of Java and Haskell.

In the rest of this chapter, in section II.2, we will present some basic concepts related to cooperative editing of documents, as well as some existing cooperative editing systems. In section II.3, we will present the main concepts of Badouel and Tchoupé's model as well as the reconciliation algorithm that we propose. In

section II.4, will be presented, the generic architecture of workflow systems that we propose, as well as a prototype system built according to it. We will conclude this chapter in section II.5.

II.2. Basic Concepts on Cooperative Editing Workflows

Cooperative editing is a work of (hierarchically) organised groups, that operate according to a schedule involving delays and a division of labor (coordination). Like any CSCW, cooperative editing is subject to spatial and temporal constraints. Thus, one distinguishes distributed or not, and synchronous or asynchronous cooperative editing. When distributed, the various editing sites are geographically dispersed and each of them has a local copy of the document to be edited; systems that support such an edition should offer algorithms for data replication (Saito and Shapiro, 2005) and for the fusion of updates. When asynchronous, various co-authors get involved at different times to bring their different contributions.

A cooperative editing workflow goes generally, from the creation of the document to edit, to the production of the final document through the alternation and repetition of distribution, editing and synchronisation phases. The literature is full of several cooperative editing workflows and of their management systems. We present a few in this section.

II.2.1. Real-Time Cooperative Editing Workflows

In these generally centralised systems (Etherpad¹ (Giannetti and Lord, 2015), Google Docs², Framapad³, Fidus Writer⁴ (Wilm and Frebel, 2014), etc.), the original document is created by a co-author on the central server. The latter then invites his colleagues to join him for the editing; they therefore connect to the editing session usually identified by a URL (distribution phase, although the document is generally not really duplicated). During an editing session (synchronous editing phase), all connected co-authors work on a single copy of the document but in different contexts. When the integration is automatic, changes performed by one of them are immediately (automatically) propagated to be incorporated into the basic document (synchronisation phase), and the latter is then redistributed to others.

1. Official website of Etherpad: <http://www.etherpad.org/>, visited the 04/04/2020.

2. Google Docs is accessible online at <https://www.docs.google.com/>, visited the 04/04/2020.

3. Get more information on Framapad at <http://www.framasoft.org/>, visited the 04/04/2020.

4. Official website of Fidus Writer: <https://www.fiduswriter.org/>, visited the 04/04/2020.

The changes are saved progressively and the server usually keeps multiple versions of the document.

The majority of real-time editors use the model of operational transformations (Oster, 2005; Tlili, 2011). Their architectures are therefore based on the one defined by this model. Meaning that, they distinguish two main components: an *integration algorithm*, responsible for the receipt, dissemination and execution of operations and a *set of processing functions* that are responsible for "merging" updates by serialising two concurrent operations.

II.2.2. Asynchronous Cooperative Editing Workflows

This edit mode is distinguished by real distribution phases in which, the document to be edited is replicated on different sites, using appropriate algorithms (Saito and Shapiro, 2005). A co-author may then contribute at any time, by editing his local copy of the document. Here, we focus on a few asynchronous cooperative editors operating in client-server mode.

Wikiwikiweb (Wikis)

Wikis (Cunningham, 1995) are a family of collaborative editors for editing web pages from a browser. To edit a page on a Wiki, one must duplicate it and contribute. After editing, he just have to save it and to publish a new version of that page. In a competing editing case, it is the last published version which will be visible. Even though it is still possible to access the previously published versions, there is no guarantee that a new version of the page preserves intentions (incorporates aspects) of previous versions. For this aspect, a Wiki can be seen much more as a web page version manager.

CVS (Concurrent Versions System)

Under CVS (Berliner et al., 1990), versions of a document are managed in a space called repository, and each user has a personal workspace. To edit a document, the user must create a replica in his workspace. He will amend this replica, then will release a new version of the document in the repository. In case the document is concurrently edited by several authors and at least one update has already been published, the author wishing to publish a new update, will be forced to consult and integrate all previous updates through dedicated tools, integrated in CVS.

SVN (Subversion)

SVN⁵ (Pilato, Collins-Sussman, and Fitzpatrick, 2008) was created to replace CVS. Its main goal was to propose a better implementation of CVS. So as CVS, SVN relies on an optimistic protocol of concurrent access management: the *copy-edit-merge* paradigm. SVN provides many technical changes like a new commit algorithm, the management of metadata versions, new user commands and many others features.

Git

The main purpose of Git⁶ is the management of various files in a content tree considered as a deposit (all files of a source code for example). To edit a deposit, a given user connects to it and clones (forks). He obtains a copy of that deposit, modifies it locally through a set of commands provided by Git. Then he offers his contribution to primary maintainer which can validate it and thus, merges it with the original deposit. During this operation, new versions of modified files are created in the main repository. It is therefore possible under Git, to access any revision of a given file.

II.2.3. Badouel and Tchoupé's Cooperative Editing Workflow

Badouel and Tchoupé (2008) proposed a workflow for cooperative editing of structured documents (those with regular structures defined by grammatical models such as DTD, XML schema (Bray et al., 2000), etc.), based on the concept of "view". The authors use context free grammars as documents models. A document is thus, a derivation tree for a given grammar.

The lifecycle of a document in their workflow can be sketched as follows: initially, the document to edit (t) is in a specific state (initial state); various co-authors who are potentially located in distant geographical sites, get a copy of t which they locally edit. For several reasons (confidentiality, security, efficiency, etc. (Tchoupé Tchendji, Djeumen D., and Atemkeng T., 2017)), a given co-author " i " does not necessarily have access to all the grammatical symbols that appear in the tree (document); only a subset of them can be considered relevant for him: that is his *view* (\mathcal{V}_i). The locally edited document, is therefore a *partial replica* (denoted $t_{\mathcal{V}_i}$) of the original document. This one is obtained by *projection* (π) of the original document with regard to the view of the considered co-author ($t_{\mathcal{V}_i} = \pi_{\mathcal{V}_i}(t)$). The

5. Check more about SVN at <http://www.subversion.apache.org/>, visited the 04/04/2020.

6. Official website of Git: <https://www.git-scm.com/>, visited the 04/04/2020.

edition is asynchronous and local documents obtained are called updated partial replicas denoted by $t_{\mathcal{V}_i}^{maj}$.

Badouel and Tchoupé focus only on the positive edition: edited documents are only increasing; thus, the co-authors cannot remove portions of the document when a synchronisation has already been performed. To both ensure that property, and be able to tell a co-author where he shall contribute, the documents being edited are represented by trees with *buds* that indicate the only places where editions are possible. Buds are typed; a *bud of sort X* is a leaf node labelled X_ω : it can only be edited (extended in a subtree) by using a *X-production* (production with X as left hand side).

When a synchronisation point is reached, all contributions $t_{\mathcal{V}_i}^{maj}$ of different co-authors are merged in a single global document t_f . To ensure that the merging is always possible (convergence), Badouel and Tchoupé assume that on each site, the editions are controlled by a local grammar. These local grammars are obtained from the global one, by projection along the corresponding views (Badouel and Tchoupé Tchendji, 2008; Tchoupé Tchendji et al., 2017).

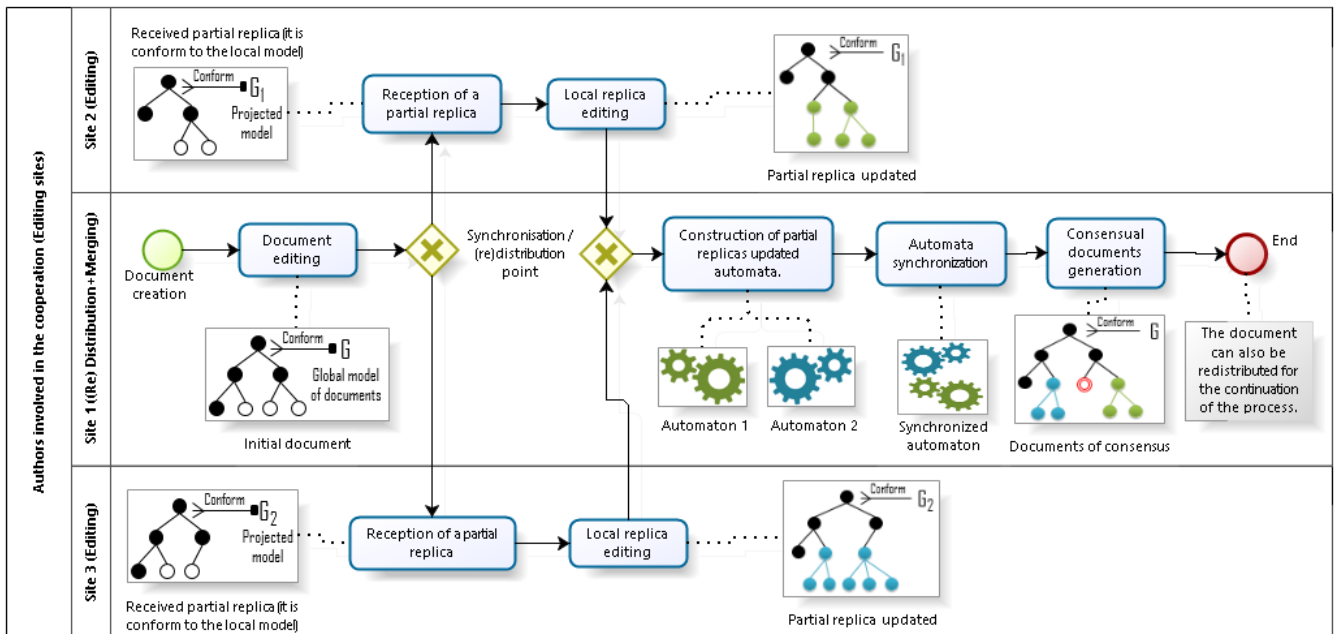


Figure 20 – A BPMN orchestration diagram sketching a cooperative editing workflow of a structured document according to Badouel and Tchoupé.

Figure 20 gives an overview, with a BPMN orchestration diagram, of the structured documents' cooperative editing workflow according to Badouel and Tchoupé's proposal; at site 1, operations of (re)distribution and merging of the document in accordance with a (global) model G , are realised; at sites 2 and 3, edition of partial

replicas in accordance with (local) models G_1 and G_2 , derived by projecting the global documents model G , are done.

In summary, the workflow of Badouel and Tchoupé is different from the others with its concept of "view" and by the fact that, it exclusively manipulates (partial) structured documents.

II.3. Tree Automata for Extracting Consensus from Partial Replicas of a Structured Document

In this section, we will better study Badouel and Tchoupé work on structured editing. We will adopt and adapt the different mathematical tools they proposed, to produce a more general algorithm for merging partial replicates, by taking into account the cases where these would be in conflict.

II.3.1. Structured Cooperative Editing and Notion of Partial Replication

II.3.1.1. Structured Document, Editing and Conformity

In the XML community, the document model is typically specified using a DTD or a XML Schema (Bray et al., 2000). It is shown that these DTD are equivalent to (regular) grammars with special characteristics called *XML grammars* (Berstel and Boasson, 2000). The (context free) grammars are therefore a generalisation of the DTD and, on the basis of the studies they have undergone, mainly as formal models for the specification of programming languages, they provide an ideal framework for the formal study of the transformations involved in XML technologies. That's why we use them in our work as tools for specifying the structure of documents.

We are only interested in the structure of the documents regardless of their contents and the attributes they may contain. We will therefore represent the abstract structure of a structured document by a tree, and its model by an abstract context free grammar; a valid structured document will then be a derivation tree for this grammar. A context free grammar defines the structure of its instances (the documents that are conform to it) by means of productions. A production, generally denoted $p : X_0 \rightarrow X_1 \dots X_n$, is comparable in this context, to a structuring rule which shows how the symbol X_0 , located in the left part of the production, is divided into a sequence of other symbols $X_1 \dots X_n$, located on its right side. More formally,

Definition 2 An *abstract context free grammar* is given by $\mathbb{G} = (S, \mathcal{P}, A)$ composed of a finite set S of **grammatical symbols** or **sorts** corresponding to the different **syntactic categories** involved, a particular grammatical symbol $A \in S$ called **axiom**, and a finite set $\mathcal{P} \subseteq S \times S^*$ of **productions**. A production $P = (X_{P(0)}, X_{P(1)} \cdots X_{P(n)})$ is denoted $P: X_{P(0)} \rightarrow X_{P(1)} \cdots X_{P(n)}$ and $|P|$ denotes the length of the right hand side of P . A production with the symbol X as left part is called a X -production.

For certain treatments on trees (documents), it is necessary to designate precisely a particular node. Several indexing techniques exist, among them, the so-called *Dynamic Level Numbering* (Böhme and Rahm, 2004) based on identifiers with variable lengths, inspired by the *Dewey decimal classification* (see fig. 21). According to this indexing system, a tree can be defined as follows:

Definition 3 A *tree* whose nodes are labelled in an alphabet S , is a partial map $t: \mathbb{N}^* \rightarrow S$, whose domain $Dom(t) \subseteq \mathbb{N}^*$ is a prefix closed set such that, for all $u \in Dom(t)$, the set $\{i \in \mathbb{N} \mid u \cdot i \in Dom(t)\}$ is a non-empty interval of integers $[1, \dots, n] \cap \mathbb{N}$ ($\epsilon \in Dom(t)$ is the root label); the integer n is the **arity** of the node whose address is u . $t(u)$ is the value (label) of the node in t , whose address is u . If t_1, \dots, t_n are trees and $a \in S$, we denote $t = a[t_1, \dots, t_n]$, the tree t of domain $Dom(t) = \{\epsilon\} \cup \{i \cdot u \mid 1 \leq i \leq n, u \in Dom(t_i)\}$ with $t(\epsilon) = a$, and $t(i \cdot u) = t_i(u)$.

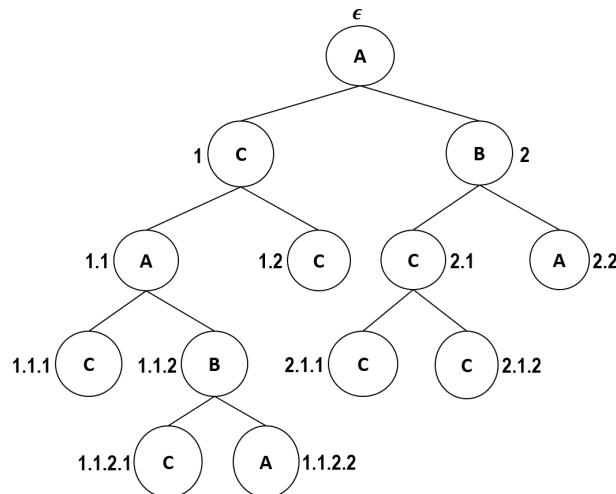


Figure 21 – Example of an indexed tree.

Let t be a document and $\mathbb{G} = (S, \mathcal{P}, A)$ a grammar. t is a derivation tree for \mathbb{G} if its root node is labelled by the axiom A of \mathbb{G} , and if for all internal node n_0 labelled by the sort X_0 , and whose sons n_1, \dots, n_n , are respectively labelled by the

sorts X_1, \dots, X_n , there is one production $P \in \mathcal{P}$ such that, $P : X_0 \rightarrow X_1 \cdots X_n$ and $|P| = n$. It is also said in this case, that t belongs to the language generated by \mathbb{G} from the symbol A , and it is denoted $t \in \mathcal{L}(\mathbb{G}, A)$ or $t : : \mathbb{G}$.

There is a bijective correspondence between the set of derivation trees of one grammar and all its *Abstract Syntax Tree (AST)*. In an AST, nodes are labelled by the names of the productions.

Definition 4 The set $AST(\mathbb{G}, X)$ of **abstract syntax trees** according to the grammar \mathbb{G} associated with grammatical symbol X , consists of trees in the form $P[t_1, \dots, t_n]$ where P is a production such that, $X = X_{P(0)}$, $n = |P|$ and $t_i \in AST(\mathbb{G}, X_i)$, $X_i = X_{P(i)}$ for all $1 \leq i \leq n$.

AST are used to show that a given tree, labelled with grammatical symbols, is an instance of a given grammar.

A structured document being edited, is represented by a tree containing *buds* (or *open nodes*) which indicate in a tree, the only places where editions (i.e updates) are possible⁷. Buds are typed; a *bud of sort X* is a leaf node labelled by X_ω (see fig. 22): it can only be edited (i.e extended to a subtree) by using an *X-production*. Thus, a structured document being edited and that have the grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, A)$ as model, is a derivation tree for the extended grammar $\mathbb{G}_\Omega = (\mathcal{S} \cup \mathcal{S}_\omega, \mathcal{P} \cup \mathcal{S}_\Omega, A)$, obtained from \mathbb{G} as follows: for all sort X , we not only add in the set \mathcal{S} of sorts a new sort X_ω , but we also add a new ε -production $X_\Omega : X_\omega \rightarrow \varepsilon$ in the set \mathcal{P} of productions; so we have: $\mathcal{S}_\omega = \{X_\omega, X \in \mathcal{S}\}$ and $\mathcal{S}_\Omega = \{X_\Omega : X_\omega \rightarrow \varepsilon, X_\omega \in \mathcal{S}_\omega\}$.

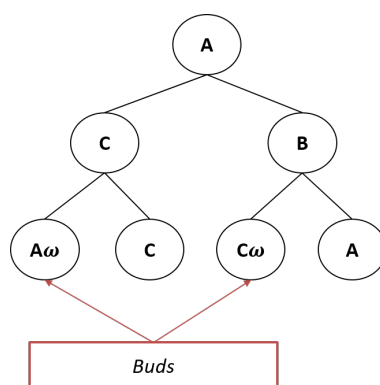


Figure 22 – Example of a tree that contains buds.

7. Note that, we are interested only in the *positive edition* based on a partial optimistic replication (Saito and Shapiro, 2005) of edited documents. In fact, the edited documents are only increasing: there is no possible erasure as soon as a synchronisation has been performed.

When we look at the productions of a grammar, we can notice that each sort is associated with a set of productions. From this point of view therefore, we can consider a grammar as an application

$$gram : symb \rightarrow [(prod, [symb])]$$

which associates to each sort, a list of pairs formed by a production name and the list of sorts in the right hand side of this production. Such an observation suggests that a grammar can be interpreted as a (descending) tree automaton that can be used for recognition or for the generation of its instances.

Definition 5 A (descending) **tree automaton** defined on Σ , is a quadruplet $\mathcal{A} = (\Sigma, Q, R, q_0)$ of a set Σ of symbols ; its elements are the nodes' labels of the trees to be generated (or recognised), a set Q of states, a particular state $q_0 \in Q$ called initial state, and a finite set $R \subseteq Q \times \Sigma \times Q^*$ of transitions.

- An element of R is denoted $q \rightarrow (\sigma, [q_1, \dots, q_n])$ or in an equivalent way $q \xrightarrow{\sigma} (q_1, \dots, q_n)$: intuitively, $[q_1, \dots, q_n]$ is the list of states accessible from q by crossing a transition labelled σ .
- If $q \xrightarrow{\sigma_1} (q_1^1, \dots, q_{n_1}^1), \dots, q \xrightarrow{\sigma_k} (q_1^k, \dots, q_{n_k}^k)$ denotes the set of transitions associated to the state q , we denote next $q = [(\sigma_1, [q_1^1, \dots, q_{n_1}^1]), \dots, (\sigma_k, [q_1^k, \dots, q_{n_k}^k])]$, the list that consists of pairs $(\sigma_i, [q_1^i, \dots, q_{n_i}^i])$. A transition of the form $q \rightarrow (\sigma, [])$, is called final transition and a state possessing this transition is a final state.

One can interpret a grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, A)$ as a (descending) tree automaton (Comon et al., 1997) $\mathcal{A} = (\Sigma, Q, R, q_0)$ considering that:

- (1) $\Sigma = \mathcal{P}$ is the type of labels of the nodes forming the tree to recognise;
- (2) $Q = \mathcal{S}$ is the type of states and,
- (3) $q \rightarrow (\sigma, [q_1, \dots, q_n])$ is a transition of the automaton when the pair $(\sigma, [q_1, \dots, q_n])$ appears in the list $(gram \ q)$ ⁸.

We note $\mathcal{A}_{\mathbb{G}}$ the tree automaton derived from \mathbb{G} .

To obtain the set $AST_{\mathcal{A}}$ of AST generated by a tree automaton \mathcal{A} from an initial state q_0 , one must:

⁸ Reminder: $gram$ is the application obtained by abstraction of \mathbb{G} and have as type : $gram : symb \rightarrow [(prod, [symb])]$.

- (1) Create a root node r , associate the initial state q_0 and add it to the set $AST_{\mathcal{A}}$ initially empty;
- (2) Remove from $AST_{\mathcal{A}}$ an AST t under construction, i.e. with at least one leaf node $node$ unlabelled. Let q be the state associated to $node$. For all transition $q \xrightarrow{\sigma} (q_1, \dots, q_n)$ of \mathcal{A} , add in $AST_{\mathcal{A}}$ the trees t' which are replicas of t in which, the node $node$ has been substituted by a node $node'$ labelled σ and possessing n (unlabelled) sons, each associated to a (distinct) state q_i of $[q_1, \dots, q_n]$;
- (3) Iterate step (2) until he obtains trees with all the leaf nodes labelled (they are consequently associated to the final states of \mathcal{A}): these are AST .

We note $\mathcal{A} \models t \triangleright q$ the fact that the tree automaton \mathcal{A} accepts the tree t from the initial state q , and $\mathcal{L}(\mathcal{A}, q)$ (tree language) the set of trees generated by the automaton \mathcal{A} from the initial state q . Thus, $(\mathcal{A} \models t \triangleright q) \Leftrightarrow (t \in \mathcal{L}(\mathcal{A}, q))$.

As for automata on words, one can define a synchronous product on tree automata to obtain the automaton recognising the intersection, the union, etc., of regular tree languages (Comon et al., 1997). We introduce below the definition of the synchronous product of k tree automata whose adaptation will be used in the next section for the derivation of the consensual automaton.

Definition 6 Synchronous product of k automata:

Let $\mathcal{A}_1 = (\Sigma, Q^{(1)}, R^{(1)}, q_0^{(1)})$, ..., $\mathcal{A}_k = (\Sigma, Q^{(k)}, R^{(k)}, q_0^{(k)})$ be k tree automata. The synchronous product of these k automata $\mathcal{A}_1 \otimes \dots \otimes \mathcal{A}_k$ denoted $\otimes_{i=1}^k \mathcal{A}^{(i)}$, is the automaton $\mathcal{A}_{(sc)} = (\Sigma, Q, R, q_0)$ defined as follows:

- (a) Its states are vectors of states : $Q = Q^{(1)} \times \dots \times Q^{(k)}$;
- (b) Its initial state is the vector formed by the initial states of the different automata : $q_0 = (q_0^{(1)}, \dots, q_0^{(k)})$;
- (c) Its transitions are given by :

$$\begin{aligned} (q^{(1)}, \dots, q^{(k)}) &\xrightarrow{a} \left((q_1^{(1)}, \dots, q_1^{(k)}), \dots, (q_n^{(1)}, \dots, q_n^{(k)}) \right) \Leftrightarrow \\ &\left(q^{(i)} \xrightarrow{a} (q_1^{(i)}, \dots, q_n^{(i)}) \quad \forall i, 1 \leq i \leq k \right) \end{aligned}$$

II.3.1.2. Notions of View, Projection, Reverse Projection and Merging

View, associated projection and merging

The derivation tree giving the (global) representation of a structured document edited in a cooperative way, makes visible the set of grammatical symbols of the grammar that participated in its construction. As we mentioned in section II.2.3 above, for reasons of confidentiality (accreditation degree), a co-author manipulating such a document will not necessarily have access to all of these grammatical symbols; only a subset of them can be considered relevant for him: it is his *view*. A view \mathcal{V} is then a subset of grammatical symbols ($\mathcal{V} \subseteq \mathcal{S}$).

A partial replica of t according to the view \mathcal{V} , is a partial copy of t obtained by deleting in t , all the nodes labelled by symbols that are not in \mathcal{V} . Figure 23 shows a document t (center) and two partial replicas t_{v_1} (left) and t_{v_2} (right) obtained respectively by projections from the views $\mathcal{V}_1 = \{A, B\}$ and $\mathcal{V}_2 = \{A, C\}$.

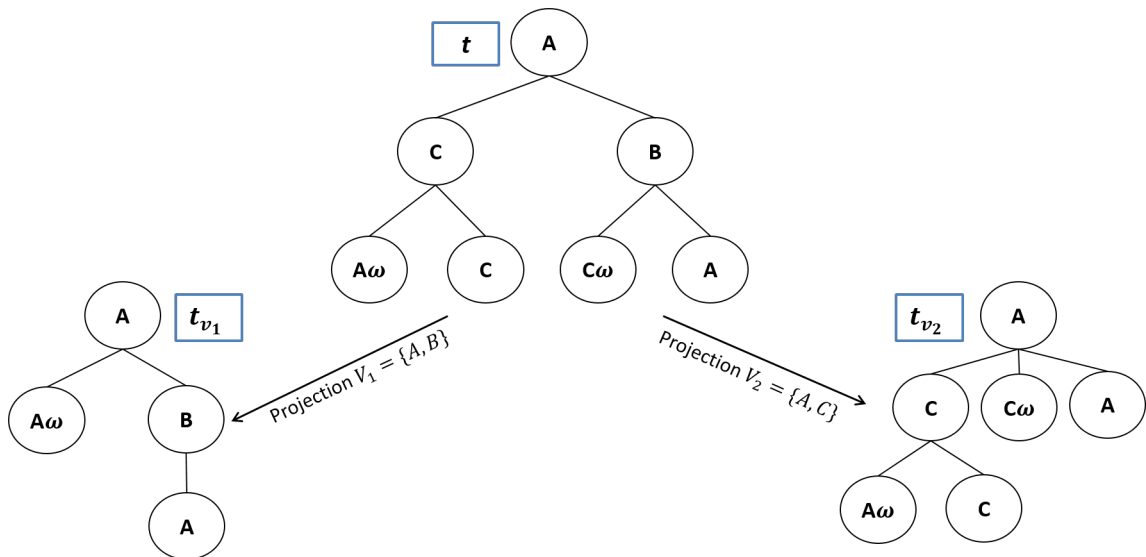


Figure 23 – Example of projections made on a document and partial replicas obtained.

Practically, a partial replica is obtained via a *projection* operation denoted π . We therefore denote $\pi_{\mathcal{V}}(t) = t_{\mathcal{V}}$ the fact that $t_{\mathcal{V}}$ is a partial replica obtained by projection of t according to the view \mathcal{V} .

Note $t_{\mathcal{V}_i} \leq t_{\mathcal{V}_i}^{maj}$ the fact that, the document $t_{\mathcal{V}_i}^{maj}$ is an update of the document $t_{\mathcal{V}_i}$, i.e. $t_{\mathcal{V}_i}^{maj}$ is obtained from $t_{\mathcal{V}_i}$ by replacing some of its buds by trees. In an asynchronous cooperative editing process, there are synchronisation points⁹ in which, one tries to merge all contributions $t_{\mathcal{V}_i}^{maj}$ of the various co-authors to obtain a sin-

9. Recall that a synchronisation point can be defined statically or triggered by a co-author as soon as certain properties are satisfied.

gle comprehensive document t_f ¹⁰. A merging algorithm that does not incorporate conflict management and relies on a solution to the *reverse projection* problem was given by Badouel and Tchoupé.

Partial replica and reverse projection (expansion)

The *reverse projection* (also called the *expansion*) of an updated partial replica $t_{\mathcal{V}_i}^{maj}$ relatively to a given grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, is the set $T_{i\mathcal{S}}^{maj}$ of documents conform to \mathbb{G} , that admit $t_{\mathcal{V}_i}^{maj}$ as partial replica according to \mathcal{V}_i :

$$T_{i\mathcal{S}}^{maj} = \left\{ t_{i\mathcal{S}}^{maj} \cdot \mathbb{G} \mid \pi_{\mathcal{V}_i} \left(t_{i\mathcal{S}}^{maj} \right) = t_{\mathcal{V}_i}^{maj} \right\}$$

A solution to the problem of evaluating the expansion of a given partial replica using tree automata, was proposed by Badouel and Tchoupé; in that solution, productions of the grammar \mathbb{G} are used, to bind to a view $\mathcal{V}_i \subseteq \mathcal{S}$ a tree automaton $\mathcal{A}^{(i)}$ such as, the trees it recognises from an initial state built from $t_{\mathcal{V}_i}^{maj}$, are exactly those having this partial replica as projection according to the view \mathcal{V}_i : $T_{i\mathcal{S}}^{maj} = \mathcal{L} \left(\mathcal{A}^{(i)}, q_{t_{\mathcal{V}_i}^{maj}} \right)$. Practically, they have considered that a state q of the automaton $\mathcal{A}^{(i)}$ is a pair $(Tag X, ts)$ where X is a grammatical symbol, ts is a forest (tree set), and Tag is a label that is either *Open* or *Close*, and indicates whether the concerned state q can be used to generate a *closed* node or a *bud*. The states of $\mathcal{A}^{(i)}$ are typed: a state of the form $(Tag X, ts)$ is of type X . We also have a function named *typeState* which, when applied to a state, returns its type¹¹. A transition from one state q , is of one of the forms $(Close X, ts) \rightarrow (p, [q_1, \dots, q_n])$ or $(Open X, []) \rightarrow (X_\omega, [])$. A transition of the form $(Close X, ts) \rightarrow (p, [q_1, \dots, q_n])$ is used to generate AST of type X (whose root is labelled by a X -production) admitting " ts " as projection according to the view \mathcal{V}_i if X does not belong to \mathcal{V}_i , and " $x[ts]$ " otherwise. Similarly, a transition of the form $(Open X, []) \rightarrow (X_\omega, [])$ is used to generate a single AST reduced to a bud of type X .

The interested reader may consult (Badouel and Tchoupé Tchendji, 2008) for a more detailed description of the process of associating a tree automaton with a view and the section II.3.2.3 for an illustration.

10. It may happen that the edition must be continued after the merging (this is the case if there are still buds in the merged document): one must redistribute to each of the n co-authors a (partial) replica $t_{\mathcal{V}_i}$ of t_f such that $t_{\mathcal{V}_i} = \pi_{\mathcal{V}_i}(t_f)$ for the continuation of the editing process.

11. *typeState* :: *state* \rightarrow *symb*
 . *typeState* (*Open* X , ts) = X
 . *typeState* (*Close* X , ts) = X

II.3.2. Reconciliation by Consensus

II.3.2.1. Issue and Principle of the Solution of Reconciliation by Consensus

There are generally two distinct phases when synchronising replicas of a document (Balasubramaniam and Pierce, 1998): the *updates detection* phase, which consists of recognising the different replica nodes (locations) where updates have been made since the last synchronisation, and the *propagation* phase, which consists in combining the updates made on the various replicas to produce a new synchronised state (document) for each replica. In an asynchronous cooperative editing workflow of several partial replicas of a document, when you reach a synchronisation point, you can end up with unmergeable replicas in their entirety as they contain not compatible updates¹²: they must be reconciled. This can be done by questioning (cancelling) some local editing actions in order to resolve conflicts and result in a coherent global version said of consensus.

Studies on reconciling a document versions are based on heuristics (Mens, 2002) as there is no general solution to this problem. In our case, since all editing actions are reversible¹³ and it is easy to locate conflicts when trying to merge the partial replicas (see section II.3.2.2), we have a canonical method to eliminate conflicts: when merging, we replace any node of the global document whose replicas are in conflict, by a bud. Thus, we prune at the nodes where a conflict appears, replacing the corresponding subtree with a bud of the appropriate type, indicating that this part of the document is not yet edited: the documents obtained are called consensus. These are the maximum prefixes without conflicts of the fusion of the documents resulting from the different expansions of the various updated partial replicas. For example, in figure 26, the parts highlighted (blue backgrounds) in trees (f) and (g) are in conflict; they are replaced in the consensus tree (h) by a bud of type C (node labelled C_ω).

The problem of the consensual merging of k updated partial replica whose global model is given by a grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ can therefore be stated as follows :

Problem of the consensual merging: given k views $(\mathcal{V}_i)_{1 \leq i \leq k}$ and k partial replicas $(t_{q_i}^{maj})_{1 \leq i \leq k}$, merge consensually the family $(t_{q_i}^{maj})_{1 \leq i \leq k}$ is to find the largest documents t_S^{maj} conforming to \mathbb{G} such that, for any document t conforming to \mathbb{G} and

12. This is particularly the case if there is at least one node of the global document accessible by more than one co-author and edited by at least two of them using different productions.

13. Reminder: the editing actions made on a partial replica may be cancelled as long as they do not have been incorporated into the global document.

admitting $t_{\mathcal{V}_i}^{maj}$ as projection along the view \mathcal{V}_i , t_S^{maj} and t are eventually updates each for other. i.e.:

$$\left(t_S^{maj} \in \otimes_{i=1}^k t_i, t_i \in T_{iS}^{maj} \right) \Leftrightarrow \begin{cases} i) \forall i, 1 \leq i \leq k, \forall t \in \mathbb{G} \text{ such that } \pi_{\mathcal{V}_i}(t) = t_{\mathcal{V}_i}^{maj}, t_S^{maj} \cong t. \\ ii) \nexists t' \leq t_S^{maj} \text{ such that } t' \in \otimes_{i=1}^k t_i, t_i \in T_{iS}^{maj} \end{cases} \quad 14$$

The solution that we propose to this problem stems from an instrumentation of that proposed for the expansion (section II.3.1.2). Indeed, we use an associative and commutative operator noted \otimes , to synchronise the tree automata $\mathcal{A}^{(i)}$ constructed to carry out the various expansions, in order to generate the tree automaton of consensual merging. Noting $\mathcal{A}_{(sc)}$ this automaton, the documents of the consensus are the trees of the language generated by the automaton $\mathcal{A}_{(sc)}$ from an initial state built from the vector made of the initial states of the automata ($\mathcal{A}^{(i)}$): $\mathcal{L}(\mathcal{A}_{(sc)}, (q_{T_{\mathcal{V}_i}^{maj}})) = \text{consensus}\{\mathcal{L}(\mathcal{A}^{(i)}, q_{T_{\mathcal{V}_i}^{maj}})\}$. $\mathcal{A}_{(sc)}$ is obtained by proceeding as follows:

- (1) For each view \mathcal{V}_i , build the automaton $\mathcal{A}^{(i)}$ which will carry out the expansion of the partial replica $t_{\mathcal{V}_i}^{maj}$ as previously indicated (sec. II.3.1.2): $\mathcal{L}(\mathcal{A}^{(i)}, q_{T_{\mathcal{V}_i}^{maj}}) = T_{iS}^{maj}$;
- (2) Using the operator \otimes , compute the automaton generating the consensus language $\mathcal{A}_{(sc)} = \otimes_{i=1}^k \mathcal{A}^{(i)}$.

II.3.2.2. Consensus Calculation

Before presenting the consensus calculation algorithm, let us specify using the concepts introduced in section II.3.1.1, the notion of (two) documents in conflict. Let $t_1, t_2 : \mathbb{N}^* \rightarrow \mathbf{A}$ be two trees (documents) with respectively $Dom(t_1)$ and $Dom(t_2)$ their domains. We say that t_1 and t_2 admit a consensus, and we note $t_1 \hat{\wedge} t_2$, if their roots are of the same type¹⁵, i.e. $(t_1 \hat{\wedge} t_2) \Leftrightarrow (\text{typeNode}(t_1(\epsilon)) = \text{typeNode}(t_2(\epsilon)))$ ¹⁶. It is then say that, t_1 and t_2 are in conflict, and it is noted $t_1 \nabla t_2$, when they admit a consensus but are not mergeable in their entirety. Intuitively, two documents t_1

14. The binary relation \cong when it exists between two trees t_1 and t_2 expresses the fact that they are possibly updates each for other. This relationship is more explicitly explained in definition 7.

15. Trees we handle are AST and therefore, the nodes are labelled by productions names. Any node labelled by an X -production is said of type X . Furthermore, there is a function typeNode such that $\text{typeNode}(t(w))$ returns the type of the node located at the address w in t .

16. It may then be noted that two documents (AST) admit no consensus if their roots are of different types. However, for applications that interest us, namely structured editing, since the editions are done from the root (which is always of the type of the axiom) to the leaves using productions, the documents we manipulate always admit at least a consensus.

and t_2 (not reduced to buds) are not fully mergeable (see fig. 24), if there exists an address $w \in Dom(t_1) \cap Dom(t_2)$ such that, if we note n_1 (resp. n_2) the node located to address w in t_1 (resp. in t_2), then, n_1 and n_2 which are not buds, are of the same type but have different labels. i.e.

$$(t_1 \nabla t_2 \text{ with } t_1(\epsilon) \neq X_\omega, t_2(\epsilon) \neq X_\omega) \Leftrightarrow \left\{ \begin{array}{l} (t_1 \wedge t_2) \\ \text{and} \\ \left(\exists w \in Dom(t_1) \cup Dom(t_2), t_1(w) \neq t_2(w) \neq X_\omega \text{ and} \right. \\ \left. typeNode(t_1(w)) = typeNode(t_2(w)) \right) \end{array} \right.$$

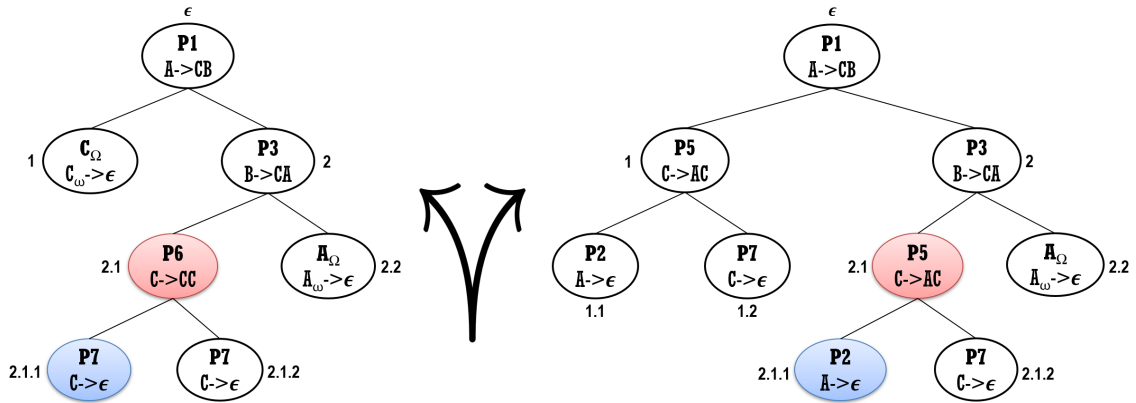


Figure 24 – Example of documents in conflict.

Figure 24 shows two conflicting documents. In fact, at address 2.1 we have two nodes of the same type ("C") but edited with different C-productions: production $C \rightarrow C C$ in the first document, and production $C \rightarrow A C$ in the second one.

Consensus among multiple (two) documents

Let $t_1, t_2 : \mathbb{N}^* \rightarrow \mathbf{A}$ be two trees (documents) in conflict with respectively $Dom(t_1)$ and $Dom(t_2)$ their domains. The consensual tree $t_c : \mathbb{N}^* \rightarrow \mathbf{A}$ derived from t_1 and t_2 ($t_c = t_1 \otimes t_2$) has as domain the union of domains of the two trees in which, we subtract elements belonging to domains of subtrees derived from the conflicting nodes. In fact, we prune at the nodes in conflict and they appear in the consensus tree as a (unique) bud. So,

$$t_c(w) = \begin{cases} \forall w \in \text{Dom}(t_c), \\ \left. \begin{array}{l} t_1(w) \text{ if } \text{typeNode}(t_1(w)) = \text{typeNode}(t_2(w)) \text{ and } t_1(w) = t_2(w) \\ t_1(w) \text{ if } \text{typeNode}(t_1(w)) = \text{typeNode}(t_2(w)) \text{ and } t_2(w) = X_\omega \\ t_2(w) \text{ if } \text{typeNode}(t_1(w)) = \text{typeNode}(t_2(w)) \text{ and } t_1(w) = X_\omega \\ t_1(w) \text{ if } w \notin \text{Dom}(t_2) \text{ and } \exists u, v \in \mathbb{N}^* \text{ tq } w = u.v, \\ \quad t_2(u) = X_\omega \text{ and } \text{typeNode}(t_1(u)) = \text{typeNode}(t_2(u)) \\ t_2(w) \text{ if } w \notin \text{Dom}(t_1) \text{ and } \exists u, v \in \mathbb{N}^* \text{ tq } w = u.v, \\ \quad t_1(u) = X_\omega \text{ and } \text{typeNode}(t_1(u)) = \text{typeNode}(t_2(u)) \\ X_\omega \text{ if } \text{typeNode}(t_1(w)) = \text{typeNode}(t_2(w)) \text{ and } t_1(w) \neq X_\omega \\ \quad \text{and } t_2(w) \neq X_\omega \text{ and } t_1(w) \neq t_2(w) \end{array} \right\} \end{cases}$$

Figure 25 presents the document resulting from the consensual merging of the documents in figure 24. We have pruned at the level of nodes 2.1 in both documents which are in conflict.

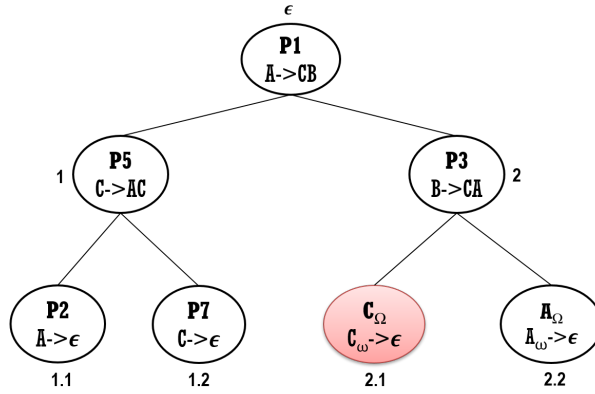


Figure 25 – Document resulting from the consensual merging of the documents in figure 24.

When $t_c = t_1 \otimes t_2$, there may be nodes of t_1 and those of t_2 which are updates of the nodes of t_c : it is said in this case that t_1 (resp. t_2) and t_c are *updates each for other*.

Definition 7 Let t_1, t_2 two documents that are not in conflict. It will be said that they are *updates each for other* and it is noted $t_1 \cong t_2$, if there exists at least two addresses w, w' of their respective domains such that $t_1(w)$ (resp. $t_2(w')$) is a bud and $t_2(w)$ (resp. $t_1(w')$) is not.

Construction of the consensus automaton

Consideration of documents with buds requires the readjustment of some models. For example, in the following, we will handle *tree automata with exit states*

instead of tree automata introduced in definition 5. Intuitively, a state q of an automaton is called an *exit state* if there is a unique transition $q \rightarrow (X_\omega, [])$ associated to it for generating a tree reduced to a bud of type $X \in \Sigma$: q is then of the form $(Open X, [])$.

Definition 8 A *tree automata with exit states* \mathcal{A} is a quintuplet $(\Sigma, Q, R, q_0, exit)$ where Σ, Q, R, q_0 designate the same objects as those introduced in definition 5, and $exit$ is a predicate defined on the states ($exit : Q \rightarrow Bool$). Any state q of Q for which $exit q$ is *True* is an *exit state*.

A type for automata with exit states can be defined in Haskell (Davie, 1992) by the listing of algorithm 1 in which, *state* and *prod* are type variables respectively representing the type of the automaton states and the type of labels of the AST to generate.

Algorithm 1 A Haskell type for automata with exit states.

```

1 data Automata prod state = Auto{
2     exit :: state -> Bool,
3     next :: state -> [(prod, [state])]
4 }
```

In section II.3.2.2 (*consensus among multiple (two) documents*) above, we said that, *when two nodes are in conflict, "they appear in the consensus tree as a (unique) bud"*. From the point of view of automata synchronisation, the concept of "nodes in conflict" is the counterpart of the concept of "states in conflict" (as we specify below), and the above extract is reflected in the automata context by: *"when two state are in conflict, they appear in the consensus automaton in the form of a (single) exit state"*. Thus, if we consider two states of the same type q_0^1 and q_0^2 (which are not exit states) of two automata $auto_1$ and $auto_2$ with associated transitions families respectively $q_0^1 \rightarrow [(a_1^1, qs_1), \dots, (a_{n_1}^1, qs_{n_1})]$ and $q_0^2 \rightarrow [(a_1^2, qs'_1), \dots, (a_{n_2}^2, qs'_{n_2})]$, we say that the states q_0^1 and q_0^2 are in conflict (and we note $q_0^1 \nabla q_0^2$) if there is no transition starting from each of them and with the same label, i.e.

$$(q_0^1 \nabla q_0^2) \Leftrightarrow (\nexists a^3, (a^3, qs) \in \{next q_0^1\}, (a^3, qs') \in \{next q_0^2\}, |qs| = |qs'|)$$

This can be coded in Haskell by the function `isInConflicts` of algorithm 2.

If X is the type of two states q and q' in conflict, they admit a single consensual state $q_\omega = (Open X, [])$ such as $next q_\omega = [(X_\omega, [])]$. It is therefore obvious that two

Algorithm 2 A Haskell function to check if two states of a given automaton are in conflict.

```

1 isInConflicts state1@(tagsymb1, ts1) state2@(tagsymb2, ts2) =
2     null [a1 | (a1, states1) <- next auto1 state1,
3               (a2, states2) <- next auto2 state2,
4               a1==a2,
5               (length states1)==(length states2)]

```

given automata admit a consensual automaton when their initials states are of the same type. The function defined in algorithm 3 performs this test.

Algorithm 3 A Haskell function used to check if two given automata admit a consensus.

```

1 haveConsensus q0 q0' = (typeState q0) == (typeState q0')

```

The operator \otimes used to calculate the synchronised consensual automaton $\mathcal{A}_{(sc)} = \otimes_i^k \mathcal{A}^{(i)}$ is a relaxation of the operator used for calculating the automata product presented in definition 6. $\mathcal{A}_{(sc)} = (\Sigma, Q, R, q_0, exit)$ is an automaton with exit states and is constructed as follows:

- Its states are vectors of states : $Q = Q^{(1)} \times \dots \times Q^{(k)}$;
- its initial state is formed by the vector of initial states of different automata : $q_0 = (q_0^{(1)}, \dots, q_0^{(k)})$;
- For the *exit* function, it is considered that when a given automaton $\mathcal{A}^{(j)}$ reaches an *exit state*¹⁷, it no longer contributes to the behaviour, but is not opposed to the synchronisation of the other automata: it is said to be *asleep* (see algorithm 4, lines 17, 19 and 24). So, a state $q = (q^1, \dots, q^k)$ is an exit state if:
 - (a) all composite states q^i are asleep (see algorithm 4, line 6) or
 - (b) there exist any two states q^i and q^j , $i \neq j$, components of q that are in conflict (see algorithm 4, line 12)

$$(exit (q^{(1)}, \dots, q^{(k)})) \Leftrightarrow ((exit q^{(i)}, \forall i \in \{1 \dots k\}) \text{ or } (\exists i, j, i \neq j, q^{(i)} \nabla q^{(j)}));$$
- Its transitions are given by:

17. The corresponding node in the reverse projection of the document is a bud and reflects the fact that, the corresponding author did not edit it. In the case that this node is shared with another co-author who edited it in its (partial) replica, it is the edition made by the latter that will be retained when merging.

- (a) If *exit* q then $q \rightarrow (X_\omega, [])$ is the unique transition of q ; X is the type of q .
- (b) Else $(q^{(1)}, \dots, q^{(k)}) \xrightarrow{a} \left((q_1^{(1)}, \dots, q_1^{(k)}), \dots, (q_n^{(1)}, \dots, q_n^{(k)}) \right) \Leftrightarrow + \forall i, 1 \leq i \leq k$
 - (b1) *exit* $q^{(i)}$ and $(q_j^{(i)} = (\text{Open } X, []), \forall j, 1 \leq j \leq n)$ /* $q^{(i)}$ is asleep */ , else
 - (b2) $q^{(i)} \xrightarrow{a} (q_1^{(i)}, \dots, q_n^{(i)})$.

(a) reflects the fact that, if a state q is an exit one, we associate a single transition for generating a tree reduced to a bud of the type of q (see algorithm 4, line 12).

With (b1) we say that, if the component $q^{(i)}$ of q is an exit state, then for all composite state $(q_j^{(1)}, \dots, q_j^{(k)})$, $(1 \leq j \leq n)$ appearing in the right hand side of the transition (b), the i^{th} component should be asleep. Since it must not prevent other non-asleep states to synchronise, it must be of the form $(\text{Open } X, [])$ where X is the type of the other states $q_j^{(l)}$ (yet to be synchronised) belonging to $(q_j^{(1)}, \dots, q_j^{(k)})$ (see function *fwdSlpState* defined in algorithm 4 line 24, and used in lines 17 and 19). Finally, with (b2) we stipulate that if $q^{(i)} \xrightarrow{a} (q_1^{(i)}, \dots, q_n^{(i)})$ is a transition of the automaton \mathcal{A}^i , then for all composite state $(q_j^{(1)}, \dots, q_j^{(k)})$, $(1 \leq j \leq n)$ appearing in the right part of the transition (b) above, the i^{th} component is $q_j^{(i)}$ (see algorithm 4, lines 13 to 16).

Proposition 9 *The tree automaton $\mathcal{A} = \otimes_{i=1}^k \mathcal{A}^{(i)}$ recognises/generates from the initial state $q_0 = (q_{01}, \dots, q_{0k})$, all the trees from the consensual merging of trees recognised/generated by each automaton $\mathcal{A}^{(i)}$ from the initial state q_{0i} . Moreover, these trees are the biggest prefixes without conflicts of merged trees.*

$$\left(\otimes_{i=1}^k \mathcal{A}^{(i)} \models t \triangleright q_0 \right) \Leftrightarrow \begin{cases} i) \forall i, \exists t_i \quad \mathcal{A}^{(i)} \models t_i \triangleright q_{0i} \text{ and } t_i \cong t \\ ii) \forall t' \text{ prefix of } t, \quad \neg \left(\otimes_{i=1}^k \mathcal{A}^{(i)} \models t' \triangleright q_0 \right) \end{cases}$$

Proof. A tree t is recognised by the synchronised automaton $\otimes_{i=1}^k \mathcal{A}^{(i)}$ if and only if, one can label each of its nodes by a state of the automaton in accordance with what is specified by the transitions of the automaton. Moreover, all the leaf nodes of t must be labelled by using *final transitions*; in our case, they are of the form $q \rightarrow (p, [])$. This means that, if a node whose initial label is a , is labelled by the state q , and if it admits n successors respectively labelled by q_1, \dots, q_n , then

Algorithm 4 Consensus Listing.

```

1 autoconsens::(Eq p, Eq x) =>(x -> p) -> Automata p (Tag x, [st1])
2   -> Automata p (Tag x, [st2])
3   -> Automata p ((Tag x, [st1]), (Tag x, [st2]))
4 autoconsens symb2prod auto1 auto2 = Auto exit_ next_ where
5   exit_ (state1, state2) = case haveConsensus state1 state2 of
6     True  -> (exit auto1 state1) && (exit auto2 state2)
7     False -> True
8   next_ (state1, state2) = case haveConsensus state1 state2 of
9     False -> []
10    True  -> case (exit auto1 state1, exit auto2 state2) of
11      (False, False) -> case (isInConflicts state1 state2) of
12        True  -> [(symb2prod (typeState state1), [])]
13        False -> [(a1, zip states1 states2) |
14                  (a1, states1) <- next auto1 state1,
15                  (a2, states2) <- next auto2 state2,
16                  a1 == a2, (length states1) == (length states2)]
17      (False, True)  -> [(a, zip states1 (fwdSlpState states1)) |
18                        (a, states1) <- next auto1 state1]
19      (True, False)  -> [(a, zip (fwdSlpState states2) states2) |
20                        (a, states2) <- next auto2 state2]
21      (True, True)   -> [(a1, []) | (a1, []) <- next auto1 state1,
22                        (a2, []) <- next auto2 state2, a1 == a2]
23   where
24     fwdSlpState states = map (\state -> (Open (typeState state), [])) states

```

$q \xrightarrow{a} (q_1, \dots, q_n)$ must be a transition of the automaton. As the automaton is deterministic¹⁸, this labelling is unique (including the initial state attached to the root of the tree). By focusing our attention both on the state q labelling a node, and its i^{th} component q_i , on each of the branches of t ,

- (1) we cut as soon as we reach an exit state in relation to the automaton $\mathcal{A}^{(i)}$ (i.e. q_i is an exit state), or,
- (2) if q is an exit state (in this case we are handling a leaf) and q_i is not an exit state relatively to $\mathcal{A}^{(i)}$ (in this case, q_i was in conflict with at least one other component q_j of q), we replace that node with any subtree t'_i that can be generated by $\mathcal{A}^{(i)}$ from the state q_i .

So,

$$\left(\bigotimes_{i=1}^k \mathcal{A}^{(i)} \models t \triangleright q_0 \right) \Rightarrow \left(\forall i, \exists t_i \quad \mathcal{A}^{(i)} \models t_i \triangleright q_{0i} \text{ and } t_i \cong t \right)$$

18. Automata $\mathcal{A}^{(i)}$ being deterministic (see proposition 3.3.3 of (Tchoupé Tchendji, 2009)), $\bigotimes_{i=1}^k \mathcal{A}^{(i)}$ is deterministic as synchronous product of deterministic automata.

since a state of \mathcal{A} is an exit one if and only if, each of its components is an exit state (in the \mathcal{A}^i) or, if at least two of its components are in conflict.

Conversely, suppose $\mathcal{A}^{(i)} \models t_i \triangleright q_{0i}$, by definition of the synchronised consensual automaton, we have $\otimes_{i=1}^k \mathcal{A}^{(i)} \models \otimes_{i=1}^k t_i \triangleright (q_{01}, \dots, q_{0k})$. So overall,

$$L\left(\otimes_{i=1}^k \mathcal{A}^{(i)}, q_0\right) = \left\{ \otimes_{i=1}^k t_i \mid \mathcal{A}^{(i)} \models t_i \triangleright q_{0i} \right\}$$

Suppose that t is recognised by $\otimes_{i=1}^k \mathcal{A}^{(i)}$; thus, there is a labelling of its nodes with the states of $\otimes_{i=1}^k \mathcal{A}^{(i)}$, and as such the transitions used for the labelling of its leaves are final. Let t_p be a prefix of t . Let us show that t_p is not recognised by $\otimes_{i=1}^k \mathcal{A}^{(i)}$ using the fact that, any labelling of t_p has at least one leaf node labelled by a state that is not associated to a final transition. The labels associated to the nodes of t_p are the same as those associated to the nodes of same addresses in t because, t_p is a prefix of t and $\otimes_{i=1}^k \mathcal{A}^{(i)}$ is deterministic. t_p is obtained from t by pruning some subtrees of t ; so, naturally, it has a (non-zero) number of leaf nodes that can be developed to obtain t . Let us choose such a node and call it n_f . Suppose that it is labelled p and was associated with a state $q_f = (q_1, \dots, q_k)$ when labelling t . The p -transition that allowed to recognise n_f is not a final transition. Indeed, n_f has in t , $|p|$ sons whose labels can be supposed to be the states $q_{f_1}, \dots, q_{f_{|p|}}$. This means that, according to the labelling process and considering the fact that $\otimes_{i=1}^k \mathcal{A}^{(i)}$ is deterministic, the single transition used for the labelling of n_f and of its $|p|$ sons is $q_f \xrightarrow{p} (q_{f_1}, \dots, q_{f_{|p|}})$ which, is not a final transition. Therefore, t_p is not recognised by $\otimes_{i=1}^k \mathcal{A}^{(i)}$. \square

II.3.2.3. Illustration

Figure 26 is an illustration of an asynchronous cooperative editing process generating partial replicas (fig. 26(c) and fig. 26(e)) in conflict¹⁹ from the grammar having as productions:

$$\begin{aligned} P_1 : A \rightarrow C B & \quad P_3 : B \rightarrow C A & \quad P_5 : C \rightarrow A C \\ P_2 : A \rightarrow \varepsilon & \quad P_4 : B \rightarrow B B & \quad P_6 : C \rightarrow C C \\ & & \quad P_7 : C \rightarrow \varepsilon \end{aligned}$$

Initially in the process, two partial replicas (fig. 26(b) and fig. 26(d)) are obtained by projections of the global document (fig. 26(a)). After their update (fig. 26(c) and fig. 26(e)) a synchronisation point is reached and, by applying the approach described in section II.3.2.1, a consensus document is found (fig. 26(h)).

19. By realising expansions of each of the replicas, we respectively obtain among others, the documents presented by figure 26(f) and figure 26(g) on which, one can easily observe a conflict highlighted by areas having a blue background.

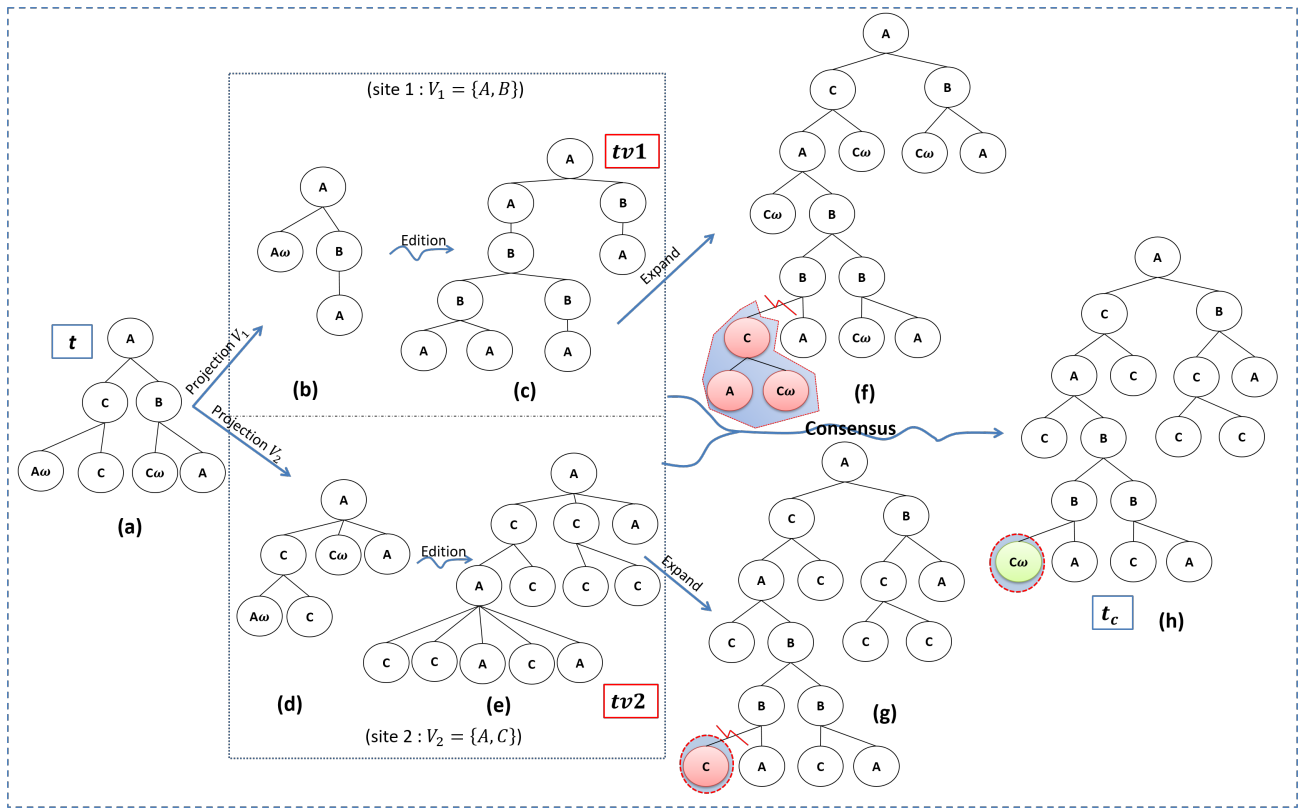


Figure 26 – An edition with conflicts and corresponding consensus.

More precisely, as detailed below, we associate the automata $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$ respectively to the updated partial replicas $tv1$ and $tv2$ (fig. 26(c) and fig. 26(e)), then we build the automaton of consensus $\mathcal{A}_{(sc)} = \mathcal{A}^{(1)} \otimes \mathcal{A}^{(2)}$ by applying the approach described in section II.3.2.2 (*construction of the consensus automaton*) and finally, we generate the simplest documents of the consensus (fig. 28) among which is the document in figure 26(h).

Linearisation of a structured document

To simplify the presentation, we represent in the following, trees by their linearisation in the form of a Dyck word. To do this, we associate a (various) pair of brackets to each grammatical symbol and the linearisation of a tree is obtained by performing a Depth First Search (DFS) of the resulting tree (see fig. 27).

The transition schemas for the view $\{A, B\}$

A list of trees (forest) is represented by the concatenation of their linearisation. We use the opening parenthesis '(' and the closing one ')' to represent Dyck symbols associated with the visible symbol A , and the opening bracket '[' and the closing one ']' to represent those associated with the visible symbol B . Each tran-

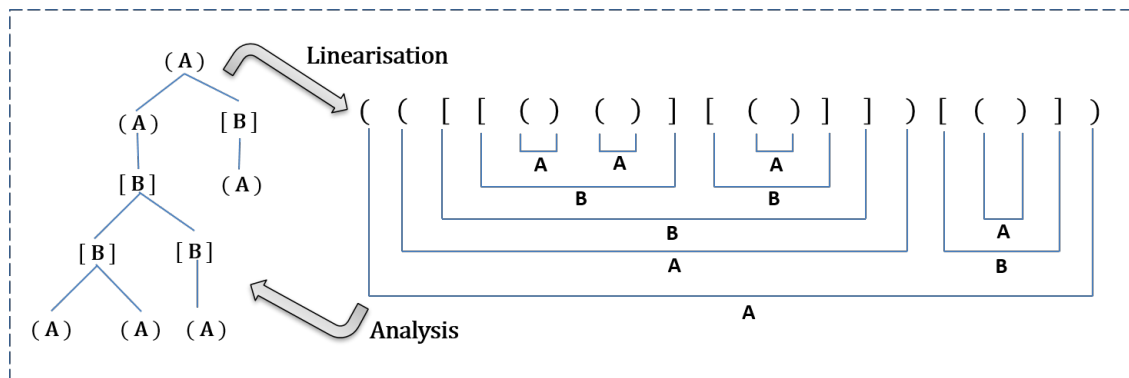


Figure 27 – Linearisation of a tree ($tv1$): the Dyck symbols '(' and ')' (resp. '[' and ']') have been associated with the grammatical symbol A (resp. B).

sition of the automata associated to partial replicas according to the view $\{A, B\}$ is conform to one of the transition schemas²⁰ in table II. These schemas are obtained

Table II – The transition schemas for the view $\{A, B\}$.

| | | | |
|--------------------------|-------------------|---|--------------------------------|
| $\langle A, w_1 \rangle$ | \longrightarrow | $(P_1, [\langle C, u \rangle, \langle B, v \rangle])$ | if $w_1 = u[v]$ |
| $\langle A, w_2 \rangle$ | \longrightarrow | $(P_2, [])$ | if $w_2 = \varepsilon$ |
| $\langle B, w_3 \rangle$ | \longrightarrow | $(P_3, [\langle C, u \rangle, \langle A, v \rangle])$ | if $w_3 = u(v)$ |
| $\langle B, w_4 \rangle$ | \longrightarrow | $(P_4, [\langle B, u \rangle, \langle B, v \rangle])$ | if $w_4 = [u][v]$ |
| $\langle C, w_5 \rangle$ | \longrightarrow | $(P_5, [\langle A, u \rangle, \langle C, v \rangle])$ | if $w_5 = (u)v$ |
| $\langle C, w_6 \rangle$ | \longrightarrow | $(P_6, [\langle C, u \rangle, \langle C, v \rangle])$ | if $w_6 = uv \neq \varepsilon$ |
| $\langle C, w_7 \rangle$ | \longrightarrow | $(C_\omega, [])$ | if $w_7 = \varepsilon$ |

from the grammar productions (Badouel and Tchoupé Tchendji, 2008), and the pairs $\langle X, w_i \rangle$ are states where X is a grammatical symbol and w_i a forest encoded in the Dyck language. The first schema for example, states that the AST generated from the state $\langle A, w_1 \rangle$ are those obtained using the production P_1 to create a tree of the form $P_1[t_1, t_2]$; t_1 and t_2 being generated respectively from the states $\langle C, u \rangle$ and $\langle B, v \rangle$ such that $w_1 = u[v]$. The state $\langle C, w_7 \rangle$ with $w_7 = \varepsilon$ being an exit state, the rule $\langle C, w_7 \rangle \longrightarrow (C_\omega, [])$ linked to the production P_7 states that, the AST generated from the state $\langle C, w_7 \rangle$ is reduced to a bud of type C (C is the symbol located at the left hand side of P_7).

Construction of the automaton $\mathcal{A}^{(1)}$ associated to $tv1$

Having associated Dyck symbols '(' and ')' (resp. '[' and ']') to the grammatical symbol A (resp. B), the linearisation of the partial replica $tv1$ (fig. 26(c)) gives

20. We do not represent the whole set of transition schemas in this example; only the useful subset for reconciliation of closed documents is shown here because the documents to reconcile in this example are all closed (has no buds). To consider buds, one should, for each visible sort X , associate a new pair of Dyck symbols to the bud of type X then, derive the new schemas.

$(([[()()]]([()]))([()]))$. As A is the axiom of the grammar, the initial state of the automaton $\mathcal{A}^{(1)}$ is $q_0^1 = \langle A, ([[()()]]([()]))([()]))$. When considering only the states accessible from q_0^1 and by applying the previous schema of transition, we obtain the automaton in table III, for the replica $tv1$ (fig. 26(c)). The state $q_4^1 = \langle C, \varepsilon \rangle$ is the only exit

Table III – The tree automaton associated to the replica $tv1$.

| | |
|--|---|
| $q_0^1 \longrightarrow (P_1, [q_1^1, q_2^1])$ | with $q_1^1 = \langle C, ([[()()]]([()])) \rangle$ and $q_2^1 = \langle B, () \rangle$ |
| $q_1^1 \longrightarrow (P_5, [q_3^1, q_4^1])$ | with $q_3^1 = \langle A, [[()()]]([()]) \rangle$ and $q_4^1 = \langle C, \varepsilon \rangle$ |
| $q_1^1 \longrightarrow (P_6, [q_4^1, q_1^1]) \mid (P_6, [q_1^1, q_4^1])$ | |
| $q_2^1 \longrightarrow (P_3, [q_4^1, q_5^1])$ | with $q_5^1 = \langle A, \varepsilon \rangle$ |
| $q_3^1 \longrightarrow (P_1, [q_4^1, q_6^1])$ | with $q_6^1 = \langle B, [()()]]([()]) \rangle$ |
| $q_4^1 \longrightarrow (C_\omega, [])$ | |
| $q_5^1 \longrightarrow (P_2, [])$ | |
| $q_6^1 \longrightarrow (P_4, [q_7^1, q_2^1])$ | with $q_7^1 = \langle B, ()() \rangle$ |
| $q_7^1 \longrightarrow (P_3, [q_8^1, q_5^1])$ | with $q_8^1 = \langle C, () \rangle$ |
| $q_8^1 \longrightarrow (P_5, [q_5^1, q_4^1])$ | |
| $q_8^1 \longrightarrow (P_6, [q_8^1, q_4^1]) \mid (P_6, [q_4^1, q_8^1])$ | |

state of $\mathcal{A}^{(1)}$. It is easy to verify that the document of figure 26(f) resulting from the reverse projection of $tv1$, belongs to the language accepted by the automaton $\mathcal{A}^{(1)}$.

Construction of the automaton $\mathcal{A}^{(2)}$ associated to $tv2$

As before, by associating to the grammatical symbol C (resp. A) the Dyck symbols '[' and ']' (resp. '(' and)'), we obtain the transition schemas for the automata associated to the partial replicas according to the view $\{A, C\}$ (see table IV).

Table IV – The transition schemas for the view $\{A, C\}$.

| | |
|--|--------------------------------|
| $\langle A, w_1 \rangle \longrightarrow (P_1, [\langle C, u \rangle, \langle B, v \rangle])$ | if $w_1 = [u]v$ |
| $\langle A, w_2 \rangle \longrightarrow (P_2, [])$ | if $w_2 = \varepsilon$ |
| $\langle B, w_3 \rangle \longrightarrow (P_3, [\langle C, u \rangle, \langle A, v \rangle])$ | if $w_3 = [u](v)$ |
| $\langle B, w_4 \rangle \longrightarrow (P_4, [\langle B, u \rangle, \langle B, v \rangle])$ | if $w_4 = uv \neq \varepsilon$ |
| $\langle B, w_5 \rangle \longrightarrow (B_\omega, [])$ | if $w_5 = \varepsilon$ |
| $\langle C, w_6 \rangle \longrightarrow (P_5, [\langle A, u \rangle, \langle C, v \rangle])$ | if $w_6 = (u)[v]$ |
| $\langle C, w_7 \rangle \longrightarrow (P_6, [\langle C, u \rangle, \langle C, v \rangle])$ | if $w_7 = [u][v]$ |
| $\langle C, w_8 \rangle \longrightarrow (P_7, [])$ | if $w_8 = \varepsilon$ |

The linearisation of the partial replica $tv2$ (fig. 26(e)) is $(([[[]]]([()])[[[]]]([()]))$. The automaton $\mathcal{A}^{(2)}$ associated to this replica has as initial state $q_0^2 = \langle A, ([[[]]]([()])[[[]]]([()]))$.

Table V – The tree automaton associated to the replica tv2.

| | |
|--|---|
| $q_0^2 \longrightarrow (P_1, [q_1^2, q_2^2])$ | with $q_1^2 = \langle C, ([[]]())[] \rangle$ and $q_2^2 = \langle B, [[]] \rangle$ |
| $q_1^2 \longrightarrow (P_5, [q_3^2, q_4^2])$ | with $q_3^2 = \langle A, ([[]]())[] \rangle$ and $q_4^2 = \langle C, \varepsilon \rangle$ |
| $q_2^2 \longrightarrow (P_3, [q_5^2, q_6^2])$ | with $q_5^2 = \langle C, [[]] \rangle$ and $q_6^2 = \langle A, \varepsilon \rangle$ |
| $q_3^2 \longrightarrow (P_1, [q_4^2, q_7^2])$ | with $q_7^2 = \langle B, []()[] \rangle$ |
| $q_4^2 \longrightarrow (P_7, [])$ | |
| $q_5^2 \longrightarrow (P_6, [q_4^2, q_4^2])$ | |
| $q_6^2 \longrightarrow (P_2, [])$ | |
| $q_7^2 \longrightarrow (P_4, [q_8^2, q_7^2]) \mid (P_4, [q_9^2, q_{10}^2]) \mid (P_4, [q_{11}^2, q_{11}^2]) \mid (P_4, [q_{12}^2, q_{13}^2]) \mid (P_4, [q_7^2, q_8^2])$ | with $q_8^2 = \langle B, \varepsilon \rangle, q_9^2 = \langle B, [] \rangle, q_{10}^2 = \langle B, ()[] \rangle, q_{11}^2 = \langle B, []() \rangle, q_{12}^2 = \langle B, []()[] \rangle$ and $q_{13}^2 = \langle B, () \rangle$ |
| $q_8^2 \longrightarrow (B_\omega, [])$ | |
| $q_9^2 \longrightarrow (P_4, [q_8^2, q_9^2]) \mid (P_4, [q_9^2, q_8^2])$ | |
| $q_{10}^2 \longrightarrow (P_4, [q_8^2, q_{10}^2]) \mid (P_4, [q_{13}^2, q_{11}^2]) \mid (P_4, [q_{14}^2, q_{13}^2]) \mid (P_4, [q_{10}^2, q_8^2])$ | with $q_{14}^2 = \langle B, ()[] \rangle$ |
| $q_{11}^2 \longrightarrow (P_3, [q_4^2, q_6^2])$ | |
| $q_{12}^2 \longrightarrow (P_4, [q_8^2, q_{12}^2]) \mid (P_4, [q_9^2, q_{14}^2]) \mid (P_4, [q_{11}^2, q_9^2]) \mid (P_4, [q_{12}^2, q_8^2])$ | |
| $q_{13}^2 \longrightarrow (P_4, [q_8^2, q_{13}^2]) \mid (P_4, [q_{13}^2, q_8^2])$ | |
| $q_{14}^2 \longrightarrow (P_4, [q_8^2, q_{14}^2]) \mid (P_4, [q_{13}^2, q_9^2]) \mid (P_4, [q_{14}^2, q_8^2])$ | |

and its transitions are the ones in table V. The state $q_8^2 = \langle B, \varepsilon \rangle$ is the only exit state of the automaton $\mathcal{A}^{(2)}$.

Construction of the consensus automaton $\mathcal{A}_{(sc)}$

By application of synchronous product of several tree automata described in section II.3.2.2 (*construction of the consensus automaton*) to the automata $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$, the consensual automaton $\mathcal{A}_{(sc)} = \mathcal{A}^{(1)} \otimes \mathcal{A}^{(2)}$ has $q_0 = (q_0^1, q_0^2)$ as initial state. $\mathcal{A}^{(1)}$ has a transition from q_0^1 to $[q_1^1, q_2^1]$ labelled P_1 . Similarly, $\mathcal{A}^{(2)}$ has a transition from q_0^2 to $[q_1^2, q_2^2]$ labelled P_1 . So, we have in $\mathcal{A}_{(sc)}$, a transition labelled P_1 for accessing states $[q_1 = (q_1^1, q_1^2), q_2 = (q_2^1, q_2^2)]$ from the initial state $q_0 = (q_0^1, q_0^2)$. Following this principle, we construct the consensual automaton in table VI.

The states $\{q_{10}, q_{11}, q_{12}, q_{15}, q_{16}, q_{17}, q_{21}, q_{22}\}$ are the exit states of the automaton $\mathcal{A}_{(sc)}$. They are states whose composite states are either in conflict (for example $q_{10} = (q_2^1, q_7^2)$ et $q_2^1 \nabla q_7^2$), or are all exit states (for example $q_{22} = (q_4^1, q_{s1})$).

The use of the function that generates the simplest AST (with buds) of a tree language from its automaton (Badouel and Tchoupé Tchendji, 2008) on $\mathcal{A}_{(sc)}$, pro-

Table VI – *The consensual tree automaton.*

| | | |
|--|------|---|
| | | $q_0 = (q_0^1, q_0^2)$ |
| $q_0 \longrightarrow (P_1, [q_1, q_2])$ | with | $q_1 = (q_1^1, q_1^2)$ and $q_2 = (q_2^1, q_2^2)$ |
| $q_1 \longrightarrow (P_5, [q_3, q_4])$ | with | $q_3 = (q_3^1, q_3^2)$ and $q_4 = (q_4^1, q_4^2)$ |
| $q_2 \longrightarrow (P_3, [q_5, q_6])$ | with | $q_5 = (q_5^1, q_5^2)$ and $q_6 = (q_6^1, q_6^2)$ |
| $q_3 \longrightarrow (P_1, [q_4, q_7])$ | with | $q_7 = (q_7^1, q_7^2)$ |
| $q_4 \longrightarrow (P_7, [])$ | | |
| $q_5 \longrightarrow (P_6, [q_8, q_8])$ | with | $q_8 = (q_{s1}, q_4^2)$ and $q_{s1} = \langle Open C, [] \rangle$ |
| $q_6 \longrightarrow (P_2, [])$ | | |
| $q_7 \longrightarrow (P_4, [q_9, q_{10}]) \mid (P_4, [q_{11}, q_{12}]) \mid (P_4, [q_{13}, q_{14}]) \mid (P_4, [q_{15}, q_{16}]) \mid (P_4, [q_{17}, q_{18}])$ | with | $q_9 = (q_7^1, q_8^2), q_{10} = (q_2^1, q_7^2), q_{11} = (q_7^1, q_9^2), q_{12} = (q_2^1, q_{10}^2), q_{13} = (q_7^1, q_{11}^2), q_{14} = (q_2^1, q_{11}^2), q_{15} = (q_7^1, q_{12}^2), q_{16} = (q_2^1, q_{13}^2), q_{17} = (q_7^1, q_7^2)$ and $q_{18} = (q_2^1, q_8^2)$ |
| $q_8 \longrightarrow (P_7, [])$ | | |
| $q_9 \longrightarrow (P_3, [q_{19}, q_{20}])$ | with | $q_{19} = (q_8^1, q_{s1})$ and $q_{20} = (q_5^1, q_{s2}), q_{s2} = \langle Open A, [] \rangle$ |
| $q_{13} \longrightarrow (P_3, [q_{21}, q_6])$ | with | $q_{21} = (q_8^1, q_4^2)$ |
| $q_{14} \longrightarrow (P_3, [q_4, q_6])$ | | |
| $q_{18} \longrightarrow (P_3, [q_{22}, q_{20}])$ | with | $q_{22} = (q_4^1, q_{s1})$ |
| $q_{19} \longrightarrow (P_5, [q_{20}, q_{22}]) \mid (P_6, [q_{19}, q_{22}]) \mid (P_6, [q_{22}, q_{19}])$ | | |
| $q_{20} \longrightarrow (P_2, [])$ | | |
| $q_{10} \longrightarrow (B_\omega, [])$ | | |
| $q_{11} \longrightarrow (B_\omega, [])$ | | |
| $q_{12} \longrightarrow (B_\omega, [])$ | | |
| $q_{15} \longrightarrow (B_\omega, [])$ | | |
| $q_{16} \longrightarrow (B_\omega, [])$ | | |
| $q_{17} \longrightarrow (B_\omega, [])$ | | |
| $q_{21} \longrightarrow (C_\omega, [])$ | | |
| $q_{22} \longrightarrow (C_\omega, [])$ | | |

duces *four* AST whose derivation trees (the consensus) are shown in figure 28.

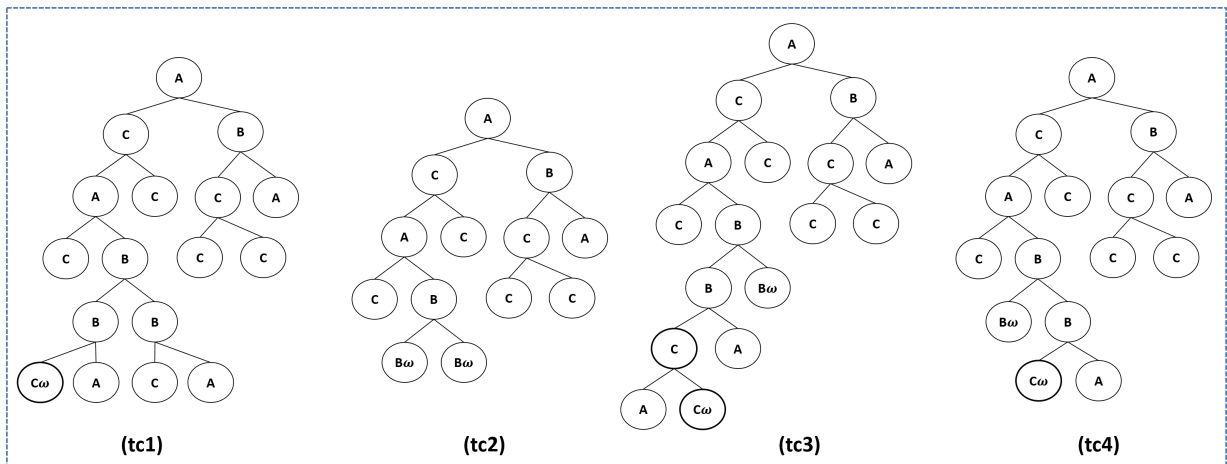


Figure 28 – Consensual trees generated from the automaton $\mathcal{A}_{(sc)}$

II.4. A Software Architecture for Centralised Management of Structured Documents in a Cooperative Editing Workflow

In this section, we will focus on the implementation of a system that can support cooperative editing as perceived by Badouel and Tchoupé. This effort is motivated by the fact that:

1. *This type of editing workflow applies to structured documents:* this leads to the fact that, one can locally perform validations in accordance with a local model derived from the global one;
2. *This type of editing workflow is particularly compatible with administrative workflows:* concepts of "view" and partial replica introduced by Badouel and Tchoupé, make that the type of workflow they offer is particularly adapted for the specification of many administrative processes. Consider, for example, the process "tracking a medical record in a health center with the reception and consultation services": the aforesaid record can be modelled as a structured document in which the members of the host service (reception) cannot view and/or modify certain information contained therein; those information, requesting the expertise of the consulting staff for example. Therefore, one can associate views to each of these services. It is left only to specify the medical record's circuit and an editing workflow of the type described in the previous section is obtained;
3. *A generic architectural model describing precisely an approach for the implementation of this type of workflow does not exist:* the only prototype (Tchoupé Tchendji, 2010) which was designed around the concepts handled

(view, partial replica, merging, etc.) for this type of workflow, was more of a graphic tool (editor) for the experimentation of concepts and algorithms presented in (Badouel and Tchoupé Tchendji, 2008); workflow management is not addressed in it: this tool cannot be used to specify an editing workflow, it does not support routing or storage of artifacts, nothing is done concerning monitoring, etc., yet these concerns are among the most important to be taken care of by a workflow management infrastructure (IMA, 2002).

II.4.1. The Proposed Architecture

II.4.1.1. Overall Operations

The architecture that we propose is composed of three tiers: some *clients*, a *central server* and several *administration tools*. We consider that, each participant in a given workflow has a client. Initially, the workflow owner (comparable to a deposit owner in Git) connects to the server from his client. He creates his workflow by specifying all necessary informations (the workflow name, the overall grammar, different participants, their rights and their views, the basic document and the workflow's circuit), then triggers the process. Next, participants concerned by the newly created workflow receive an alert message from the system, inviting them to participate. Each participant must therefore connect himself to the server to obtain a partial replica of the workflow model (encoded in a specification file written in a dedicated DSL) and state (his local document model, a partial replica of the initial document, etc.) according to his rights and his view on the given workflow. A given participant performs his duties and submits his local (partial) replica to the central server which performs synchronisations as soon as possible and the process continues (see fig. 20) until the end. For specific needs (authentication, access to corporate data, etc.), clients and server may require the intervention of an administration tool (database, paperwork and many others). These three tiers are interconnected around a middleware as presented in figure 29.

II.4.1.2. Server Architecture

The server is responsible for the storage, restoration, execution and monitoring of workflows. Its architecture is based on three basic elements as shown in figure 29(a) : its *model*, *storage module* and its *runtime engine*.

1. *The model*: it is the one orchestrating all the tasks supported by the server. It consists of a workflow engine, a set of parsers and three communication

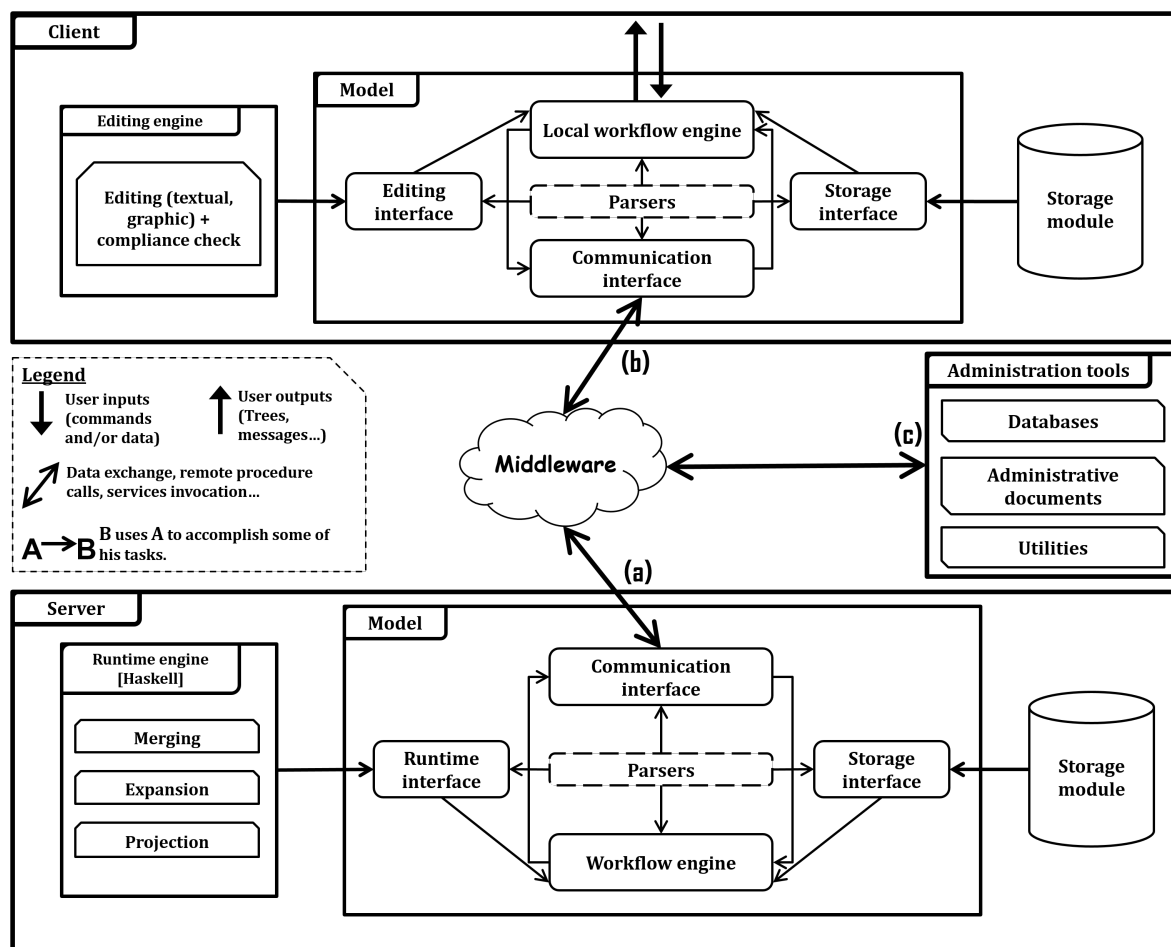


Figure 29 – A software architecture (three-tiers) for centralised management of structured documents' cooperative editing workflows.

interfaces (the interface with the middleware, that with the storage module and the one with the runtime engine).

2. *The storage module*: it is responsible for the storage of workflows. Like CVS, it maintains a main repository for each workflow. The repository space of a given workflow includes its specification file written in a DSL. There are also (global) document versions showing the state of the workflow at given times. These versions of the underlying documents, facilitate the control and monitoring of workflows.
3. *The runtime engine*: it consists of implementations of projection, expansion and consensual merging algorithms. These implementations are used by the workflow engine in the realisation of these tasks. A runtime engine written entirely in Haskell, was proposed in (Tchoupé Tchendji, 2010). However, it is quite rigid and almost impossible to adapt to the architecture presented

here. To this end, we present in section II.4.2.2, a more flexible version of the latter.

II.4.1.3. Client Architecture

The client (figure 29(b)) is also based on three entities: a *model*, an *editing engine* and a *storage module*. The model is responsible for organising and controlling the execution of tasks and user commands. For each new local workflow, the model generates an editing environment which is used by the editing engine to provide conventional facilities of structured document editors (compliance check, syntax highlighting, graphical editing of documents presentations, etc.). Each workflow is locally represented by a specification file and by one structured document representing the current perception of the overall workflow from the current local site. When reaching synchronisation phases, the local structured document is forwarded to the server site, where it is merged with others, in one structured document representing the current state of the overall workflow : it is therefore, a coordination support between the workflow engines of the client and of the server.

II.4.1.4. The Middleware

The middleware is responsible for the interaction between different tiers of our architecture. It must be designed so that, the coupling between these tiers is as weak as possible. One can for this purpose, consider a SOA in which:

- Our clients are service clients;
- The server is a service provider for clients and a client of services offered by the administration tools;
- The administration tools are service providers.

With such an architecture, we can guarantee the independence of each tier and thus, an easier maintenance.

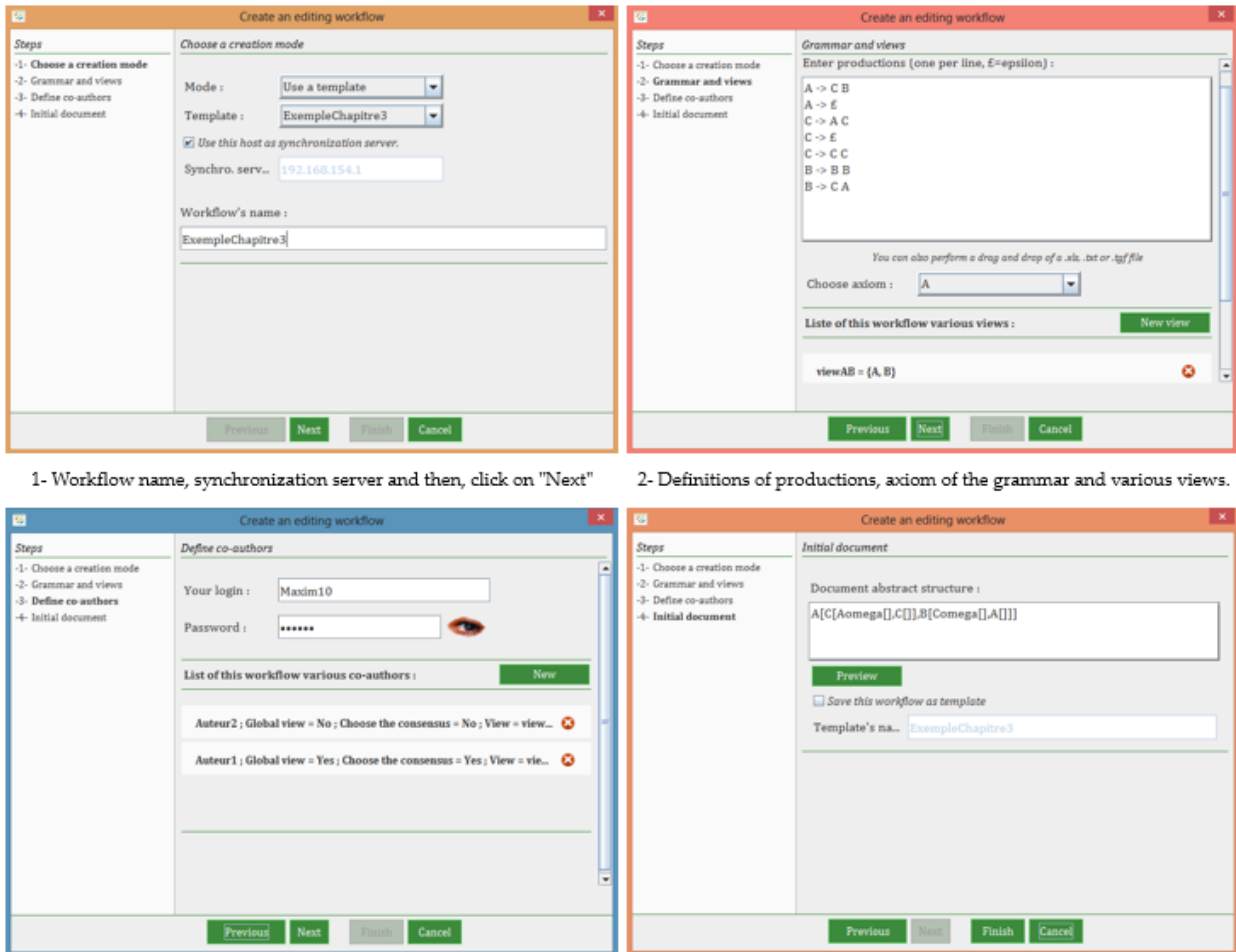
II.4.2. TinyCE v2

II.4.2.1. Presentation of TinyCE v2

Due to its technical nature and to the number of technologies it needs for its instantiation, the architecture presented above has not yet been fully implemented. However, many of its components have already been implemented and tested in a test project called TinyCE v2²¹ (a Tiny Cooperative Editor version 2).

21. TinyCE v2 is a more advanced version of TinyCE (Tchoupé Tchendji, 2010).

TinyCE v2 is an editor prototype providing graphic and cooperative editing of the abstract structure of structured documents. It is used following a networked client-server model. Its user interface offers to the user, facilities for the creation of workflows (documents, grammars, actors and views (see fig. 30)), edition and validation of partial replicas (see fig. 31). Moreover, this interface also offers the functionality to experiment the concepts of projection, expansion and consensual merging (see fig. 32). TinyCE v2 is designed using Java and Haskell languages. It offers several implementations of our architecture concepts namely: parsers, storage modules, server's runtime engine, workflow engines and communication interfaces.



1- Workflow name, synchronization server and then, click on "Next"

2- Definitions of productions, axiom of the grammar and various views.

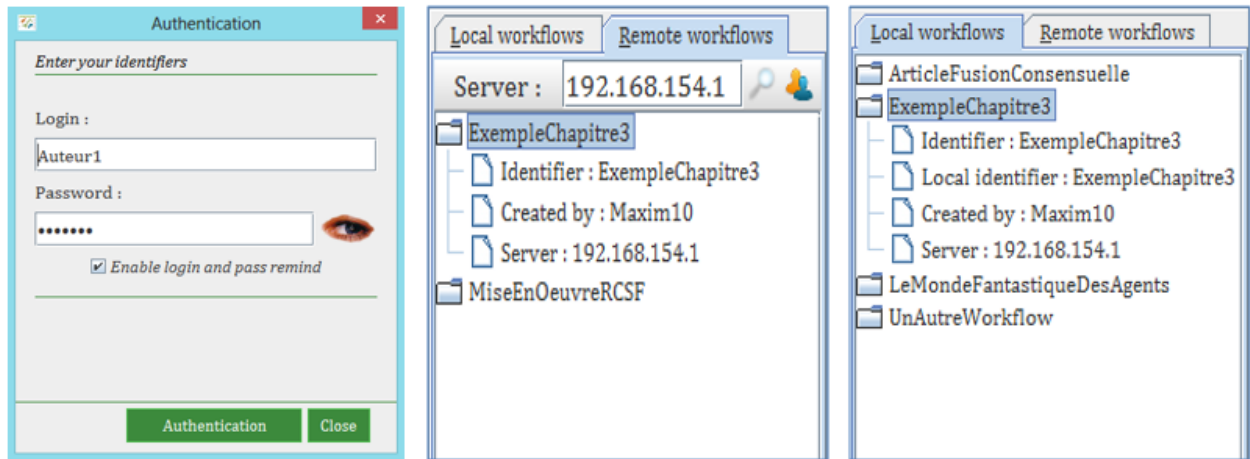
3- Informations on workflow owner and on different co-authors

4- Creation of the global initial document

Figure 30 – Some screenshots showing the creation process of a cooperative editing workflow in TinyCE v2.

II.4.2.2. Java-Haskell Cross-Fertilisation in TinyCE v2

As in (Tchoupé Tchendji, 2010), the runtime engine of TinyCE v2 exploits the possibility offered by Java, to run an "external program". Indeed, we designed an



1- Authentication of the co-author Auteur1

2- Showing available remote workflows for Auteur1

3- Displaying local workflows of Auteur1

Figure 31 – Some screenshots of TinyCE v2 showing the authentication window of a co-author (Auteur1) as well as those displaying the various local and remote workflows in which he is implicated.

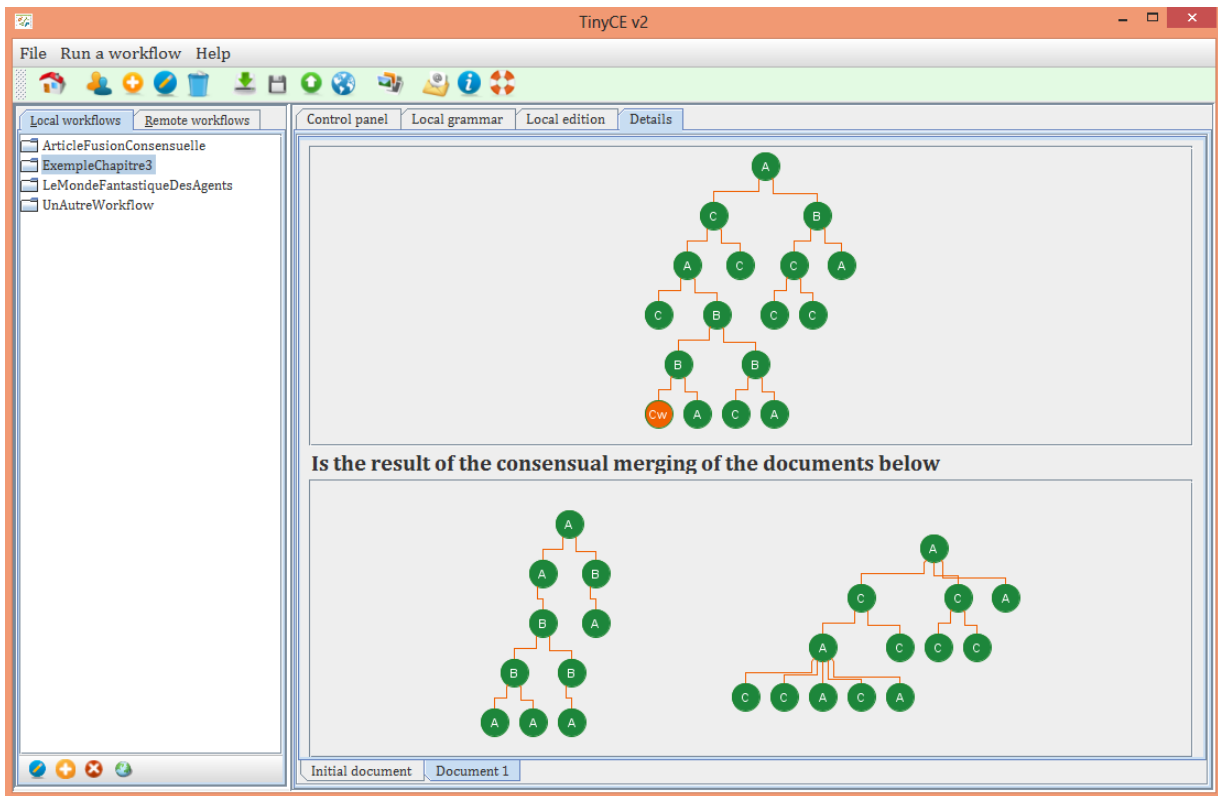


Figure 32 – An illustration of consensual merging in TinyCE v2.

interface of TinyCE v2 (runtime interface) capable of launching a Haskell interpreter (GHCi - Glasgow Haskell Compiler interactive²² - in this case) and make it execute various commands. When creating a workflow, TinyCE v2 generates

22. Official website of GHC: <http://www.haskell.org/ghc/>, visited the 04/04/2020.

a Haskell program file (.hs), containing data types and functions necessary to achieve the operations of projection, expansion and consensual merging on the structured document representing the state of that workflow. In this way, we considerably reduce the use frequency of parsers presented in (Tchoupé Tchendji, 2010). The functions are more open to changes as they are contained in a text file and not in a compiled program as in (Tchoupé Tchendji, 2010). In fact, the main differences between our Java-Haskell cross-fertilisation approach and the one of (Tchoupé Tchendji, 2010) are almost the same that drive the debates on interpreted and compiled languages; our approach is likened to interpreted languages and that of (Tchoupé Tchendji, 2010), to compiled languages. So, even though our approach can present security risks (that can be addressed using PKI (Public Key Infrastructure) and standard encryption systems like AES (Advanced Encryption Standard), RSA (Rivest Shamir Adleman), etc.), it has the advantage of being portable and easier to maintain.

II.5. Summary

This chapter was devoted to the study of asynchronous cooperative editing concepts in general, and to the study of notions related to the model introduced by Badouel and Tchoupé for structured documents cooperative editing (Badouel and Tchoupé Tchendji, 2008). In order to be more efficient, we have chosen to study these models with new contributions, including: an algorithm for reconciling potentially conflicting partial replicas of a structured document, and a generic architecture for designing workflow systems that can be modelled as structured cooperative editing systems in the sense of Badouel and Tchoupé. The correction of the proposed algorithms has been demonstrated. Implementations of these have been made, notably in TinyCE v2, the cooperative editor prototype implemented in Java and Haskell according to the new architecture proposed in this chapter.

The aim of these studies was to familiarise us with some key mathematical tools, in particular: grammars, views, projection/replication algorithms, fusion/reconciliation algorithms, etc. These mathematical tools are proving to be effective in the implementation of artifact-centric BPM models as demonstrated by Badouel et al. with the AWGAG model (Badouel et al., 2014, 2015). For this purpose, they will form the foundation of the new artifact-centric model for the completely decentralised design and execution of administrative workflows (assimilated to the asynchronous cooperative edition of mobile structured documents) on a SOA that we propose in the next chapter.

III

CHAPTER

A CHOREOGRAPHY-LIKE WORKFLOW DESIGN AND DISTRIBUTED EXECUTION FRAMEWORK BASED ON STRUCTURED MOBILE ARTIFACTS' COOPERATIVE EDITING

CONTENTS

| | |
|---|-----|
| III.1 - Introduction | 90 |
| III.2 - Overview of the Artifact-Centric Model Presented in this Thesis | 92 |
| III.3 - Modelling Artifacts | 96 |
| III.4 - Agent and choreography | 103 |
| III.5 - Illustrating the Choreography on the Peer-Review Process | 123 |
| III.6 - Experimentation | 126 |
| III.7 - Related Works and Discussion | 130 |
| III.8 - Summary | 132 |

III.1. Introduction

As outlined in chapter I, section I.4.1.2, the execution of a given business process according to the artifact-centric approach can be assimilated to the cooperative editing of documents. Indeed, in IBM's work (Nigam and Caswell, 2003), an artifact (also called "*adaptive document*") is considered as a document that conveys all the information concerning a particular case of execution of a given process, from its inception into the system to its termination. In particular, this information provides details on the execution status of the case as well as on its lifecycle

(a representation of the possible evolutions of this status). To do this, during the execution of a given process, the actions carried out by each of the stakeholders (agents) have the effect of updating (*editing*) the artifacts involved in that execution. If the process is cooperative, the artifact representing it will be updated by several agents: it is said to be cooperatively edited (*cooperative editing*).

In this chapter, we propose a new artifact-centric approach to BPM. In this one, artifacts are seen as structured documents (annotated trees) that can be exchanged between the different agents involved in the execution of a given business process particular case (it is in this sense that they are said to be mobile); during their life, they are edited appropriately to make the system converge towards the achievement of one of the considered process's business goals. The approach presented in this chapter is based on the asynchronous structured cooperative editing techniques proposed in the work of Badouel et al. (Badouel and Tchoupé Tchendji, 2008; Tchoupé Tchendji, 2009; Tchoupé Tchendji et al., 2017) and extended in chapter II (Tchoupé Tchendji and Zekeng Ndadji, 2016, 2017; Zekeng Ndadji and Tchoupé Tchendji, 2018) of this manuscript.

The major contributions of this chapter are as follows:

1. The proposal of another tree-based model of "business artifact", which makes it possible to better assimilate them to structured documents edited cooperatively;
2. The proposal of a choreography-oriented artifact-centric execution model in which agents execute the same and unique update (editing of artifact upon receipt) and diffusion (dissemination of updates) protocol;
3. The proposal of a prototype of a distributed system allowing to fully experiment the approach investigated in this chapter.

In the rest of this chapter, in section III.2, we present an overview of the studied artifact-centric model and the distributed execution of the peer-review process used as a running example in this manuscript. In section III.3, we introduce and formally define the concepts of artifact and artifact-type (GMWf). We then present in section III.4, the internal structure (architecture and features) of an agent, the notion of accreditation as well as the new artifact-centric and completely decentralised execution model of administrative processes that we propose. Illustrations of section III.4's algorithms are given in section III.5 in order to facilitate their understanding. In the same vein, a prototype system allowing to fully experiment the approach investigated in this chapter is presented in section III.6. In section III.7,

we discuss the obtained results as well as a positioning of these results in relation to those in the literature. The section III.8 is devoted to the conclusion.

III.2. Overview of the Artifact-Centric Model Presented in this Thesis

In this section, a brief description of the artifact-centric model studied in this chapter is given. Furthermore, an overview of the distributed execution of the peer-review process using this model is presented.

III.2.1. A Description of the Artifact-Centric Model Presented in this Thesis

We outlined in this chapter's introduction that the presented artifact-centric model is based on the asynchronous structured cooperative editing techniques proposed in the work of Badouel et al. As in these works, an artifact is represented by a tree containing "*open nodes*" on some of its leaves, materialising the tasks to be executed or being executed and, an attributed grammar called the *Grammatical Model of Workflow* (GMWf) is used as *artifact type*. The symbols of a given GMWf represent the process tasks and each of its productions represents a scheduling of a subset of these tasks; intuitively, a production given by its left and right hand sides, specifies how the task on the left hand side precedes (must be executed before) those on the right hand side (see sec. III.3.1). When a task is executed on a given site, the corresponding open node in the artifact is closed accordingly (it is said to be *closed*) and the data produced during execution are filled in its attributes; then, one of the GMWf's production having the considered task as left hand side is chosen by the local actor to expand the open node into a subtree highlighting in the form of new open nodes, the new tasks to be executed: this is what editing an artifact consists of.

We are especially interested on administrative processes and the approach we propose for their automation is declined in two steps: derivation of different models (target artifacts and their model, accreditations, etc.) from a textual description of the process and, implementation of a choreography between the agents communicating by asynchronous exchange of artifacts for its execution. More precisely, from the observation that one can analyse the textual description of an administrative business process to exhibit all the possible execution scenarios leading to its business goals, we propose to model each of these scenarios by an annotated tree in which, each node corresponds to a task of the process assigned to

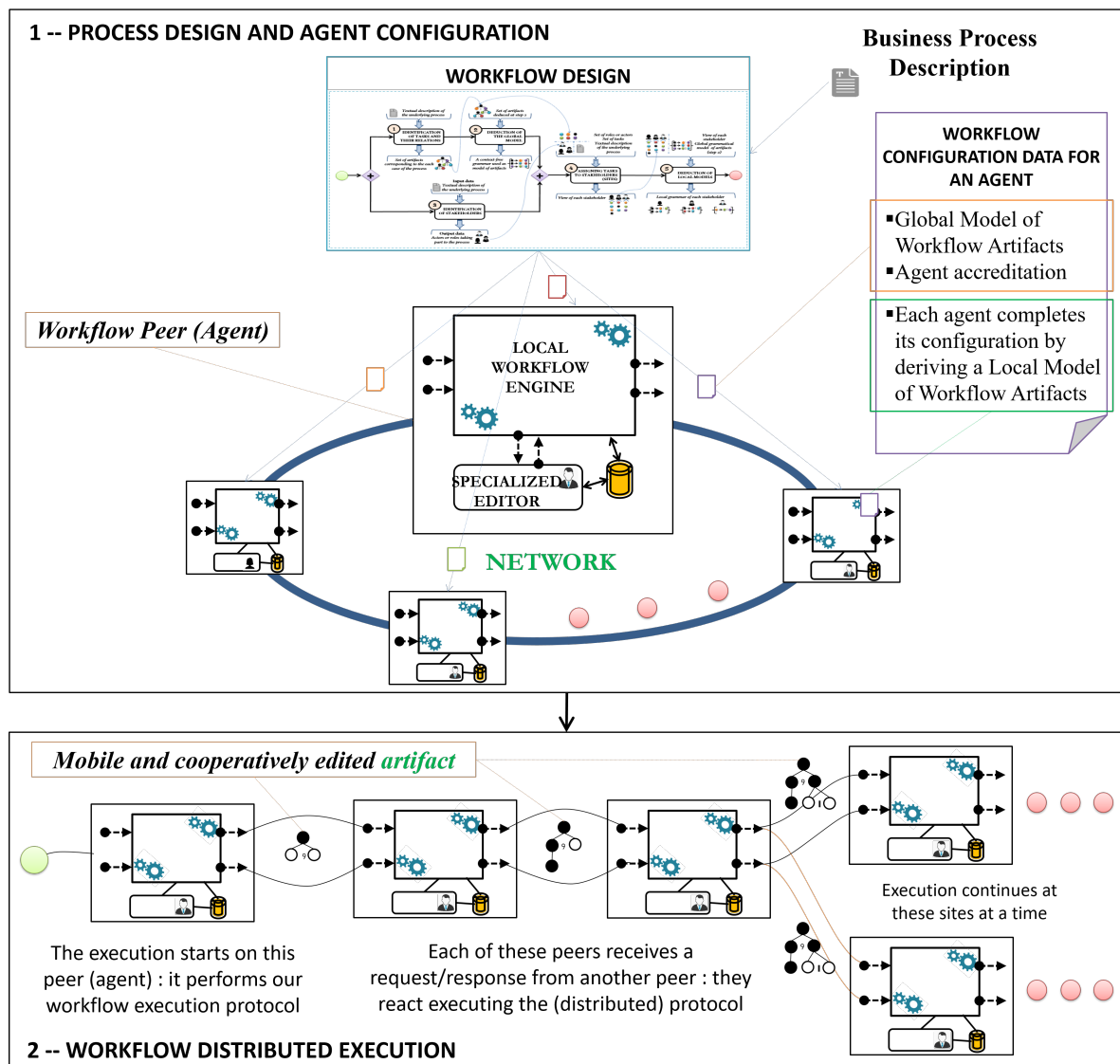


Figure 33 – An overview of the artifact-centric BPM model presented in this chapter.

a given agent, and each hierarchical decomposition (a node and its sons) represents a scheduling of these tasks: these annotated trees are called *target artifacts*. From these target artifacts, are derived a GMWf (*artifact type*) which contains both the *information model* (modelled by its attributes) and the *lifecycle model* (thanks to the set of its productions) which are two essential notions of the artifact-centric modelling paradigm (Hull et al., 2013). Once the GMWf is obtained, we propose to add organisational information called *accreditations* in this chapter; they aim, as in (Badouel and Tchoupé Tchendji, 2008; Tchoupé Tchendji, 2009; Tchoupé Tchendji et al., 2017; Tchoupé Tchendji and Zekeng Ndadji, 2016, 2017; Zekeng Ndadji and Tchoupé Tchendji, 2018), is to enrich the notion of access to different parts of artifacts, by offering a simple mechanism for modelling the gen-

erally different perceptions that actors have on processes and their data. With the couple (GMWf, accreditations), each autonomous agent is configured (see fig. 33 (1)) and is ready to proceed to the decentralised execution of the studied process.

The actual execution is a choreography in which the agents are reactive autonomous software components, communicating in P2P mode and are driven by human agents (actors) in charge of executing tasks. An agent's reaction to the reception of a message (an artifact) consists in the execution of a five-step protocol clearly described in this chapter (see sec. III.4.4.3). This protocol allows it to: (1) *merge* the received artifact with the one it hosts locally in order to consider all updates, (2) *project* the artifact resulting from the merger in order to hide the parts to which the local actor may not have access and highlight the tasks to be locally executed, (3) make the local actor *execute* the revealed tasks and thus edit the potentially partial replica of the artifact obtained after the projection, (4) integrate the new updates into the artifact through an operation called *expansion-pruning* and finally, (5) *diffuse* the updated artifact to other sites for further execution of the process if necessary. The agents' operational capabilities allow that, for the execution of a given process, an artifact created by one of them (initially reduced to an open node), moves from site to site to indicate tasks that are ready to be executed at the appropriate time and to provide necessary data (created by other agents) for that execution; the mobile artifact, cooperatively edited by agents, thus "grows" as it transits through the distributed system so formed (see fig. 33 (2)).

III.2.2. The Running Example: the Peer-Review Process

III.2.2.1. Description of the Peer-Review Process

The peer-review process (Rowland, 2002) is a common example of administrative business process. We presented a brief description of it inspired by those made in (Badouel et al., 2014; Rowland, 2002; Van Der Aalst et al., 2001), in chapter I, section I.2.1.2. Described in this way, we will use the peer-review process as an illustrative example in this chapter.

Lets recall that from that description, we have identified all the tasks to be executed, their sequencing, actors involved and the tasks assigned to them. Precisely, four actors are involved: an editor in chief (*EC*) who is responsible for initiating the process, an associated editor (*AE*) and two referees (*R1* and *R2*). A summary of tasks assignment was presented in table I (page 19), and orchestration diagrams using BPMN and WF-Net were also presented in figure 6 (page 26).

III.2.2.2. Overview of the Peer-Review Process Artifact-Centric Execution using the Model Presented in this Thesis

To run the peer-review process described above according to the artifact-centric model presented in this chapter, four agents controlled by four actors (*the editor in chief, the associated editor* and the *two referees*) will be deployed. Each of them will be pre-configured using a global Grammatical Model of Workflow (GMWf) and a set of accreditations. As we will see later, the global GMWf is used as model of artifacts and formally describes all the process tasks to be executed as well as their execution order (see fig. 6), and the accreditations set specifies the permissions (*reading, writing* and *execution*) of each of the four actors relative to these tasks. After the pre-configuration of agents, each of them will derive (by projection (Tchoupé Tchendji et al., 2017)) a local GMWf which will locally guide the execution of the tasks to guarantee the confidentiality of some workflow data (contained in a mobile artifact) and the consistency of local updates with the global GMWf.

The artifact-centric execution of a scientific paper validation workflow will be triggered on the editor in chief's site, by introducing (in this site) an artifact (an annotated tree) reduced to its root node. Each node of the artifact represents a task and encapsulates an attribute containing its execution status. Therefore at a given time, the whole artifact contains information on already executed tasks and on data produced during their execution; it also exhibits tasks that are ready to be executed. The analysis of this artifact by the local agent will highlight the expected contributions from the editor in chief. Guided by the local GMWf, he (here tasks are executed by a human) will perform tasks resulting in the consistent updating of the artifact's local copy; meaning that new nodes will be added to the artifact and some of its existing nodes will be updated: this is what we call editing an artifact. Then, this (updated) copy will be immediately analysed by the local agent to determine whether the currently managed process scenario is complete (this is the case when the artifact local copy structure matches one of the target artifacts: all causally dependent tasks have been executed) or not: in this case, all sites on which execution must continue are determined and an execution request is addressed to each of them (the artifact is sent to them). Figure 34 sketches an overview of exchanges that can take place between the four agents of the peer-review process when validating a scientific paper using the model presented in the present chapter. The scenario presented there, corresponds to the one in which the paper is pre-validated by the editor in chief and therefore, is analysed by a peer review committee. Note that there may be situations where multiple copies of the

artifact are updated in parallel; this is notably the case when they are present on site 3 (first referee) and 4 (second referee).

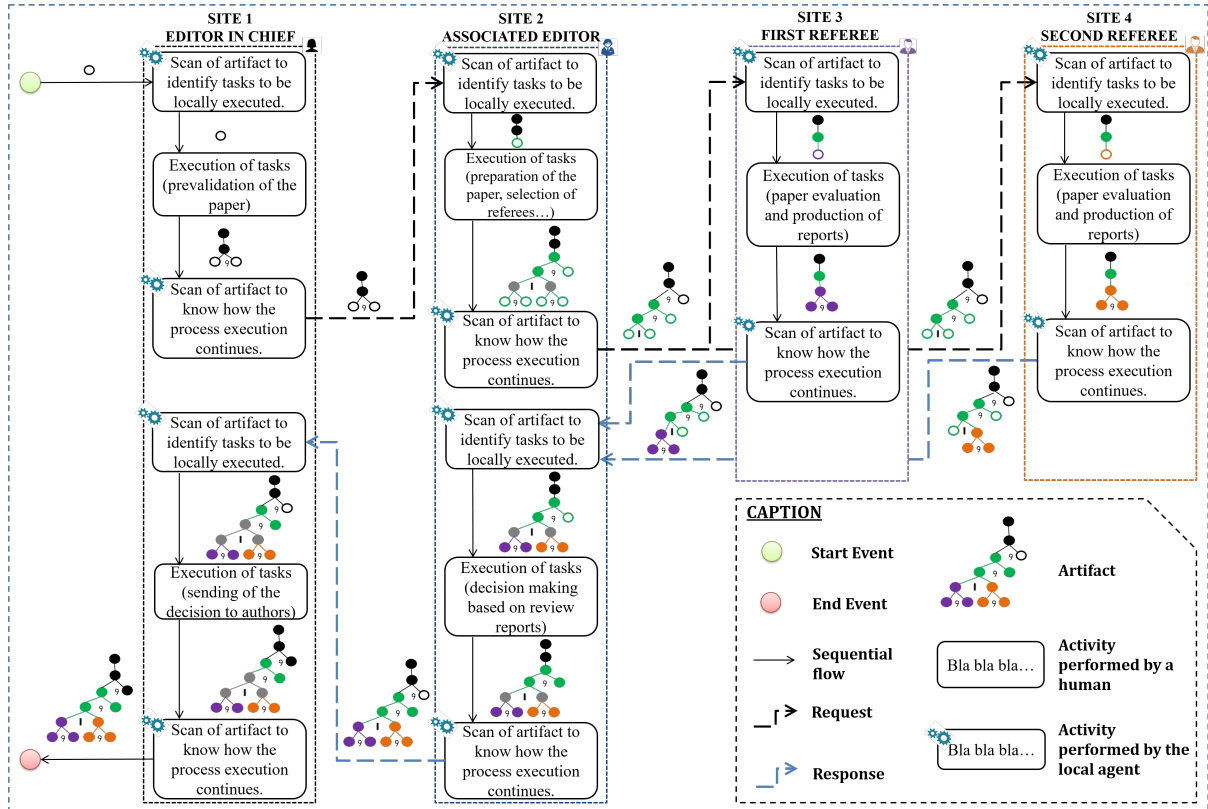


Figure 34 – An overview of the artifact-centric execution of the peer-review process using the model presented in this chapter.

III.3. Modelling Artifacts

III.3.1. Artifacts' Structure

Let's consider an administrative process \mathcal{P}_{op} to be automated. The set $\{\mathcal{S}_{op}^1, \dots, \mathcal{S}_{op}^k\}$ of \mathcal{P}_{op} execution scenarios is known in advance and so, \mathcal{P}_{op} can be specified as any oriented graph with tools like BPMN or as a petri net with tools like YAWL. Moreover, each execution scenario of \mathcal{P}_{op} can be modelled using an annotated tree t_i . Indeed, starting from the fact that a given scenario \mathcal{S}_{op}^i consists of a set $\mathbb{T}_n = \{X_1, \dots, X_n\}$ of n (non-recursive) tasks to be executed in a specific order (in parallel or in sequence), one can represent \mathcal{S}_{op}^i as a tree t_i in which each node (labelled X_i) potentially corresponds to a task X_i of \mathcal{S}_{op}^i , and each hierarchical decomposition (a node and its sons) corresponds to a scheduling: the task associated with the parent node must be executed before those associated with the son nodes; the

latter must be executed according to an order - parallel or sequential - that can be specified by particular annotations. Indeed, it is enough to have two annotations "§" (is sequential to) and "||" (is parallel to) to be applied to each hierarchical decomposition. The annotation "§" (resp. "||") reflects the fact that the tasks associated with the son nodes of the decomposition must (resp. can) be executed in sequence (resp. in parallel).

Considering the running example (the peer-review process), the two scenarios that make it up can be modelled using the two annotated trees in figure 35. In particular, we can see that the tree art_1 shows how the task "Receipt and pre-validation of a submitted paper" assigned to the editor in chief (EC), and associated with the symbol A (see table I, page 19), must be executed before tasks associated with the symbols B and D , that are to be executed in sequence. This annotated tree represents the scenario where the paper received by the editor in chief, is immediately rejected.

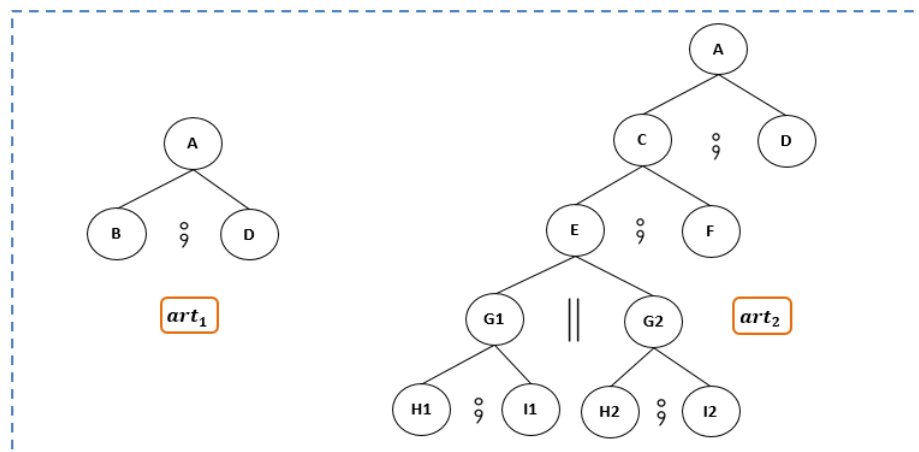


Figure 35 – Target artifacts of a peer-review process.

III.3.2. Target Artifacts and Grammatical Model of Workflow

In this chapter, we use the expression *target artifact* to designate the annotated tree t_i modelling a given scenario S_{op}^i of a given administrative process \mathcal{P}_{op} . From the set of target artifacts of a given process, it is possible to derive an abstract grammar¹ that can be enriched to serve as a *artifact type* as defined in (Hull et al., 2009): it is this grammar that we designate by the expression *Grammatical Model of Workflow (GMWf)*.

1. It is enough to consider the set of target artifacts as a regular tree language: there is therefore a (abstract) grammar to generate them.

Let's consider the set $\{t_1, \dots, t_k\}$ of target artifacts modelling the k execution scenarios of a given process \mathcal{P}_{op} of n tasks ($\mathbb{T}_n = \{X_1, \dots, X_n\}$). Each t_i is a derivation tree for an abstract grammar (a GMWf) $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ whose set of symbols is $\mathcal{S} = \mathbb{T}_n$ (all process tasks) and each production $p \in \mathcal{P}$ reflects a hierarchical decomposition contained in at least one of the target artifacts. Each production is therefore exclusively of one of the following two forms: $p : X_0 \rightarrow X_1 \circlearrowright \dots \circlearrowright X_n$ or $p : X_0 \rightarrow X_1 \parallel \dots \parallel X_n$. The first form $p : X_0 \rightarrow X_1 \circlearrowright \dots \circlearrowright X_n$ (resp. the second form $p : X_0 \rightarrow X_1 \parallel \dots \parallel X_n$) means that task X_0 must be executed before tasks $\{X_1, \dots, X_n\}$, and these must be (resp. these can be) executed in sequence (resp. in parallel). A GMWf can therefore be formally defined as follows:

Definition 10 A *Grammatical Model of Workflow (GMWf)* is defined by $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ where:

- \mathcal{S} is a finite set of **grammatical symbols** or **sorts** corresponding to various **tasks** to be executed in the studied business process;
- $\mathcal{A} \subseteq \mathcal{S}$ is a finite set of particular symbols called **axioms**, representing tasks that can start an execution scenario (roots of target artifacts), and
- $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{S}^*$ is a finite set of **productions** decorated by the annotations " \circlearrowright " (is sequential to) and " \parallel " (is parallel to): they are **precedence rules**. A production $P = (X_{P(0)}, X_{P(1)}, \dots, X_{P(|P|)})$ is either of the form $P : X_0 \rightarrow X_1 \circlearrowright \dots \circlearrowright X_{|P|}$, or of the form $P : X_0 \rightarrow X_1 \parallel \dots \parallel X_{|P|}$ and $|P|$ designates the length of P 's right-hand side. A production with the symbol X as left-hand side is called a *X-production*.

Let's illustrate the notion of GMWf by considering the one generated from an analysis of the target artifacts obtained in the case of the peer-review process (see fig. 35). The derived GMWf is $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ in which, the set \mathcal{S} of grammatical symbols is $\mathcal{S} = \{A, B, C, D, E, F, G1, G2, H1, H2, I1, I2\}$ (see table I); the only initial task (axiom) is A (then $\mathcal{A} = \{A\}$) and the set \mathcal{P} of productions is:

$$\begin{array}{l}
 P_1 : A \rightarrow B \circlearrowright D \\
 P_5 : G1 \rightarrow H1 \circlearrowright I1 \\
 P_9 : F \rightarrow \varepsilon \\
 P_{13} : I2 \rightarrow \varepsilon
 \end{array}
 \left|
 \begin{array}{l}
 P_2 : A \rightarrow C \circlearrowright D \\
 P_6 : G2 \rightarrow H2 \circlearrowright I2 \\
 P_{10} : H1 \rightarrow \varepsilon
 \end{array}
 \right|
 \begin{array}{l}
 P_3 : C \rightarrow E \circlearrowright F \\
 P_7 : B \rightarrow \varepsilon \\
 P_{11} : I1 \rightarrow \varepsilon
 \end{array}
 \left|
 \begin{array}{l}
 P_4 : E \rightarrow G1 \parallel G2 \\
 P_8 : D \rightarrow \varepsilon \\
 P_{12} : H2 \rightarrow \varepsilon
 \end{array}
 \right.$$

For some administrative business processes, there may be special cases where it is not possible to strictly schedule the tasks of a scenario using the two (only) forms of productions selected for GMWf. For example, this is the case for the scenario of

a four-task process with tasks A, B, C and D , where the task A precedes all others, the tasks B and C can be executed in parallel and precede D . In these cases, the introduction of a certain number of new symbols known as *(re)structuring symbols* (not associated with tasks) can make it possible to produce a correct scheduling that respects the form imposed on productions. For the previous example, the introduction of a new symbol S allows us to obtain the following productions: $p_1 : A \rightarrow S; D$, $p_2 : S \rightarrow B \parallel C$, $p_3 : B \rightarrow \epsilon$, $p_4 : C \rightarrow \epsilon$ and $p_5 : D \rightarrow \epsilon$, which properly model the required scheduling. To deal with such cases, the previously given GMWf definition (definition 10) is slightly adapted by integrating the (re)structuring symbols; the resulting definition is as follows:

Definition 11 A *Grammatical Model of Workflow (GMWf)* is defined by $\mathbb{G} = (S, \mathcal{P}, \mathcal{A})$ wherein, \mathcal{P} and \mathcal{A} refer to the same purpose as in definition 10, $S = \mathcal{T} \cup \mathcal{T}_{Struc}$ is a finite set of **grammatical symbols or sorts** in which, those of \mathcal{T} correspond to **tasks** of the studied business process, while those of \mathcal{T}_{Struc} are *(re)structuring symbols*.

III.3.3. Artifact Type and Artifact Edition

As formalised in definition 11, a GMWf perfectly models the tasks and control flow of administrative processes (lifecycle model). To remain faithful to the artifact-centric philosophy, the GMWf definition must be adjusted to be able to use it as an artifact type. In particular, it is necessary to equip it with tools allowing to represent the information model (the data) of processes as well as the dynamic (evolutionary) character of artifacts.

III.3.3.1. Modelling the Information Model of Processes with GMWf

The structure of the consumed and produced data by business processes differs from one process to another. It is therefore not easy to model them using a general type, although several techniques to do so have emerged in recent years (Badouel et al., 2014). For the work presented in this chapter, tackling the data structure of automated processes has no proven interest because, it does not bring any added value to the proposed model: a representation of these data using a set of variables is largely sufficient.

To represent the potential consumed and produced data by the tasks of a process modelled using GMWf, we use the notion of *attribute* embedded in the nodes associated with tasks. To take them into account, we adjust for the last time, the definition of GMWf. We thus attach to each symbol, an attribute named *status*, allowing to store all the data of the associated task; its precise type is left to the

discretion of the process designer. However, for the purposes of this work, we will consider it a string. The new definition of GMWf is thus the following one:

Definition 12 A *Grammatical Model of Workflow (GMWf)* is defined by $\mathbb{G} = (S, \mathcal{P}, \mathcal{A})$ wherein, S , \mathcal{P} and \mathcal{A} refer to the same purpose as in definition 11. Each grammatical symbol $X \in S$ is associated with an attribute named *status* of type string, that can be updated when task X is executed; $X.status$ provides access (read and write) to its content.

A GMWf is therefore ultimately an attributed grammar whose instances represent the different execution scenarios of the underlying business process. In artifact-centric models, the artifact used as a communication medium between agents executing the tasks, must represent at each moment, the execution state of the underlying process. As defined up to now, GMWf models do not satisfy this second concern: they cannot therefore be used as artifact types. We will now equip them with tools to allow them to endow their instances (the artifacts) with the ability to report about the execution state of the process they represent.

III.3.3.2. Artifact Type

For each task, it is important to know whether or not it has already been executed; if not, it is also important to know whether or not it is ready to be executed. Recall also that, we model the execution of processes as the desynchronised cooperative editing of mobile artifacts (which are exchanged by agents). This implies that the artifact-centric model of this chapter considers that, an artifact is a structured document that is initially empty, and which is completed as it circulates between the agents. Contrary to the models in the literature, at each moment, the artifact thus contains only a (potentially empty) part of the lifecycle model of the process. This is why we have chosen not to represent it as a (tree) state machine but rather as an annotated tree that is incrementally built in accordance with an attributed grammar.

Concretely, an *artifact* is an annotated tree that potentially contains buds (this is the equivalent of the notion of structured document being edited as presented in chapter II). A *bud* or *open node* is a typed leaf node indicating in an artifact, a place where an edition is possible; i.e. a node associated with a task that has not yet been executed. A bud can be unlocked (*unlocked bud*) or locked (*locked bud*) depending on whether the task associated with it is ready to be executed² or not.

2. A task is ready to be executed if all the tasks that precede it according to the precedence constraint set have already been executed and the agent that currently holds the mobile artifact, have the necessary accreditation to trigger its execution.

More formally, a *bud of type* $X \in \mathcal{S}$ is a leaf node labelled either by $X_{\bar{\omega}}$ or by X_{ω} depending on its state (*locked* or *unlocked*). An artifact containing no buds is said to be *closed*. Such an artifact, symbolises the end of tasks execution with respect to the agent hosting the artifact. An example of an artifact related to the peer-review process and containing buds is shown in figure 36. In this one, we can see that the tasks associated with symbols A and C have already been executed. Task E is ready to be executed while tasks F and D are not ready to be executed yet.

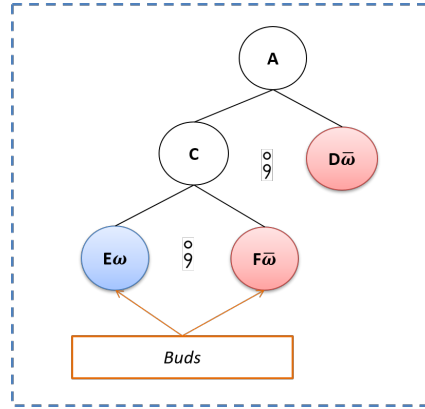


Figure 36 – An intentional representation of an annotated artifact containing buds.

From the thus given definition of (mobile) artifact, it is clear that an artifact is updated only at the level of its leaves and therefore, it only "grows" (positive editing). Knowing moreover that, the correct and complete execution of a given administrative process corresponds to the execution of one of its scenarios, we deduce that: for a process \mathcal{P}_{op} whose GMWf is $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, a given mobile artifact, is a prefix to one of its target artifacts. Thus, the type (model) of this artifact is a grammar $\mathbb{G}_{\Omega} = (\mathcal{S} \cup \mathcal{S}_{\omega}, \mathcal{P} \cup \mathcal{S}_{\Omega}, \mathcal{A} \cup \mathcal{A}_{\omega})$ obtained by extending \mathbb{G} (for bud recognition and recognition of all possible prefixes of target artifacts) as follows:

1. For all sort X , add in the set \mathcal{S} of sorts, two new sorts $X_{\bar{\omega}}$ and X_{ω} ;
2. For all new sort X_{ω} added to \mathcal{S} , add in the set \mathcal{P} of productions two new ε -productions $X_{\Omega} : X_{\omega} \rightarrow \varepsilon$ and $X_{\bar{\Omega}} : X_{\bar{\omega}} \rightarrow \varepsilon$; we then have: $\mathcal{S}_{\omega} = \{X_{\bar{\omega}}, X_{\omega}, X \in \mathcal{S}\}$, $\mathcal{A}_{\omega} = \{X_{\bar{\omega}}, X_{\omega}, X \in \mathcal{A}\}$ and $\mathcal{S}_{\Omega} = \{X_{\Omega} : X_{\omega} \rightarrow \varepsilon, X_{\bar{\Omega}} : X_{\bar{\omega}} \rightarrow \varepsilon, X_{\bar{\omega}} \text{ and } X_{\omega} \in \mathcal{S}_{\omega}\}$.

III.3.3.3. Artifact Edition

If we still consider a running process \mathcal{P}_{op} whose GMWf is $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, then, the *editing* of an artifact t circulating between agents consists of developing one or more of its buds into a subtree while updating their *status* attributes. Concretely, for a bud X_{ω} of the said artifact one can:

1. Execute the task associated with X ;
2. Choose an X -production $P \in \mathcal{P}$ to be used for the development of X_ω ;
3. If P is of the form $P : X \rightarrow X_1 \parallel X_2 \parallel \dots \parallel X_{|P|}$ (resp. $P : X \rightarrow X_1 \circlearrowleft X_2 \circlearrowleft \dots \circlearrowleft X_{|P|}$) then, create $|P|$ buds $X_{1\omega}, X_{2\omega}, \dots, X_{|P|\omega}$ (resp. $X_{1\omega}, X_{2\bar{\omega}}, \dots, X_{|P|\bar{\omega}}$) respectively of type $X_1, X_2, \dots, X_{|P|}$, and replace in the artifact, the considered bud X_ω by the parallel (resp. sequential) subtree $X[X_{1\omega}, X_{2\omega}, \dots, X_{|P|\omega}]^3$ (resp. $X[X_{1\omega}, X_{2\bar{\omega}}, \dots, X_{|P|\bar{\omega}}]$);
4. Update the execution status of the task associated with X : $X.status = \text{''bla bla ...''}$. Updating t results in an artifact t^{maj} and we note $t \leq t^{maj}$.

Although it is obvious, it seems important to clarify that the editing of an artifact is only a consequence of process tasks' execution by actors located on agents. We can therefore imagine this scenario for the peer-review process (see fig. 37): the associated editor who received a request from the editor in chief to peer-review a given article, has executed task C (i.e. he has appraised the paper and formatted it to prepare the peer-review). Through a dedicated tool (a specialised editor), he has been invited to submit a report on the execution of the said task (via the filling of a form for example). The submission of that report, in which he may have provided a copy of the formatted paper as well as comments for referees, will cause (in background) the update of the artifact. The retrieved data will thus be stored in the *status* attribute of task C and the bud C_ω will be extended into a subtree as described above (the only production available for this purpose is $P_3 : C \rightarrow E \circlearrowleft F$).

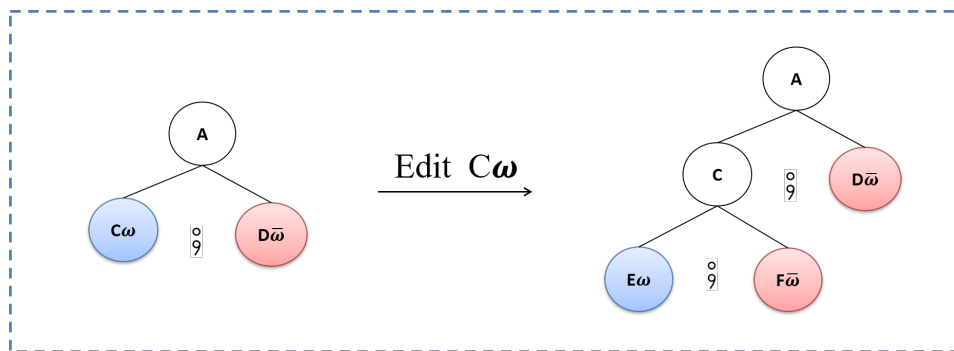


Figure 37 – An example of artifact edition: the bud C_ω is extended in a subtree.

3. The tree coded by $X[X_{1\omega}, X_{2\omega}, \dots, X_{|P|\omega}]$ is the one whose root is labelled X and has as sons, $|P|$ nodes labelled by $X_{1\omega}, X_{2\omega}, \dots, X_{|P|\omega}$ respectively.

III.4. Agent and choreography

Now that we have formally defined the structure and the editing model of artifacts, let's focus on the structure of agents that oversee the execution of tasks and update the artifact accordingly, as well as on the artifact-centric choreography implemented between them.

III.4.1. Relations between Agent, Actor and Choreography

We borrowed the term *agent* from (Lohmann and Wolf, 2010). In the case of our study, an *agent* (which we also call a *peer*) is a software component, installed at a given site, piloted by a human agent called *actor* (the tasks of the processes we handle are executed by humans) and capable of interacting with other agents by service invocation (message exchange). An agent is completely *autonomous*: i.e. it encapsulates all the data and functions necessary for the execution of the tasks assigned to it, or precisely, tasks assigned to the actor piloting it. The agent is *reactive*: it reacts in the same way to each message it receives by executing a well-defined protocol that goes from the analysis of the received message (artifact) to the possible transmission of other messages. As announced in the introduction, each message contains a collectively edited *mobile artifact*. For the execution of a given process, the choreography is therefore a result of the messages (artifact replicas) exchanges between the agents involved and of the reaction of the latter to the reception of messages.

III.4.2. Structure of an Agent

An agent is built to be able to fully manage the lifecycle (creation, storage, edition/execution) of a given business process' artifacts. Thus, an agent is made up of three major software components: a local workflow engine (LWfE), a specialised graphical editor and a storage device (see fig. 38).

III.4.2.1. The Local Workflow Engine

The local workflow engine (LWfE) is the main component of an agent. It receives messages from other agents and reacts by executing a well defined protocol (see sec. III.4.4.3). It communicates with the other agent's engines via its communication interface that exposes four services : two input services or *provided services* (*returnTo* and *forwardTo*) connected to two corresponding output services or *required services* (*returnTo* and *forwardTo*) so that:

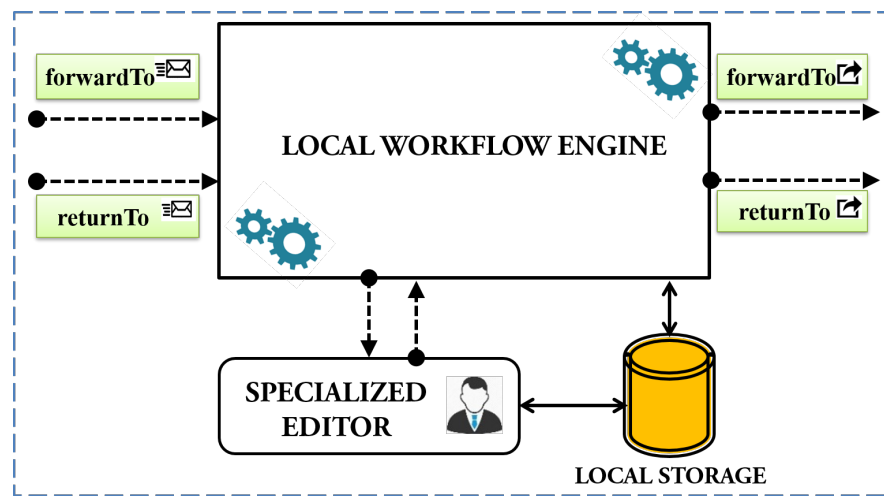


Figure 38 – Simplified architecture of an agent.

- The invocation by an agent j of the service *forwardTo* offered by an agent i , causes on i , the execution of its corresponding input service *forwardTo*. This service makes it possible to send a **request** from agent j to agent i . The request contains the replica of the mobile artifact located on agent j . This artifact must contain buds to be completed (executed) by actor A_i (the human agent piloting agent i).
- The invocation by an agent i of the service *returnTo* offered by an agent j , causes the execution on j , of its corresponding input service *returnTo*. This service allows agent i to return the **response** to a request previously received from agent j . As the request, the response contains the replica of the mobile artifact located on agent i .

III.4.2.2. The Storage Device

A database (DB) of documents (a JSON⁴ DB for example) is used by the LWfE to store an agent's configuration and data (especially artifacts) that it handles.

III.4.2.3. The Specialised Editor

Each agent provides a specialised editor (preferably WYSIWYG⁵) that allows its actor (the pilot) to execute tasks. More precisely, the specialised editor allows the actor to view the tasks that are assigned to him, those ready to be executed, and when he has executed a task, it gives him the means to record an execution report. Any (editing) action carried out by the local actor via the specialised editor, causes

4. JavaScript Object Notation, <http://www.json.org>, <https://www.mongodb.com>, visited the 04/04/2020.

5. What You See Is What You Get.

the consistent update (as presented in section III.3.3.3) of the mobile artifact local replica.

The specialised editor is particularly important as it guarantees controlled access to the artifact. Indeed, as announced in the introduction and following the steps of (Hull et al., 2009), for reasons of confidentiality/security, actors do not necessarily have the right to access all information relating to the execution of a process in which they are involved. It is therefore important to provide a mechanism for regulating access to this information (stored in the artifact). In our case, we define this mechanism under the name *accreditation* and we include it in the configuration of an agent in the same way as the GMWf of the studied process.

III.4.3. Concepts of Accreditation, Partial Replica of an Artifact and Local GMWf

III.4.3.1. Concept of Accreditation

Let's consider a process \mathcal{P}_{op} and its GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$. The accreditation of an agent provides information on the rights (permissions) its actor has on each sort (task) of \mathbb{G} . To simplify, the nomenclature of rights manipulated here is inspired by the one used in Unix-like operating systems. Three types of accreditation are then defined: accreditation in reading (r), in writing (w) and in execution (x).

1. *Accreditation in reading (r)*: when an agent is accredited in reading on a sort X , its actor has the right to know if the associated task is executed. Moreover, he can access its execution status. We call an agent's (actor's) *view* the set of sorts on which it (he) is accredited in reading.
2. *Accreditation in writing (w)*: when an agent is accredited in writing on a sort X , its actor can execute the associated task. Note that a task can be executed only by exactly one actor: for a given sort, a single agent is accredited in writing; this is an important point of the model which guarantees the absence of execution conflicts. Since the dedicated editors for "updating artifacts" are of type WYSIWYG (see sec. III.4.2.3), any agent accredited in writing on a symbol must therefore be accredited in reading on it.
3. *Accreditation in execution (x)*: an agent accredited in execution on a sort X is authorised to ask the agent which is accredited in writing on it, to execute

it. Note that this request can be made without the agent being accredited in reading⁶ on the considered sort.

More formally, an accreditation is defined as follows:

Definition 13 An *accreditation* \mathcal{A}_{A_i} defined on the set S of grammatical symbols for an agent i piloted by an actor A_i , is a triplet $\mathcal{A}_{A_i} = (\mathcal{A}_{A_i(r)}, \mathcal{A}_{A_i(w)}, \mathcal{A}_{A_i(x)})$ such that, $\mathcal{A}_{A_i(r)} \subseteq S$ also called *view* of i (or *view* of A_i), is the set of symbols on which i is accredited in reading, $\mathcal{A}_{A_i(w)} \subseteq \mathcal{A}_{A_i(r)}$ is the set of symbols on which i is accredited in writing and $\mathcal{A}_{A_i(x)} \subseteq S$ is the set of symbols on which i is accredited in execution.

The accreditations of various agents must be produced by the workflow designer just after modelling the scenarios in the form of target artifacts. From the task assignment for the peer-review process in the running example (see table I), it follows that the accreditation in writing of the editor in chief is $\mathcal{A}_{EC(w)} = \{A, B, D\}$, that of the associated editor is $\mathcal{A}_{AE(w)} = \{C, E, F\}$ and that of the first (resp. the second) referee is $\mathcal{A}_{R_1(w)} = \{G1, H1, I1\}$ (resp. $\mathcal{A}_{R_2(w)} = \{G2, H2, I2\}$). Even more, since the editor in chief can only perform the task D if the task C is already executed (see artifacts art_1 and art_2 , fig. 35), in order for the editor in chief to be able to ask the associated editor to perform this task, it (the agent) must be accredited in execution on it; so we have $\mathcal{A}_{EC(x)} = \{C\}$. Moreover, in order to be able to access all the information on the peer-review evaluation of a paper (task C) and to summarise the right decision to send to the author, the editor in chief must be able to consult the reports (tasks $I1$ and $I2$) and the messages (tasks $H1$ and $H2$) of the different referees, as well as the final decision taken by the associated editor (task F). These tasks, added to $\mathcal{A}_{EC(w)}$ ⁷ constitute the set $\mathcal{A}_{EC(r)} = \mathcal{V}_{EC} = \{A, B, C, D, H1, H2, I1, I2, F\}$ of tasks on which, it is accredited in reading. By doing so for each of other agents, we deduce the accreditations represented in table VII.

Table VII – Accreditations of the different agents taking part in the peer-review process.

| Agent | Accreditation |
|------------------------|--|
| Editor in Chief (EC) | $\mathcal{A}_{EC} = (\{A, B, C, D, H1, H2, I1, I2, F\}, \{A, B, D\}, \{C\})$ |
| Associated Editor (AE) | $\mathcal{A}_{AE} = (\{A, C, E, F, H1, H2, I1, I2\}, \{C, E, F\}, \{G1, G2\})$ |
| First referee (R1) | $\mathcal{A}_{R1} = (\{C, G1, H1, I1\}, \{G1, H1, I1\}, \emptyset)$ |
| Second referee (R2) | $\mathcal{A}_{R2} = (\{C, G2, H2, I2\}, \{G2, H2, I2\}, \emptyset)$ |

6. In fact, as we will see later (sec. III.4.4.3 (the diffusion protocol, page 122)), it is an automatic task (of the agent) that sends the execution request and not the actor.

7. Recall that in our case, we use WYSIWYG tools and therefore, one can only execute what he see.

In summary, what is the workflow model ?

To summarise, we state that in the artifact-centric model presented in this chapter, an administrative process \mathcal{P}_{op} is completely specified using a triplet $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ called a *Grammatical Model of Administrative Workflow Process* (GMAWfP) and composed of: a GMWf, a list of actors (agents) and a list of their accreditations. The GMWf is used to describe all the tasks of the studied process and their scheduling, while the list of accreditations provides information on the role played by each actor involved in the process execution. A GMAWfP can then be formally defined as follows:

Definition 14 *A Grammatical Model of Administrative Workflow Process (GMAWfP) \mathbb{W}_f for a given business process, is a triplet $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ wherein \mathbb{G} is the studied process (global) GMWf, \mathcal{L}_{P_k} is the set of k agents taking part in its execution and $\mathcal{L}_{\mathcal{A}_k}$ represents the set of these agents' accreditations.*

III.4.3.2. Concept of Partial Replica of an Artifact

To effectively ensure that actors only have access to information of proven interest to them, each agent let them access only to a potentially partial replica $t_{\mathcal{V}_i}$ of the mobile artifact t . The t 's partial replicas are obtained by projections according to the views of each actor. A partial replica $t_{\mathcal{V}_i}$ of t according to the view $\mathcal{V}_{A_i} = \mathcal{A}_{A_i(r)}$, is a partial copy of t obtained by means of the so-called *projection operator* denoted π as presented below.

Technically, the projection $t_{\mathcal{V}_i}$ of an artifact t according to the view $\mathcal{V}_i = \mathcal{A}_{A_i(r)}$ is obtained by deleting in t all nodes whose types do not belong to \mathcal{V}_i . In our case, the main challenges in this operation are:

- (1) nodes of $t_{\mathcal{V}_i}$ must preserve the previously existing execution order between them in t ,
- (2) $t_{\mathcal{V}_i}$ must be build by using exclusively the only two forms of production retained for GMWf and
- (3) $t_{\mathcal{V}_i}$ must be unique in order to ensure the continuation of process execution (see sec. III.4.4.3).

The projection operation is noted π . Inspired by the one proposed in (Badouel and Tchoupe Tchendji, 2008), it projects an artifact by preserving the hierarchy (father-son relationship) between nodes of the artifact (it thus meets challenge (1)); but in addition, it inserts into the projected artifact when necessary, new additional (*re*)structuring symbols (accessible in reading and writing by the agent for whom

the projection is made). This enables it to meet challenge (2). The details of how to accomplish the challenge (3) are outlined immediately after the algorithm (algorithm 5) is presented.

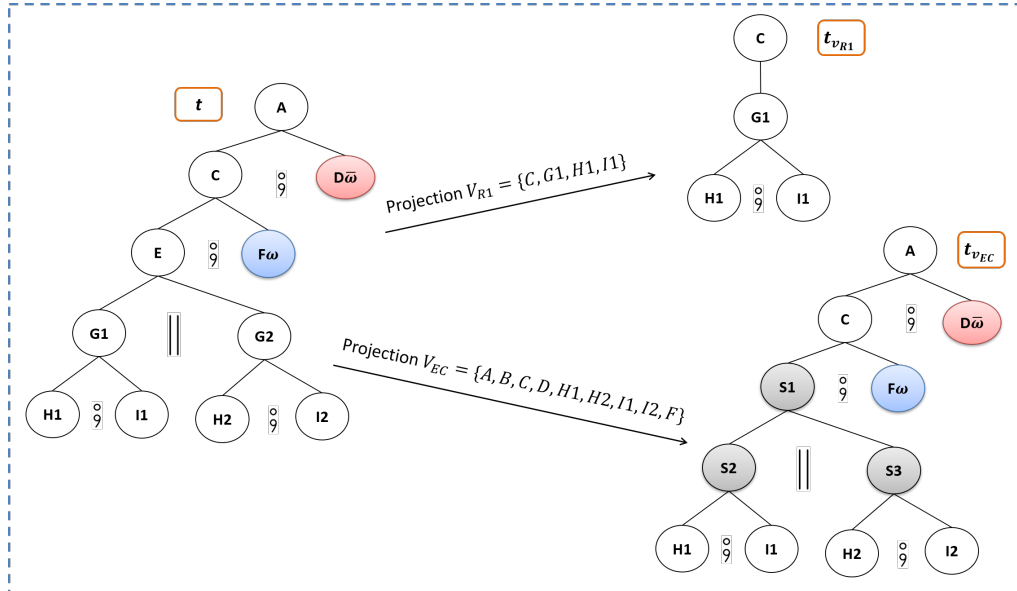


Figure 39 – Example of projections made on an artifact and partial replicas obtained.

Figure 39 illustrates the projection of an artifact of the peer-review process relatively to the $R1$ (first referee) and EC (Editor in Chief) agent views. Note the presence in $t_{V_{EC}}$ of new (re)structuring symbols (in gray). These last ones make it possible to avoid introducing in $t_{V_{EC}}$, the production $p : C \rightarrow H1 \ ; \ I1 \ \parallel \ H2 \ ; \ I2 \ ; \ F$ whose form does not correspond to the two forms of production retained for the GMWf writing⁸.

The algorithm

Let's consider an artifact t and note by $n = X [t_1, \dots, t_m]$ a node of t labelled with the symbol X and having m sub-artifacts t_1, \dots, t_m . Note also by p_n , the production of the GMWf that was used to extend node n ; the type of p_n is either *sequential* (i.e. $p_n : X \rightarrow X_1 \ ; \ \dots \ ; \ X_m$ where X_1, \dots, X_m are the roots of the sub-artifacts t_1, \dots, t_m) or *parallel* ($p_n : X \rightarrow X_1 \ \parallel \ \dots \ \parallel \ X_m$). Concretely, to project t according to a given view \mathcal{V} (i.e to find $projs_t = \pi_{\mathcal{V}}(t)$), the recursive processing presented in algorithm 5, is applied to the root node $n = X [t_1, \dots, t_m]$ of t .

8. Note that this production specifies in its right-hand side that we must have parallel and sequential treatments. Inserting $S1$, $S2$ and $S3$ allows to rewrite p in four productions $p1 : C \rightarrow S1 \ ; \ F$, $p2 : S1 \rightarrow S2 \ \parallel \ S3$, $p3 : S2 \rightarrow H1 \ ; \ I1$ and $p4 : S3 \rightarrow H2 \ ; \ I2$.

Algorithm 5 Algorithm to project a given artifact according to a given view.

- **If symbol X is visible** ($X \in \mathcal{V}$) then :
 1. n is kept in the artifact;
 2. For each sub-artifact t_i of n , having node $n_i = X_i[t_{i_1}, \dots, t_{i_k}]$ as root (of which p_{n_i} is the production that was used to extend it), the following processing is applied :
 - a. The projection of t_i according to \mathcal{V} is done. We obtain the list $projs_{t_i} = \pi_{\mathcal{V}}(t_i) = \{t_{i_{v_1}}, \dots, t_{i_{v_l}}\}$;
 - b. If the type of p_{n_i} is the same as the type of p_n or the projection of t_i has produced no more than one artifact ($|projs_{t_i}| \leq 1$), we just replace t_i by artifacts $t_{i_{v_1}}, \dots, t_{i_{v_l}}$ of the list $projs_{t_i}$;
 - Otherwise, a new (re)structuring symbol S_i is introduced and we replace the sub-artifact t_i with a new artifact new_t_i whose root node is $n_{t_i} = S_i[t_{i_{v_1}}, \dots, t_{i_{v_l}}]$;
 3. If the list of new sub-artifacts of n contains only one element t_1 having $n_1 = S_1[t_{1_{v_1}}, \dots, t_{1_{v_l}}]$ (with S_1 a newly created (re)structuring symbol) as root node, we replace in this one, t_1 by the sub-artifacts $t_{1_{v_1}}, \dots, t_{1_{v_l}}$ of n_1 . This removes a non-important (re)structuring symbol S_1 .
 - **Else**, n is deleted and the result of the projection ($projs_t$) is the union of the projections of each of its sub-artifacts: $projs_t = \pi_{\mathcal{V}}(t) = \bigcup_{i=1}^m \pi_{\mathcal{V}}(t_i)$
-

Note that the algorithm described here can return several artifacts (a forest). To avoid that it produces a forest in some cases and thus meet challenge **(3)**, we make the following assumption:

GMAWfP manipulated in this work are such that all agents are accredited in reading on the GMWf axioms (axioms' visibility assumption).

The designer must therefore ensure that all agents are accredited in reading on all GMWf axioms. To do this, after modelling a process \mathcal{P}_{op} and obtaining its GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, it is sufficient (if necessary) to create a new axiom $A_{\mathbb{G}}$ on which, all actors will be accredited in reading, and to associate it with new unit productions⁹ $pa : A_{\mathbb{G}} \rightarrow X_a$ where, $X_a \in \mathcal{A}$ is a symbol labelling the root of a target artifact. Moreover, the designer of the GMWf must statically choose the agent responsible for initiating the process. This agent will therefore be the only one to possess an accreditation in writing on the new axiom $A_{\mathbb{G}}$.

Proposition 15 *For all GMAWfP $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{A_k})$ verifying the axioms' visibility assumption, the projection of an artifact t which is conform to its GMWf ($t \cdot \mathbb{G}$) according to a given view \mathcal{V} , results in a single artifact $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ (stability property of π).*

9. A production of a context free grammar is a *unit production* if it is on the form $A \rightarrow B$, where A and B are non-terminal symbols.

Proof. Let's show that $\pi_{\mathcal{V}}(t)$ produces a single tree $t_{\mathcal{V}}$ which is an artifact. Note that the only case in which the projection of an artifact t according to a view \mathcal{V} produces a forest, is when the root node of t is associated with an invisible symbol X ($X \notin \mathcal{V}$). Knowing that $t \vdash \mathbb{G}$ and that \mathbb{W}_f validates the axioms' visibility assumption, it is deduced that the root node of t is labelled by one of the axioms $A_{\mathbb{G}}$ of \mathbb{G} and that $A_{\mathbb{G}} \in \mathcal{V}$ (hence the uniqueness of the produced tree). Since the projection operation preserves the form of productions, it is concluded that $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ is an artifact. \square

A Haskell implementation of this projection algorithm is introduced in appendix A of this manuscript. Another implementation in Java has also been proposed and integrated into the prototype that we will present in section III.6 of this chapter.

III.4.3.3. The Need of a Local GMwf

Since the artifact copy manipulated at a specific site is a potentially partial replica of the mobile (global) artifact, and since its editing depends on the agent's perception (view) of the process, it becomes crucial to provide each agent with a local GMwf. The latter will serve in addition to preserve the possible confidentiality of certain tasks and data, to guide the local actions of updating the artifact in order to ensure the convergence of the system to a coherent business goal state. The local GMwf of an agent can be derived by projecting the global GMwf \mathbb{G} according to the view \mathcal{V}_i of its pilot (**GMwf projection**). This projection is carried out using Π operator and the GMwf obtained is noted $\mathbb{G}_{\mathcal{V}_i} = \Pi_{\mathcal{V}_i}(\mathbb{G})$.

A naive algorithm for non-recursive GMwf projection

The goal of this algorithm is to derive by projection of a given GMwf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ according to a view \mathcal{V} , a local GMwf $\mathbb{G}_{\mathcal{V}} = (\mathcal{S}_{\mathcal{V}}, \mathcal{P}_{\mathcal{V}}, \mathcal{A}_{\mathcal{V}})$ (we note $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$). The proposed algorithm is algorithm 6.

Figure 40 illustrates the research of a local model $\mathbb{G}_{\mathcal{V}_{EC}}$ such as $\mathbb{G}_{\mathcal{V}_{EC}} = \Pi_{\mathcal{V}_{EC}}(\mathbb{G})$ with $\mathcal{V}_{EC} = \mathcal{A}_{EC(r)} = \{A, B, C, D, H1, H2, I1, I2, F\}$. Target artifacts generated from \mathbb{G} (fig. 40(b)) are projected to obtain two *local target artifacts for the view \mathcal{V}_{EC}* (fig. 40(c)). From the local target artifacts thus obtained, the searched GMwf is produced (fig. 40(d)).

The GMwf projection algorithm presented here only works for GMwf that do not allow recursive symbols¹⁰. We therefore assume that:

10. It is only in this context that all the target artifacts can be enumerated.

Algorithm 6 Algorithm to project a given GMWf according to a given view.

1. First of all, it is necessary to generate all the target artifacts denoted by \mathbb{G} (see note (1) below); we thus obtain a set $arts_{\mathbb{G}} = \{t_1, \dots, t_n\}$;
2. Then, each of the target artifacts must be projected according to \mathcal{V} . We thus obtain a set $arts_{\mathbb{G}_{\mathcal{V}}} = \{t_{q_1}, \dots, t_{q_m}\}$ (with $m \leq n$ because there may be duplicates; in this case, only one copy is kept) of artifacts partial replicas;
3. Then, collect the different (re)structuring symbols appearing in artifacts of $arts_{\mathbb{G}_{\mathcal{V}}}$, making sure to remove duplicates (see note (2) below) and to consequently update the artifacts and the set $arts_{\mathbb{G}_{\mathcal{V}}}$. We thus obtain a set $\mathcal{S}_{\mathcal{V}_{Struct}}$ of symbols and a final set $arts_{\mathbb{G}_{\mathcal{V}}} = \{t_{q_1}, \dots, t_{q_l}\}$ (with $l \leq m$) of artifacts. These are exactly the only ones that must be conform to the searched GMWf $\mathbb{G}_{\mathcal{V}}$. So we call them, *local target artifacts for the view \mathcal{V}* ;
4. At this stage, it is time to collect all the productions that made it possible to build each of the *local target artifacts for the view \mathcal{V}* . We obtain a set $\mathcal{P}_{\mathcal{V}}$ of distinct productions.

The searched local GMWf $\mathbb{G}_{\mathcal{V}} = (\mathcal{S}_{\mathcal{V}}, \mathcal{P}_{\mathcal{V}}, \mathcal{A}_{\mathcal{V}})$ is such as:

- a. its set of symbols is $\mathcal{S}_{\mathcal{V}} = \mathcal{V} \cup \mathcal{S}_{\mathcal{V}_{Struct}}$;
- b. its set of productions is $\mathcal{P}_{\mathcal{V}}$;
- c. its axioms are in $\mathcal{A}_{\mathcal{V}} = \mathcal{A}$

Note (1): To generate all the target artifacts denoted by a GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$, one just has to use the set of productions to generate the set of artifacts having one of the axiom $A_{\mathbb{G}} \in \mathcal{A}$ as root. In fact, for each axiom $A_{\mathbb{G}}$, it should be considered that every $A_{\mathbb{G}}$ -production $P = (A_{\mathbb{G}}, X_1 \cdots X_n)$ induces artifacts $\{t_1, \dots, t_m\}$ such as: the root node of each t_i is labelled $A_{\mathbb{G}}$ and has as its sons, a set of artifacts $\{t_{i_1}, \dots, t_{i_n}\}$, part of the Cartesian product of the sets of artifacts generated when considering each symbol X_1, \dots, X_n as root node.

Note (2): In this case, two (re)structuring symbols are identical if for all their appearances in nodes of the different artifacts of $arts_{\mathbb{G}_{\mathcal{V}}}$, they induce the same local scheduling.

*For the execution model presented in this chapter, the manipulated GMAWfP are those whose GMWf do not contain recursive symbols (**non-recursive GMWf assumption**).*

Therefore, it is no longer possible to express iterative routing between process tasks (in the general case); except in cases where the maximum number of iterations is known in advance. This algorithm has some interesting properties and the interested reader will find an introduction to its Haskell implementation in appendix A.

Proposition 16 *For all GMAWfP $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{A_k})$ verifying the axioms' visibility and the non-recursive GMWf assumptions, the projection of its GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ according to a given view \mathcal{V} , is a GMWf $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$ for a GMAWfP $\mathbb{W}_{f_{\mathcal{V}}}$ verifying the assumptions of axiom visibility and non-recursive GMWf (stability property of Π).*

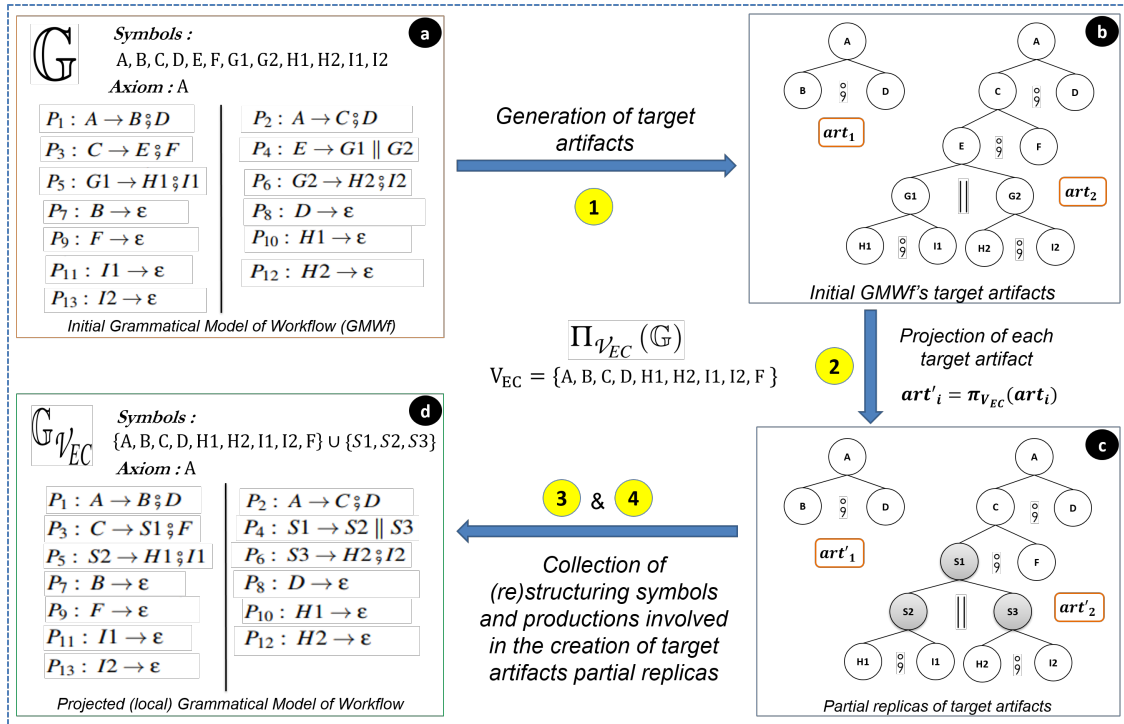


Figure 40 – Example of projection of a GMWf according to a given view.

Proof. Let's show that $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$ is a GMWf for a new GMAWfP $\mathbb{W}_{f_{\mathcal{V}}} = (\mathbb{G}_{\mathcal{V}}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_{\mathcal{V}_k}})$ that verifies the assumptions of axioms' visibility and non-recursivity of GMWf. As $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ validates the non-recursivity of GMWf assumption, the set of target artifacts ($arts_{\mathbb{G}} = \{t_1, \dots, t_n\}$) that it denotes is finite and can therefore be fully enumerated. Knowing further that \mathbb{W}_f validates the axioms' visibility assumption, it is deduced that the set $arts_{\mathbb{G}_{\mathcal{V}}} = \{t_{\mathcal{V}_1} = \pi_{\mathcal{V}}(t_1), \dots, t_{\mathcal{V}_n} = \pi_{\mathcal{V}}(t_n)\}$ is finite and the root node of each artifact $t_{\mathcal{V}_i}$ is associated with an axiom $A_{\mathbb{G}} \in \mathcal{A}$ (see proposition 15). $\mathbb{G}_{\mathcal{V}}$ being built from the set $arts_{\mathbb{G}_{\mathcal{V}}}$, its axioms $\mathcal{A}_{\mathcal{V}} = \mathcal{A}$ are visible to all actors and its productions are only of the two forms retained for GMWf. In addition, each new (re)structuring symbol ($S \in \mathcal{S}_{\mathcal{V}_{Struc}}$) is created and used only once to replace a symbol that is not visible and not recursive (by assumption) when projecting artifacts of $arts_{\mathbb{G}}$. The new symbols are therefore not recursive. By replacing in $\mathcal{L}_{\mathcal{A}_k}$ the view \mathcal{V} by $\mathcal{V} \cup \mathcal{S}_{\mathcal{V}_{Struc}}$, one obtains a new set $\mathcal{L}_{\mathcal{A}_{\mathcal{V}_k}}$ of accreditations for a new GMAWfP $\mathbb{W}_{f_{\mathcal{V}}} = (\mathbb{G}_{\mathcal{V}}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_{\mathcal{V}_k}})$ verifying the assumptions of axioms' visibility and non-recursivity of GMWf. \square

Proposition 17 *For all GMAWfP $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ verifying the axioms' visibility and the non-recursivity of GMWf assumptions, the projection of an artifact t which is conform to the GMWf \mathbb{G} according to a given view \mathcal{V} , is an artifact which is conform to the projection of \mathbb{G} according to \mathcal{V} ($\forall t, t \cdot \mathbb{G} \Rightarrow \pi_{\mathcal{V}}(t) \cdot \Pi_{\mathcal{V}}(\mathbb{G})$).*

Proof. Knowing that the considered GMAWfP $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ verifies the axioms' visibility and the non-recursivity of GMWf assumptions, it is deduced that the set of its target artifacts $arts_{\mathbb{G}}$ (those who helped to build its GMWf \mathbb{G}) is finite and any artifact that is conform to its GMWf \mathbb{G} is a target artifact ($\forall t, t \cdot \mathbb{G} \Leftrightarrow t \in arts_{\mathbb{G}}$). Therefore, considering a given artifact t such that t is conform to \mathbb{G} ($t \cdot \mathbb{G}$), one knows that it is a target artifact ($t \in arts_{\mathbb{G}}$) and its projection according to a given view \mathcal{V} produces a single artifact $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ (see "stability property of π ", proposition 15) such as t and $t_{\mathcal{V}}$ have the same root (one of the axioms $A_{\mathbb{G}} \in \mathcal{A}$ of \mathbb{G}). Since t is a target artifact, its projection $t_{\mathcal{V}}$ (through the renaming of some potential (re)structuring symbols) is part of the set $arts_{\mathbb{G}_{\mathcal{V}}}$ of artifacts that have generated $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$ by applying the projection principle described in the algorithm 6. Therefore, the productions involved in the construction of $t_{\mathcal{V}}$ are all included in the set of productions of the GMWf $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$. As the set of axioms of $\mathbb{G}_{\mathcal{V}}$ is $\mathcal{A}_{\mathcal{V}} = \mathcal{A}$, it is deduced that $A_{\mathbb{G}} \in \mathcal{A}_{\mathcal{V}}$ and concluded that $t_{\mathcal{V}} \cdot \mathbb{G}_{\mathcal{V}}$. \square

Proposition 18 Consider a GMAWfP $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ verifying the axioms' visibility and the non-recursivity assumptions. For all artifact $t_{\mathcal{V}}$ which is conform to $\Pi_{\mathcal{V}}(\mathbb{G})$, it exists at least one artifact t which is conform to \mathbb{G} such that $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ ($\forall t_{\mathcal{V}}, t_{\mathcal{V}} \cdot \Pi_{\mathcal{V}}(\mathbb{G}) \Rightarrow \exists t, t \cdot \mathbb{G}$ and $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$).

Proof. With proposition 16 ("stability property of Π ") it has been shown that the projection $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$ according to the view \mathcal{V} of a GMWf \mathbb{G} verifying the axioms' visibility and the non-recursivity assumptions, is a GMWf verifying the same assumptions. On this basis and using similar reasoning to that used to prove the proposition 17, it's been determined that an artifact $t_{\mathcal{V}}$ that is conform to $\mathbb{G}_{\mathcal{V}}$, is one of its target artifacts (*local target artifact for the view \mathcal{V}*): i.e., $t_{\mathcal{V}} \in arts_{\mathbb{G}_{\mathcal{V}}}$. Referring to the projection process which made it possible to obtain $\mathbb{G}_{\mathcal{V}}$, it is determined that the set $arts_{\mathbb{G}_{\mathcal{V}}}$ is exclusively made up of the projections of the set $arts_{\mathbb{G}} = \{t_1, \dots, t_n\}$ of \mathbb{G} 's target artifacts. $t_{\mathcal{V}}$ is therefore the projection of at least one target artifact $t_i \in arts_{\mathbb{G}}$ of \mathbb{G} ($t_{\mathcal{V}} = \pi_{\mathcal{V}}(t_i)$). Knowing that $\forall t, t \cdot \mathbb{G} \Leftrightarrow t \in arts_{\mathbb{G}}$ (see proof of proposition 17), it is deduced that $t_i \cdot \mathbb{G}$ and the proof of this proposition is made. \square

By applying the GMWf projection algorithm presented above to the running example, one obtain the productions listed in table VIII for the different agents respectively. Let us note that this algorithm simply project each target artifact according to the view of the considered agent, then gather the productions in the obtained partial replicas while removing the duplicates. In the illustrated case here,

we have considered an update of the GMWf of the peer-review process so that it validates the axioms' visibility assumption (see sec. III.4.3.2).

Table VIII – Local GMWf productions of all the agents involved in the peer-review process.

| Agent | Productions of local GMWf |
|------------------------|---|
| Editor in Chief (EC) | $ \begin{array}{l l l} P_1 : A_{\mathbb{G}} \rightarrow A & P_2 : A \rightarrow B \ ; D & P_3 : A \rightarrow C \ ; D \\ P_4 : C \rightarrow S1 \ ; F & P_5 : S1 \rightarrow S2 \ S3 & P_6 : S2 \rightarrow H1 \ ; I1 \\ P_7 : S3 \rightarrow H2 \ ; I2 & P_8 : B \rightarrow \varepsilon & P_9 : D \rightarrow \varepsilon \\ P_{10} : F \rightarrow \varepsilon & P_{11} : H1 \rightarrow \varepsilon & P_{12} : I1 \rightarrow \varepsilon \\ P_{13} : H2 \rightarrow \varepsilon & P_{14} : I2 \rightarrow \varepsilon & \end{array} $ |
| Associated Editor (AE) | $ \begin{array}{l l l} P_1 : A_{\mathbb{G}} \rightarrow A & P_2 : A \rightarrow C & P_3 : C \rightarrow E \ ; F \\ P_4 : E \rightarrow S1 \ S2 & P_5 : S1 \rightarrow H1 \ ; I1 & P_6 : S2 \rightarrow H2 \ ; I2 \\ P_7 : H1 \rightarrow \varepsilon & P_8 : I1 \rightarrow \varepsilon & P_9 : H2 \rightarrow \varepsilon \\ P_{10} : I2 \rightarrow \varepsilon & P_{11} : F \rightarrow \varepsilon & P_{12} : A_{\mathbb{G}} \rightarrow \varepsilon \end{array} $ |
| First referee (R1) | $ \begin{array}{l l l} P_1 : A_{\mathbb{G}} \rightarrow C & P_2 : C \rightarrow G1 & P_3 : G1 \rightarrow H1 \ ; I1 \\ P_4 : H1 \rightarrow \varepsilon & P_5 : I1 \rightarrow \varepsilon & P_6 : A_{\mathbb{G}} \rightarrow \varepsilon \end{array} $ |
| Second referee (R2) | $ \begin{array}{l l l} P_1 : A_{\mathbb{G}} \rightarrow C & P_2 : C \rightarrow G2 & P_3 : G2 \rightarrow H2 \ ; I2 \\ P_4 : H2 \rightarrow \varepsilon & P_5 : I2 \rightarrow \varepsilon & P_6 : A_{\mathbb{G}} \rightarrow \varepsilon \end{array} $ |

III.4.4. The Artifact-Centric Choreography

In this section, we are interested in the actual execution of a process \mathcal{P}_{op} whose GMWf is $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$.

III.4.4.1. Initial Configuration of an Agent

Each agent i taking part in the choreography, has a single identifier (its ID). For a proper execution, it manages a local copy of the process' global GMWf \mathbb{G} , accreditations of various agents involved and its local GMWf \mathbb{G}_i . In addition, it handles a list RET_i of agents who have made requests and whose answers are yet to be sent, as well as two queues: REQ_i which stores requests waiting to be executed, and ANS_i which temporally stores answers received from agents to which requests were previously made. A local copy t_i of the mobile artifact and its (potentially partial) replica t_{γ_i} are also handled by agent i .

III.4.4.2. The Execution Choreography and Agent's Behaviour

The execution of an instance of the process is triggered when an artifact t is introduced into the system (on the appropriate agent); this artifact is in fact an unlocked bud of the type of one axiom $A_G \in \mathcal{A}$ (initial task) of the (global) GMWf \mathbb{G} .

An artifact that arrives on a given agent is either a request or a response to a request; depending on the case, it is inserted in the appropriate queue (REQ_i or ANS_i). As soon as possible¹¹, the artifact is removed from the queue, merged with the local copy (if it exists) and is then completed as needed. Completing an artifact consists of executing in a coherent way, the various tasks it imposes, i.e. those on which the current agent is accredited in writing.

At the end of the completion on an artifact, if its configuration shows that it must be completed by other agents (this is the case if the artifact contains buds created by the current agent and whose agent accredited in writing, are remote), replicas of the artifact are sent to the said agents by invoking the service *forwardTo*. Otherwise, the artifact is complete (it contains no more buds), or semi-complete (it contains buds that had been created by other agents and on which, the current agent is not accredited in writing); in which case, a replica is returned to the agent from which the artifact was previously received by invoking the service *returnTo*.

The execution of the process ends when all the tasks constituting a scenario of the process have been executed. In this case, the artifact that is cooperatively edited is complete (closed) on the agent where the process was triggered.

III.4.4.3. The Protocols

The activity that takes place on an agent in relation to the handling of a given artifact, breaks down into five sub-activities (see fig. 41); each of them is managed by a dedicated protocol. These activities are the following:

- *creation* (initialisation of a new case) or *receipt-merger* of a replica.
- *replication*: it consists in the extraction (from the local replica) of the partial replica that the local agent has to complete (manage its execution).
- *execution*: it consists in the extension by the local actor (via the specialised editor) of the buds for which he is accredited in writing.
- *expansion-pruning*: it consists in the reconstruction by expansion of the local (global) replica from the updated local partial replica.

¹¹. For instance at the end of the local replica completion or after a given time interval.

- *diffusion*: it corresponds to the return of the response to a request, or to the sending of requests.

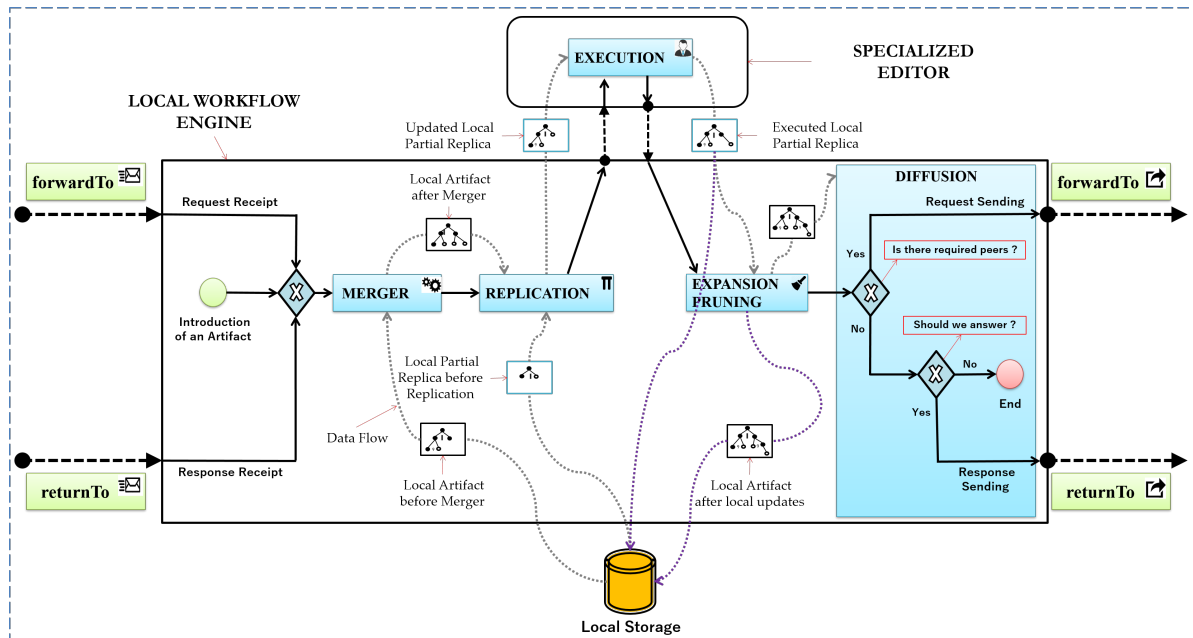


Figure 41 – Activity of an agent in the system.

The management protocols for these different activities are described in the following paragraphs.

The Receipt-Merger Protocol

An artifact is received either when a new case is initialised or after a request or a response is delivered. In all cases, a merge using an adaptation of the algorithm in (Badouel and Tchoupé Tchendji, 2008) is performed. The goal in this step is to update the local copy t_i of the global artifact from those received (the $(t_j^{maj})_{1 \leq j \neq i \leq k}$ contained in queues REQ_i and ANS_i). For that purpose (see algorithm 7), we merge t_i with each artifact t_j^{maj} from the requests queue (algorithm 7, lines 2 to 6) and from the responses queue (algorithm 7, lines 7 to 10) until they are empty. At each merge, t_i is updated (algorithm 7, lines 3 and 8). For each received request, the identity of the sender is kept in the list RET_i (algorithm 7, line 5) to be able to return a response at the end of the request processing. Note that during the merge, some previously locked buds can be unlocked: this is the case if all the tasks that precede them have been executed.

Algorithm 7 Merger protocol executed by an agent i .

```

1: procedure MERGER
2:   for  $req : REQ_i$  do ▷ While there is a request
3:      $t_i \leftarrow merge(t_i, req.artifact)$  ▷ We merge the artifact of the request with  $t_i$ 
4:     delete  $req$  from  $REQ_i$ 
5:      $enqueue(req.sender, RET_i)$  ▷ And we add the request sender in  $RET_i$  queue
6:   end for
7:   for  $ans : ANS_i$  do
8:      $t_i \leftarrow merge(t_i, ans.artifact)$  ▷ We merge the artifact of the answer with  $t_i$ 
9:     delete  $ans$  from  $ANS_i$ 
10:  end for
11: end procedure

```

The Replication Protocol

Replication is done just after the merge. The objective here is to update the local partial replica $t_{\mathcal{V}_i}$ from the local (global) artifact t_i . To do this (see algorithm 8), the local workflow engine proceeds as follows:

- It realises the expansion¹² of the partial replica $t_{\mathcal{V}_i}$ to obtain a global artifact t_i^{maj} which integrates all the updates made during the previous execution (algorithm 8, line 2). This operation is necessary, since at the end of the previous expansion, there may have been a pruning (see *the expansion-pruning protocol*, page 118) which removed from the global artifact local copy t_i , some updates contained in $t_{\mathcal{V}_i}$;
- Then, it merges t_i and t_i^{maj} in one artifact t_{i_f} (algorithm 8, line 3);
- Finally, it realises the projection of t_{i_f} relatively to the view \mathcal{V}_i to obtain the new version of $t_{\mathcal{V}_i} = \pi_{\mathcal{V}_i}(t_{i_f})$ (algorithm 8, line 4).

Algorithm 8 Replication protocol executed by an agent i .

```

1: procedure REPLICATION
2:    $t_i^{maj} \leftarrow expand(t_{\mathcal{V}_i}, t_i, \mathcal{V}_i, \mathbb{G})$ 
3:    $t_{i_f} \leftarrow merge(t_i, t_i^{maj})$ 
4:    $t_{\mathcal{V}_i} \leftarrow projection(t_{i_f}, \mathcal{V}_i, \mathbb{G}_i)$ 
5: end procedure

```

12. It is important to note that the expansion algorithm used here only returns one artifact (see *the expansion-pruning protocol*, page 118), unlike the one presented in (Badouel and Tchoupe Tchendji, 2008) which generates a potentially infinite family of artifacts represented by a tree automaton. This uniqueness is guaranteed by the fact that the expansion of $t_{\mathcal{V}_i}$ into t_i^{maj} is done using a three-way approach (*three-way merge* (Mens, 2002)). In fact, the expansion is carried out based on the grammatical model \mathbb{G} and on the view \mathcal{V}_i , but also on the prefix t_i of (the local global artifact replica) t_i^{maj} .

The Execution Protocol

This protocol (algorithm 9) is executed after the production of the local partial replica $t_{\mathcal{V}_i}$ by an agent i . It is executed by the local actor through the specialised editor, in order to extend the (unlocked) buds of $t_{\mathcal{V}_i}$ on which, he is accredited in writing.

The execution of the artifact's local replica by the agent i , must be done "as far as possible" by respecting the scheduling (sequential or parallel) of the tasks. Indeed, during the extension of a bud, if there is unlocking or creation of new unlocked buds on which the current agent is accredited in writing, its actor must extend/execute them; this is the purpose of the *while* loop in algorithm 9. In addition, the extension of buds whose type S corresponds to a (re)structuring symbol, is automatically done by the local workflow engine when the local GMWf has only one S -production.

Algorithm 9 Execution protocol executed by an agent i .

```

1: procedure EXECUTION
2:   while not isEmpty(buds  $\leftarrow$  nextLocalUnlockedBuds( $t_{\mathcal{V}_i}$ ,  $\mathcal{A}_{A_i(w)}$ )) do  $\triangleright$  While there are tasks
   that can be concurrently executed by the actor  $A_i$  of agent  $i$  in the partial replica  $t_{\mathcal{V}_i}$ 
3:     bud  $\leftarrow$  prompt("Choose a task to execute", buds)  $\triangleright$  Actor  $A_i$  chooses the task (bud) to
   execute
4:     prods  $\leftarrow$  localExecutionPossibilities(bud.type)  $\triangleright$  The specialized editor (agent) generates
   and activates the set of execution possibilities according to the current (local) configuration
5:     choice  $\leftarrow$  prompt("Choose an execution possibility", prods)  $\triangleright$  Actor  $A_i$  executes the
   selected task and provide feedback through the specialized editor
6:      $t_{\mathcal{V}_i} \leftarrow$  updateArtifact(choice,  $t_{\mathcal{V}_i}$ )  $\triangleright$  Then  $t_{\mathcal{V}_i}$  is updated accordingly
7:   end while
8: end procedure

```

The Expansion-Pruning Protocol

After completion of the partial replica $t_{\mathcal{V}_i}$, the updates must be propagated to the local (global) replica t_i of the artifact. This makes it possible to highlight (if they exist) the tasks for which requests must be made, or to determine if answers to requests can be returned. Algorithm 10 allows addressing this concern. For that, the expansion of the local updated partial replica $t_{\mathcal{V}_i}$ is made (algorithm 10, line 2) to obtain the global artifact t_i^{maj} which integrates all the contributions made by actor A_i during the previous local execution phase.

Note that the artifact t_i^{maj} may have so called *upstairs buds*¹³ (these are the internal nodes of t_i^{maj} that do not belong to $\mathcal{V}_i = \mathcal{A}_{A_i(r)}$, and which are not in t_i - see fig. 44). To prevent and manage this situation, a pruning of t_i^{maj} is performed (algorithm 10, line 3) to ensure compliance with task-related precedence constraints of executions. To do this, for every path of t_i^{maj} starting from the root, we prune at the level of the first (upstairs) bud encountered; it must appear unlocked if all the tasks that precede it have already been executed. The artifact obtained after this phase is the new version of t_i , and represents the current state of the process execution from the point of view of agent i .

Algorithm 10 Expansion-Pruning protocol executed by an agent i .

```

1: procedure EXPANSION-PRUNING
2:    $t_i^{maj} \leftarrow \text{expand}(t_{\mathcal{V}_i}, t_i, \mathcal{V}_i, \mathbb{G})$ 
3:    $t_i \leftarrow \text{pruning}(t_i^{maj}, t_i)$ 
4: end procedure

```

A three-way merging expansion algorithm: consider an (global) artifact under execution t , and $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ its partial replica on the site of an actor A_i whose view is \mathcal{V} . Consider the partial replica $t_{\mathcal{V}}^{maj} \geq t_{\mathcal{V}}$ obtained by developing some unlocked buds of $t_{\mathcal{V}}$ as a result of A_i 's contribution. The expansion problem consists in finding an (global) artifact under execution t_f , which integrates nodes of t and $t_{\mathcal{V}}$. To solve this problem made difficult by the fact that t and $t_{\mathcal{V}}$ are conform to two different models (\mathbb{G} and $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$), we perform a three-way merge (Mens, 2002). We merge the artifacts t and $t_{\mathcal{V}}$ using a (global) target artifact t_g such that:

- (a) t is a prefix of t_g ($t \leq t_g$)
- (b) $t_{\mathcal{V}}^{maj}$ is a prefix of the partial replica of t_g according to \mathcal{V} ($t_{\mathcal{V}}^{maj} \leq \pi_{\mathcal{V}}(t_g)$)

The proposed algorithm proceeds in two steps.

Step 1 - Search for the merging guide t_g : the search of a merging guide is done by the algorithm 11.

Step 2 - Merging t , $t_{\mathcal{V}}^{maj}$ and t_g : we want to find an artifact t_f that includes all the contributions already made during the workflow execution. The structure of the

13. Intuitively, a node $n_{X_{\circ}}$ associated with the task X is an *upstairs bud* if, X (not already executed) precedes at least one task Y made visible (and naturally not already executed) on the site of an agent i , i not having any accreditation in reading on X .

Algorithm 11 Algorithm to search a merging guide.

1. First of all, we have to generate the set $arts_{\mathbb{G}} = \{t_1, \dots, t_n\}$ of target artifacts denoted by \mathbb{G} ;
 2. Then, we must filter this set to retain only the artifacts t_i admitting t as a prefix (criterion **(a)**) and whose projections according to \mathcal{V} ($t_{i_{\mathcal{V}_j}}$) admit $t_{\mathcal{V}}^{maj}$ as a prefix (criterion **(b)**). It is said that an artifact t_a (whose root node is $n_a = X_a[t_{a_1}, \dots, t_{a_l}]$) is a prefix of a given artifact t_b (whose root node is $n_b = X_b[t_{b_1}, \dots, t_{b_m}]$) if and only if the root nodes n_a and n_b are of the same types (i.e $X_a = X_b$) and:
 - a. The node n_a is a bud or,
 - b. The nodes n_a and n_b have the same number of sub-artifacts (i.e $l = m$), the same type of scheduling for the sub-artifacts and each sub-artifact t_{a_i} of n_a is a prefix of the sub-artifact t_{b_i} of n_b .
 We obtain the set $guides = \{t_{g_1}, \dots, t_{g_k}\}$ of artifacts that can guide the merging;
 3. Finally, we randomly select an element t_g from the set $guides$.
-

searched artifact t_f is the same as that of t_g : hence the interest to use t_g as a guide. The merging is carried out by the algorithm 12.

Algorithm 12 Three-way merging algorithm.

- A prefixed depth path of the three artifacts (t , $t_{\mathcal{V}}^{maj}$ and t_g) is made simultaneously until there is no longer a node to visit in t_g . Let n_{t_i} (resp. $n_{t_{\mathcal{V}_j}^{maj}}$ and $n_{t_{g_k}}$) be the node located at the address w_i (resp. w_j and w_k) of t (resp. $t_{\mathcal{V}}^{maj}$ and t_g) and currently being visited. If nodes n_{t_i} , $n_{t_{\mathcal{V}_j}^{maj}}$ and $n_{t_{g_k}}$ are such that **(processing)**:
1. $n_{t_{\mathcal{V}_j}^{maj}}$ is associated with a (re)structuring symbol (fig. 42(d)) then: we take a step forward in the depth path of $t_{\mathcal{V}}^{maj}$ and we resume processing;
 2. n_{t_i} , $n_{t_{\mathcal{V}_j}^{maj}}$ and $n_{t_{g_k}}$ exist and are all associated with the same symbol X (fig. 42(a) and 42(b)) then: we insert $n_{t_{\mathcal{V}_j}^{maj}}$ (it is the most up-to-date node) into t_f at the address w_k ; if $n_{t_{\mathcal{V}_j}^{maj}}$ is a bud then we prune (delete sub-artifacts) t_g at the address w_k ; we take a step forward in the depth path of the three artifacts and we resume processing.
 3. n_{t_i} , $n_{t_{\mathcal{V}_j}^{maj}}$ and $n_{t_{g_k}}$ exist and are respectively associated with symbols X_i , X_j and X_k such that $X_k \neq X_i$ and $X_k \neq X_j$ (fig. 42(e)) then: we add $n_{t_{g_k}}$ in t_f at address w_k . This is an upstairs bud; we take a step forward in the depth path of t_g and we resume processing.
 4. n_{t_i} (resp. $n_{t_{\mathcal{V}_j}^{maj}}$) and $n_{t_{g_k}}$ exist and are associated with the same symbol X (fig. 42(c) and 42(f)) then: we insert n_{t_i} (resp. $n_{t_{\mathcal{V}_j}^{maj}}$) into t_f at the address w_k ; if n_{t_i} (resp. $n_{t_{\mathcal{V}_j}^{maj}}$) is a bud, we prune t_g at the address w_k ; we take a step forward in the depth path of the artifacts t (resp. $t_{\mathcal{V}}^{maj}$) and t_g , then we resume processing.
-

As for the other key algorithms, a Haskell implementation of this expansion-pruning algorithm is introduced in appendix A.

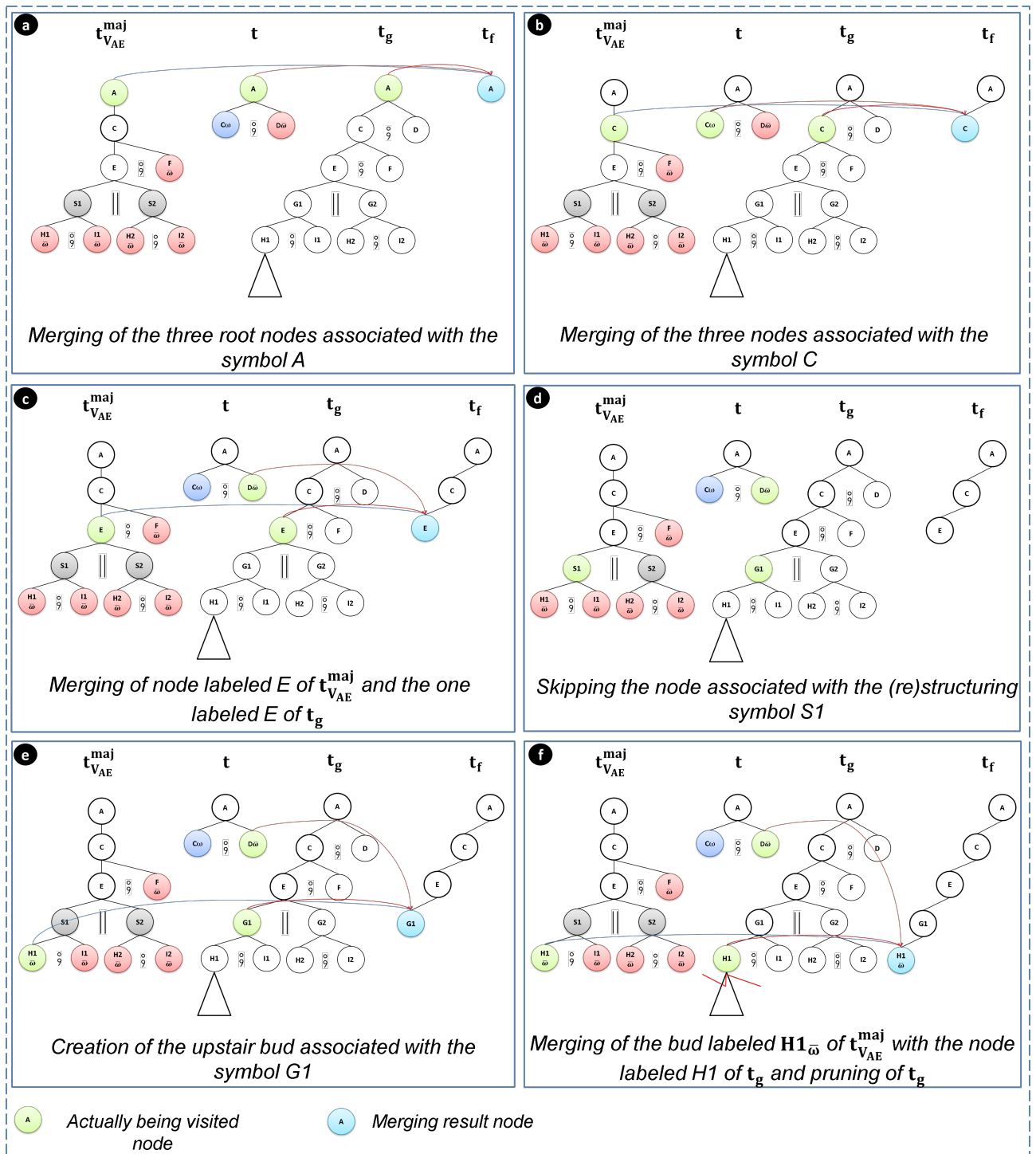


Figure 42 – Some scenarios to be managed during the expansion.

Proposition 19 For any update t_{ν}^{maj} in accordance with a $GMWf \mathbb{G}_{\nu} = \Pi_{\nu}(\mathbb{G})$, of a partial replica $t_{\nu} = \pi_{\nu}(t)$ obtained by projecting (according to the view \mathcal{V}) an artifact t being executed in accordance with the $GMWf \mathbb{G}$ of a $GMAWfP$ verifying the axioms' visibility and the non-recursivity assumptions, there is at least one target artifact (the three-way merge guide) $t_g \in arts_{\mathbb{G}}$ of \mathbb{G} such as:

- (a) t is a prefix of t_g ($t \leq t_g$)
- (b) $t_{\mathcal{V}}^{maj}$ is a prefix of the partial replica of t_g according to \mathcal{V} ($t_{\mathcal{V}}^{maj} \leq \pi_{\mathcal{V}}(t_g)$)

Proof. Thanks to the proposals 16, 17 and the artifact editing model used here (see sec. III.3.3.3) it is established that since the artifact t being executed in accordance with \mathbb{G} is a prefix of a non-empty set of \mathbb{G} 's target artifacts $arts'_{\mathbb{G}} = \{t'_1, \dots, t'_n\}$ ($\forall 1 \leq i \leq n, t \leq t'_i$), its projection $t_{\mathcal{V}}$ according to the view \mathcal{V} is a prefix of a non-empty set $arts'_{\mathbb{G}_{\mathcal{V}}} = \{t'_{\mathcal{V}_1}, \dots, t'_{\mathcal{V}_m}\}$ of $\mathbb{G}_{\mathcal{V}} = \Pi_{\mathcal{V}}(\mathbb{G})$'s local target artifacts for the said view ($\forall 1 \leq j \leq m, t_{\mathcal{V}} \leq t'_{\mathcal{V}_j}$): elements of $arts'_{\mathbb{G}}$ are potential merging guides candidates that all verify the property (a). In addition, using the propositions 16 and 18, it is established that each element of $arts'_{\mathbb{G}_{\mathcal{V}}}$ is the projection of at least one element of $arts'_{\mathbb{G}}$ according to the view \mathcal{V} (1). Given that $t_{\mathcal{V}}^{maj}$ is obtained by developing buds of $t_{\mathcal{V}}$ in accordance with $\mathbb{G}_{\mathcal{V}}$, it is inferred that $t_{\mathcal{V}}^{maj}$ is a prefix of a non-empty subset $arts^{maj}_{\mathbb{G}_{\mathcal{V}}} \subseteq arts'_{\mathbb{G}_{\mathcal{V}}}$ of local target artifacts for the view \mathcal{V} (2). With the proposition 18 once again, it is determined that for each artifact $t'_{\mathcal{V}_j} \in arts^{maj}_{\mathbb{G}_{\mathcal{V}}}$, there is at least one artifact t_{g_j} that is conform to \mathbb{G} such as $t'_{\mathcal{V}_j} = \pi_{\mathcal{V}}(t_{g_j})$: this new set $arts^{maj}_{\mathbb{G}} = \{t_{g_1}, \dots, t_{g_k}\}$ is made up of potential merging guides candidates that all verify the property (b). Results (1) and (2) show that $arts^{maj}_{\mathbb{G}}$ and $arts'_{\mathbb{G}}$ are not disjoint. As a consequence, the set $guides = arts^{maj}_{\mathbb{G}} \cap arts'_{\mathbb{G}}$ of potential merging guides that all verify both property (a) and (b) is not empty. \square

Corollary 20 *For an artifact t being executed in accordance with a GMWf \mathbb{G} of a GMAWfP verifying the axioms' visibility and the non-recursivity assumptions, and an update $t_{\mathcal{V}}^{maj} \geq t_{\mathcal{V}}$ of its partial replica $t_{\mathcal{V}} = \pi_{\mathcal{V}}(t)$ according to the view \mathcal{V} , the expansion of $t_{\mathcal{V}}^{maj}$ contains at least one artifact and the expansion-pruning algorithm presented here returns one and only one artifact.*

This result (corollary 20) derives from the proof of the proposition 19 (there is always at least one artifact in the expansion of $t_{\mathcal{V}}^{maj}$ under the conditions of corollary 20) and from the fact that in the last instruction of the algorithm ??, an artifact is randomly selected and returned from a non-empty set of potential guides (only one of the expansion artifacts is used in the three-way merging).

The Diffusion Protocol

After expansion-pruning, the local workflow engine must examine whether requests need to be sent to other agents (this is the case if t_i still have unlocked buds

created on its site¹⁴, on which the current agent is not accredited in writing) or, if responses are to be returned (this is the case if t_i is complete or semi-complete).

To build the list of requests to diffuse, the local workflow engine scans the artifact t_i produced by expansion-pruning and builds the list of required agents (those to receive a request) from buds¹⁵ (algorithm 13, lines 2 to 7). If the required agents list is not empty (the artifact is not complete), it sends a request to each agent in the list (algorithm 13, lines 8 to 12). Otherwise, if there are agents who have previously made requests, it sends responses instead (algorithm 13, lines 13 to 18).

Algorithm 13 Diffusion protocol executed by an agent i .

```

1: procedure DIFFUSION
2:   for  $bud : unlockedBuds(t_i)$  and  $bud$  have been created by  $i$  do
3:     if  $bud.type \in \mathcal{A}_{A_i(x)}$  then
4:        $agent \leftarrow executorOf(bud.type)$ 
5:        $enqueue(agent, requiredAgents)$ 
6:     end if
7:   end for
8:   if  $not isEmpty(requiredAgents)$  then
9:      $req \leftarrow new Request(i, t_i)$ 
10:    for  $agent : requiredAgents$  do
11:       $invokeService("forwardTo", req, agent)$ 
12:    end for
13:   else if  $not isEmpty(RET_i)$  then
14:      $ans \leftarrow new Answer(i, t_i)$ 
15:     for  $agent : RET_i$  do
16:        $invokeService("returnTo", ans, agent)$ 
17:        $delete agent from RET_i$ 
18:     end for
19:   else
20:      $alert("The process execution is terminated.")$ 
21:   end if
22: end procedure

```

III.5. Illustrating the Choreography on the Peer-Review Process

The execution of an instance of our running example begins when under the editor in chief's action (via the GUI of the specialised editor), an unlocked bud of

14. An agent only requests the execution of a bud if it was created on its site.

15. Normally (due to assumptions of our model) at this stage, for each unlocked bud of t_i , agent i is accredited in execution on the associated task. Any other situation would be a design flaw.

type A_G is created on his site. Figure 43 which must be read following the direction of the arrows it contains, summarises the state¹⁶ of the agent EC (editor in chief site) before and after the event creating the artifact; it also illustrates the running of the five-step protocol on the agent EC .

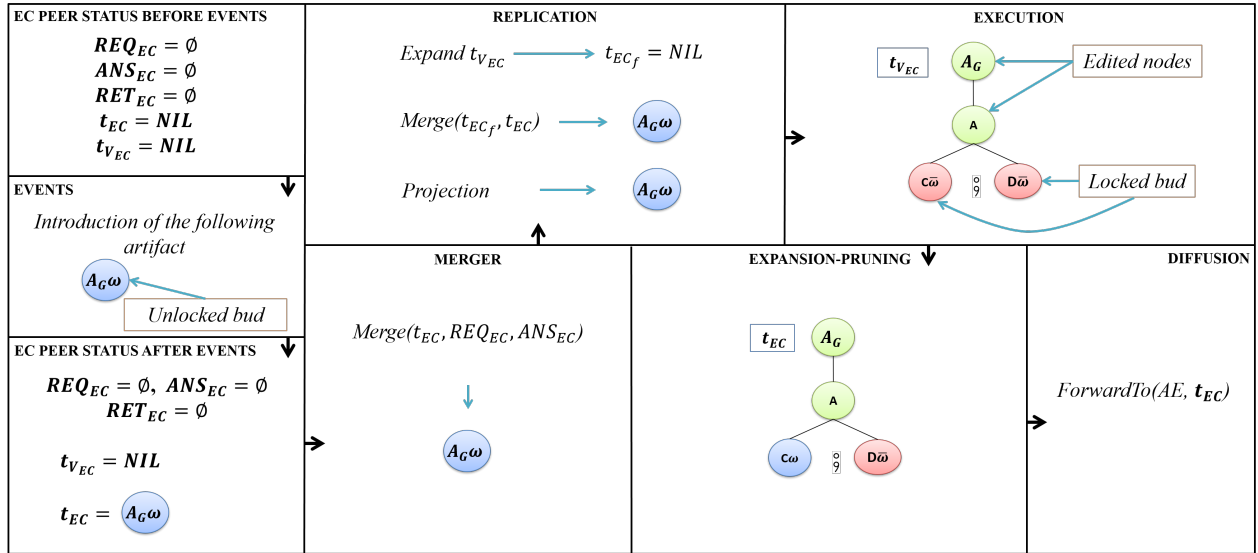


Figure 43 – Beginning of the peer-review process on the editor in chief's site.

As soon as a bud of type A_G is created, the local workflow engine extends it using the unique A_G -production ($P_1 : A_G \rightarrow A$) of the local GMWf. This results in the creation of a bud of type A that the editor in chief must extend via the specialised editor by choosing an A -production. For this scenario, it is assumed that he chooses the production $P_3 : A \rightarrow C \circlearrowleft D$. The task (A) executed by the latter is shown in green colour on figure 43. The newly created tasks (C and D) appear in the form of locked buds (the locked buds are shown in red colour) because the editor in chief is not accredited in writing on C and, since D is linked to C by a sequential scheduling constraint, it can only be executed when all tasks ($C, E, F, G1, G2, H1, H2, I1, I2$) preceding it will have been executed. After expansion-pruning, the only required agent is the associated editor (responsible for executing task C): a request is sent to it by invoking the service *forwardTo*.

The event that triggers the workflow execution on the site of associated editor (fig. 44) is the receipt of the request sent by the editor in chief. In the artifact sent by the latter, there are buds (C_{ω} and D_{ω}). After merging, the bud C_{ω} is unlocked (the unlocked buds are shown in blue). It indicates the only place where the contribution of the associated editor is expected. During the execution phase, the

16. The state of an agent i at a given moment is given by the values of variables REQ_i, ANS_i, RET_i and the replicas t_i and $t_{\mathcal{V}_i}$.

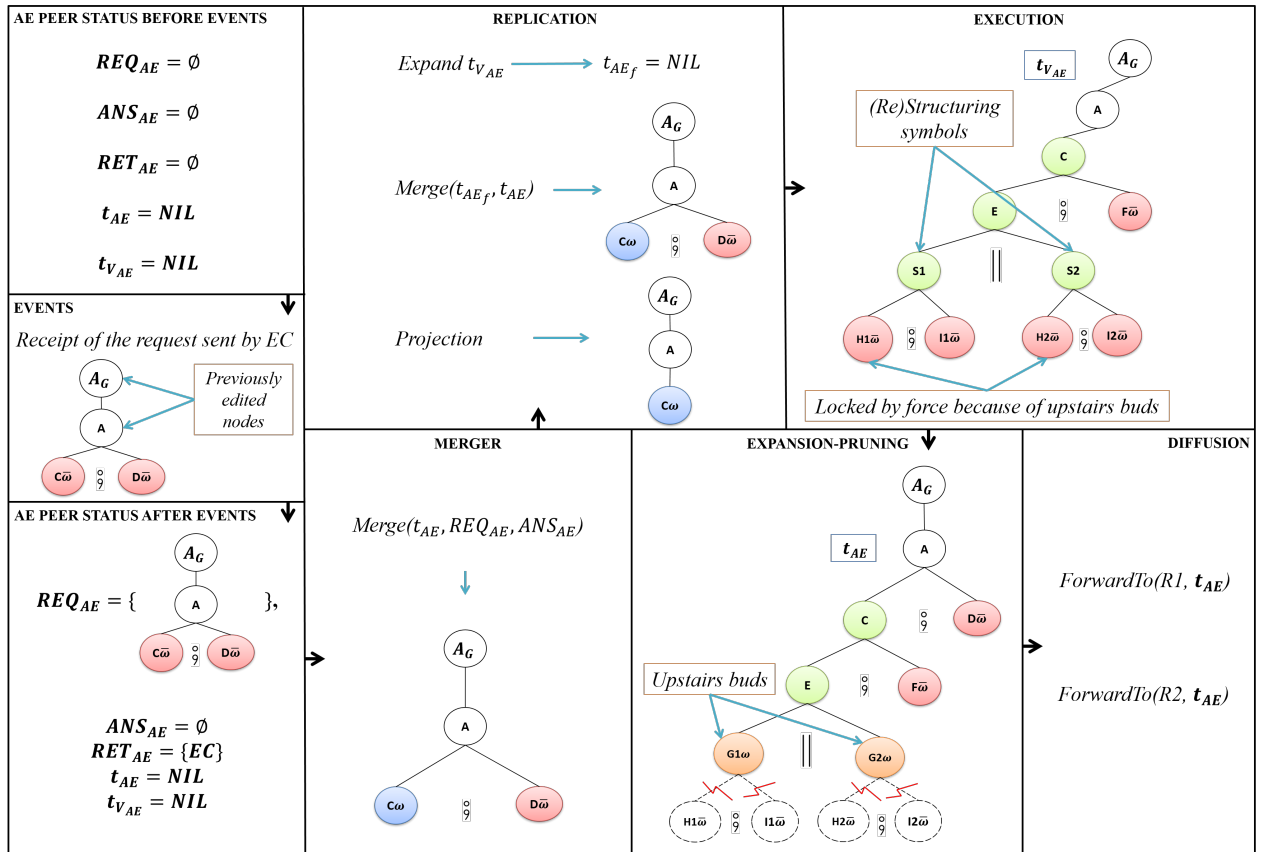


Figure 44 – Continuation of the peer-review process execution on the associated editor's site; the latter receives the request formulated by the editor in chief.

local artifact partial replica is updated by the associated editor via the productions $P_3 : C \rightarrow E \ ; \ F$, $P_4 : E \rightarrow S1 \ || \ S2$, $P_5 : S1 \rightarrow H1 \ ; \ I1$ and $P_6 : S2 \rightarrow H2 \ ; \ I2$ of his local GMWf. At the end of this phase, buds of types $H1, H2, I1$ and $I2$ appear locked not only because they are constrained by a sequential scheduling (case of $I1$ and $I2$), but especially because of the presence of *upstairs buds* (the upstairs buds are represented in orange colour). Indeed, tasks $G1$ and $G2$ made visible after the expansion are upstairs buds because they must be executed before the tasks of type $H1$ and $H2$. So, there is pruning at $G1$ and $G2$ before sending (in parallel) the artifact to both referees.

Figure 45 illustrates how the protocol takes place on the site of one of the referees (the first referee). After the contribution of the latter, no new bud is created: no request is formulated. It is rather a response corresponding to the request previously received from the associated editor which is returned by invoking the service *returnTo*.

The execution protocol is unrolled again on the site of the associated editor following events related to the reception of responses from the two referees (fig.

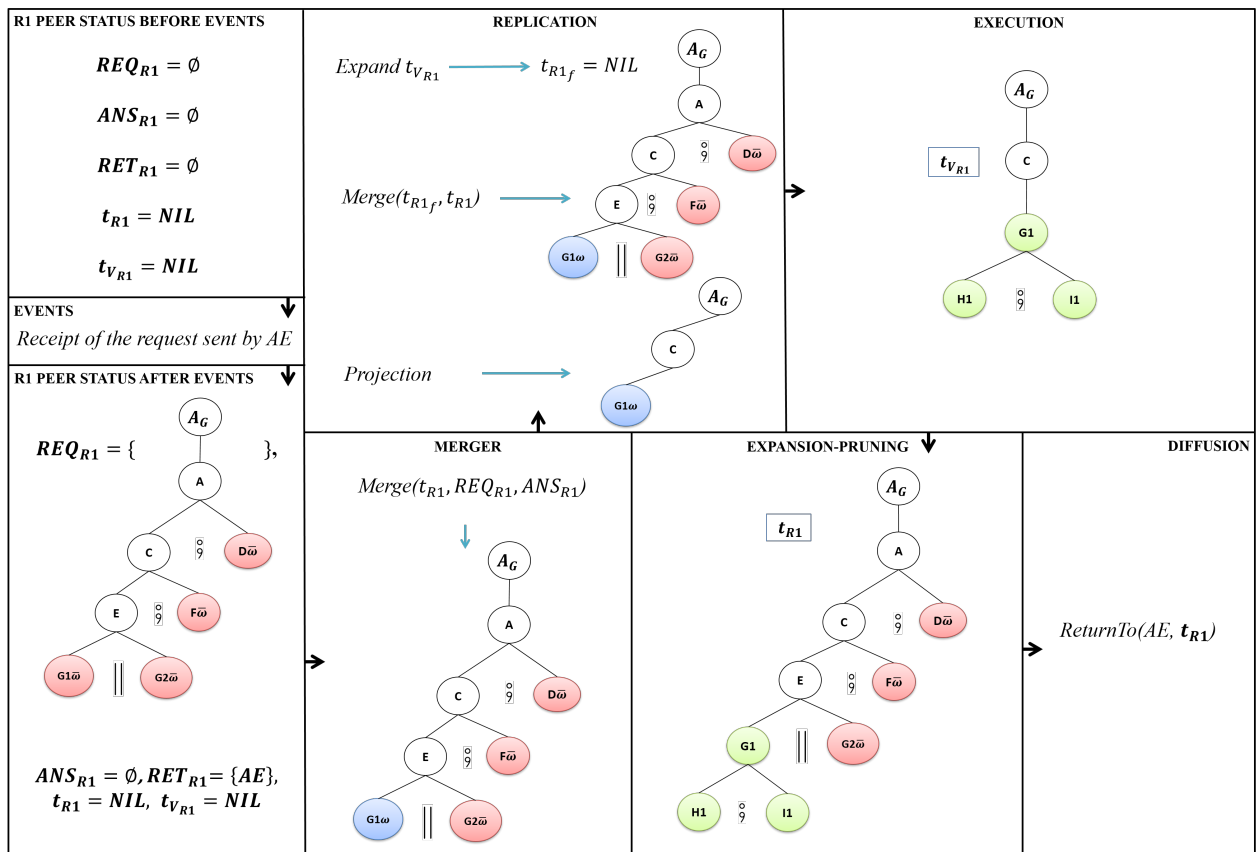


Figure 45 – Continuation of the peer-review process execution on the first referee’s site: the request of the associated editor arrives at the first referee.

46). We choose to treat these responses simultaneously; but we could do otherwise and obtain the same result. At merge, since the subtree rooted in E is closed, the bud $F_{\bar{\omega}}$ is unlocked and the associated editor extends it through production $P_{11} : F \rightarrow \epsilon$. Having no request to make, the answer to the request previously received from the editor in chief is returned.

The editor in chief receives the response from associated editor and once again runs the execution protocol (fig. 47). After its contribution (on the node D), the artifact obtained after expansion-pruning is closed and the execution of the process ends successfully.

III.6. Experimentation

In this section, we present and experiment *P2PTinyWfMS* (a Peer-to-Peer Tiny Workflow Management System), an experimental prototype system implemented according to the approach proposed in this chapter.

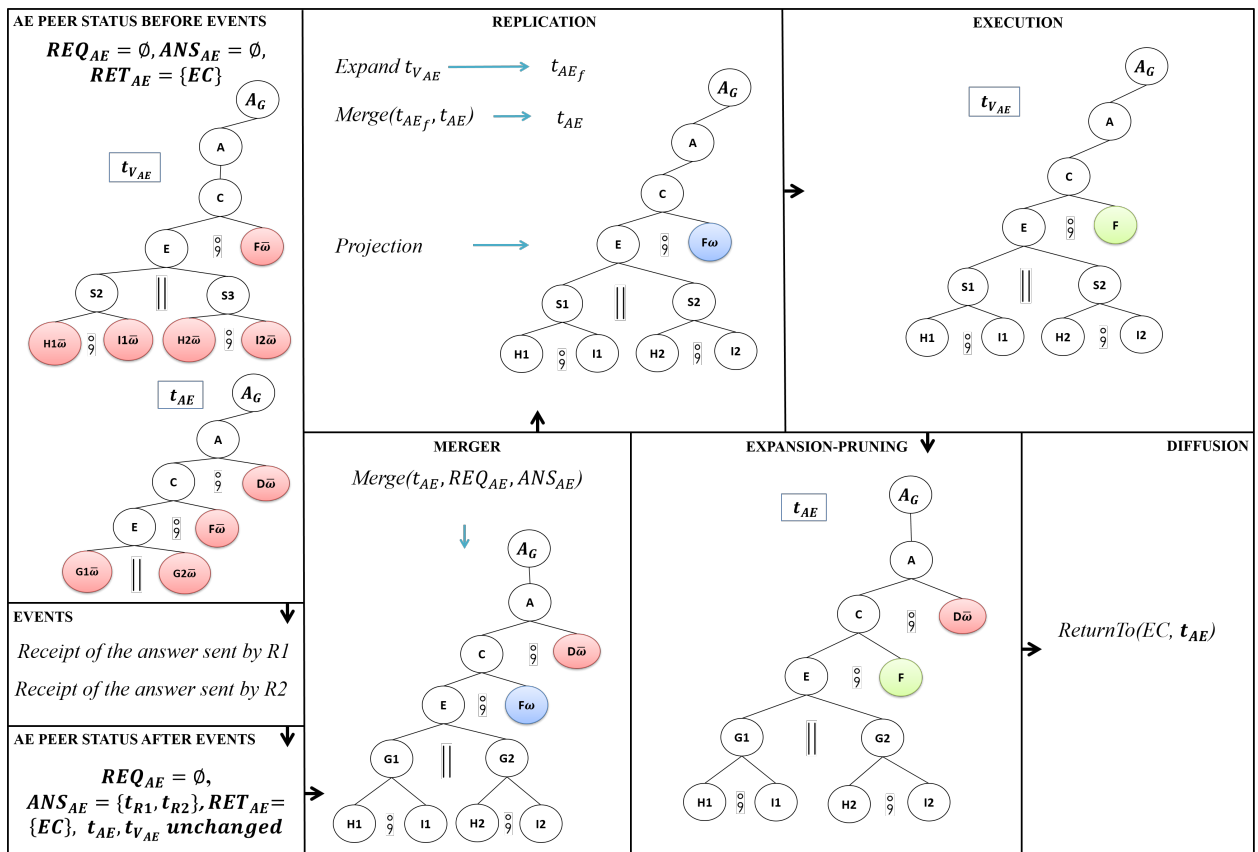


Figure 46 – Continuation of the peer-review process execution: the associated editor receives answers from referees, to requests that he has previously made.

III.6.1. P2PTinyWfMS: an Experimental Prototype System

P2PTinyWfMS is a tool developed in Java under Eclipse¹⁷ and dedicated to the distributed execution of administrative workflows specified using GMWf. In accordance with the agent’s architecture of this chapter (see fig. 41), *P2PTinyWfMS* has a front-end for displaying and graphically editing artifacts manipulated during the execution of a business process (see fig. 48 and 49), as well as a communication module built from SON¹⁸.

Let’s recall that SON (Shared-data Overlay Network) (Lahcen and Parigot, 2012) is a middleware offering several DSL to facilitate the implementation of P2P systems whose components communicate by service invocations. Component Description Meta Language (CDML) is the DSL provided by SON to specify among other things the services required and provided by each peers; from a CDML specification, SON generates Java code for allowing peers to communicate. The following listing shows the contents of the CDML file used in the case of

17. Official website of Eclipse: <https://www.eclipse.org>, visited the 04/04/2020.

18. SON is available under Eclipse from a family of SmartTools plugins.

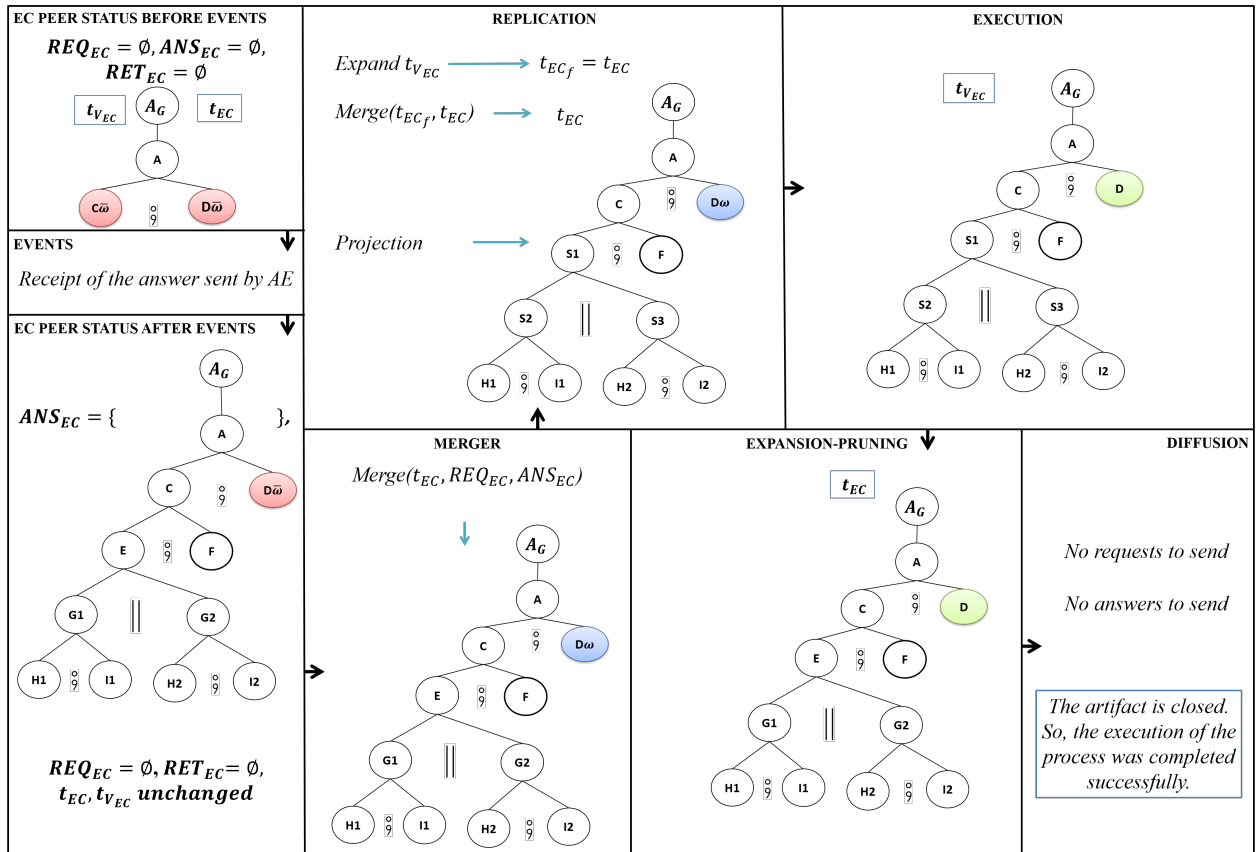


Figure 47 – Continuation and end of the peer-review process execution: the editor in chief receives a response containing referees' contributions, from the associated editor.

P2PTinyWfMS to specify the four services that its instances expose; they are: two input services (*inForwardTo* - lines 6 to 9 -, and *inReturnTo* - lines 10 to 13 -) and two output services (*outForwardTo* - lines 14 to 17 - and *outReturnTo* - lines 18 to 21 -). These services take as argument an artifact corresponding to either a request or a response.

```

_____ CDML file: specification of required and provided services of P2PTinyWfMS _____
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <component name="p2pTinyWfMS" type="p2pTinyWfMS" extends="inria.communicationprotocol"
3   ns="p2pTinyWfMS">
4   <containerclass name="P2pTinyWfMSContainer"/>
5   <facadeclass name="P2pTinyWfMSFacade" userclassname="P2pTinyWfMS"/>
6   <input name="forwardTo" method="inForwardTo">
7     <attribute name="request"
8       javatype="smartworkflow.dwfms.lifa.miu.util.p2pworkflow.PeerToPeerWorkflowRequest"/>
9   </input>
10  <input name="returnTo" method="inReturnTo">
11    <attribute name="response"
12      javatype="smartworkflow.dwfms.lifa.miu.util.p2pworkflow.PeerToPeerWorkflowResponse"/>
13  </input>
14  <output name="forwardTo" method="outForwardTo">
15    <attribute name="request"
16      javatype="smartworkflow.dwfms.lifa.miu.util.p2pworkflow.PeerToPeerWorkflowRequest"/>
17  </output>
18  <output name="returnTo" method="outReturnTo">
19    <attribute name="response"

```

```

20     javatype="smartworkflow.dwfms.lifa.miu.util.p2pworkflow.PeerToPeerWorkflowResponse"/>
21     </output>
22 </component>

```

III.6.2. Executing our Running Example under P2PTinyWfMS

SON offers a DSL (the ".world" files) for the description of the deployment of a distributed system whose components have been specified by a CDML file. In order to execute our running example, we deployed four instances of *P2PTinyWfMS* identified by *EC*, *AE*, *R1* and *R2* respectively. As explained in section III.5, each instance is initially equipped with the global GMWf as well as accreditations of various agents from which it derives its local GMWf by projection.

Figures 48, 49 and 50 are screen shots with some highlights of the workflow's distributed execution. We have the tab "Workflow overview" presenting at the beginning of the execution, various tasks, agents, target artifacts etc., on the editor in chief's site (fig. 48). We also have the tabs "Workflow execution" of the sites of the associated editor (fig. 49) and of the editor in chief (fig. 50) that present the artifacts resulting from their execution after receiving a request from the editor in chief (resp. after receiving a response from the associated editor).

The screenshot displays the P2PTinyWfMS interface. The 'Workflow tasks' table lists various tasks with their descriptions and symbols. The 'Workflow target artifacts' diagram shows a hierarchical structure of artifacts (Ag, A, B, D, C, E, F, G1, G2, H1, I1, H2, I2) connected by arrows. The 'Workflow Stakeholders' table shows the accreditations for each instance (EC, AE, R1, R2) across different artifacts. The 'Workflow overview' tab is highlighted with a red circle.

| ID | A(r) | A(w) | A(x) |
|----|-------------------------------------|---------------|----------|
| EC | [Ag, A, B, C, D, F, H1, H2, I1, I2] | [Ag, A, B, D] | [C] |
| AE | [Ag, A, C, E, F, H1, H2, I1, I2] | [C, E, F] | [G1, G2] |
| R1 | [Ag, C, G1, H1, I1] | [G1, H1, I1] | [] |
| R2 | [Ag, C, G2, H2, I2] | [G2, H2, I2] | [] |

Figure 48 – *P2pTinyWfMS* on the editor in chief's site: presentation of the GMWf (the tasks and their relations, the actors and their accreditations).

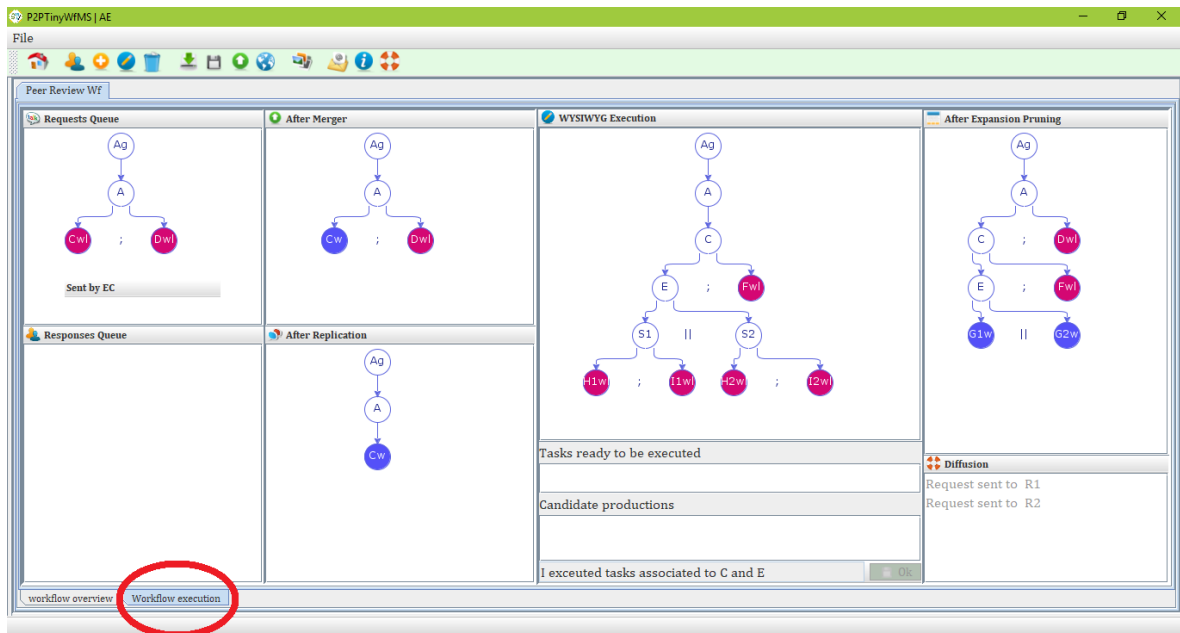


Figure 49 – P2pTinyWfMS on the associated editor’s site: receipt of editor in chief’s request, execution of tasks, expansion-pruning, and diffusion.

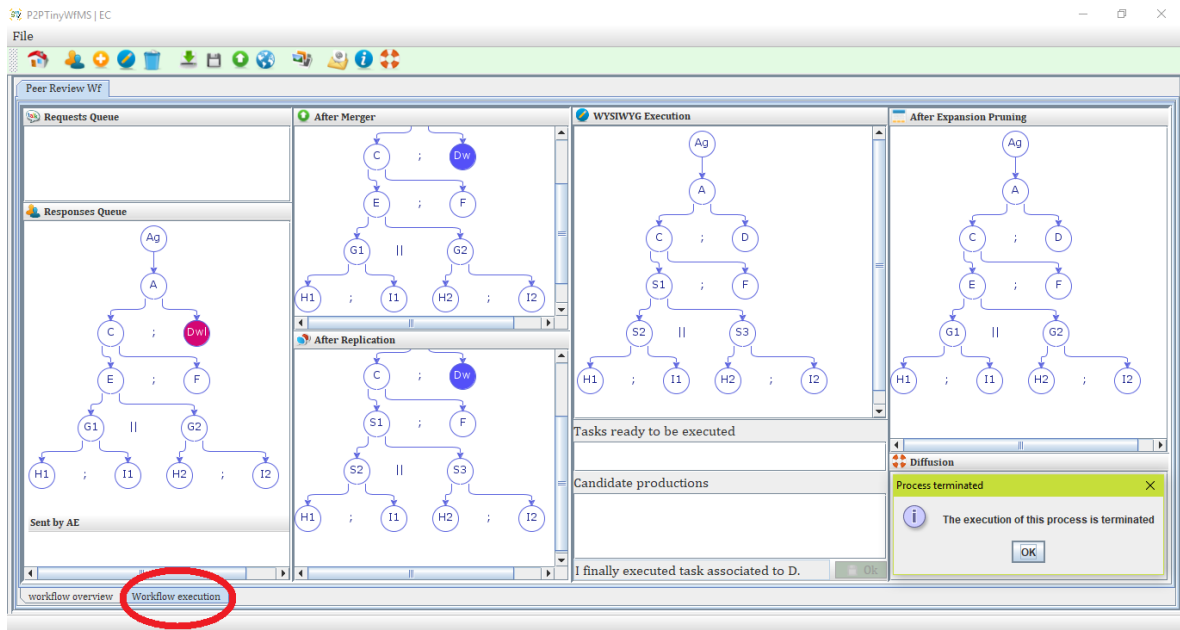


Figure 50 – P2pTinyWfMS on the editor in chief’s site: reception of the associated editor’s response, execution of tasks, expansion-pruning and end of the case.

III.7. Related Works and Discussion

In this section we briefly discuss the similarities and differences of the model presented in this chapter, comparing it with some related work presented earlier (Chapter I). We will mention a few related studies and discuss directly; a more

formal comparative study using qualitative and quantitative metrics should be the subject of future work.

Hull et al. (2009) provide an interoperation framework in which, data are hosted on central infrastructures named *artifact-centric hubs*. As in the work presented in this chapter, they propose mechanisms (including user views) for controlling access to these data. Compared to choreography-like approach as the one presented in this chapter, their settings has the advantage of providing a conceptual rendezvous point to exchange status information. The same purpose can be replicated in this chapter's approach by introducing a new type of agent called "*monitor*", which will serve as a rendezvous point; the behaviour of the agents will therefore have to be slightly adapted to take into account the monitor and to preserve as much as possible the autonomy of agents.

Lohmann and Wolf (2010) abandon the concept of having a single artifact hub (Hull et al., 2009) and they introduce the idea of having several agents which operate on artifacts. Some of those artifacts are mobile; thus, the authors provide a systematic approach for modelling artifact location and its impact on the accessibility of actions using a Petri net. Even though we also manipulate mobile artifacts, we do not model artifact location; rather, our agents are equipped with capabilities that allow them to manipulate the artifacts appropriately (taking into account their location). Moreover, our approach considers that artifacts can not be remotely accessed, this increases the autonomy of agents.

The process design approach presented in this chapter, has some conceptual similarities with the concept of *proclets* proposed by Wil M. P. van der Aalst et al. (2001; 2009): they both split the process when designing it. In the model presented in this chapter, the process is split into execution scenarios and its specification consists in the diagramming of each of them. Proclets (Van Der Aalst et al., 2001, 2009) uses the concept of *proclet-class* to model different levels of granularity and cardinality of processes. Additionally, proclets act like agents and are autonomous enough to decide how to interact with each other.

The model presented in this chapter uses an attributed grammar as its mathematical foundation. This is also the case of the AWGAG model by Badouel et al. (2014; 2015). However, their model puts stress on modelling process data and users as first class citizens and it is designed for Adaptive Case Management.

To summarise, the proposed approach in this chapter allows the modelling and decentralized execution of administrative processes using autonomous agents. In it, process management is very simply done in two steps. The designer only needs to focus on modelling the artifacts in the form of task trees and the rest is easily

deduced. Moreover, we propose a simple but powerful mechanism for securing data based on the notion of accreditation; this mechanism is perfectly composed with that of artifacts. The main strengths of our model are therefore :

- The simplicity of its syntax (process specification language), which moreover (well helped by the accreditation model), is suitable for administrative processes;
- The simplicity of its execution model; the latter is very close to the blockchain's execution model (Hull, 2017; Mendling et al., 2018). On condition of a formal study, the latter could possess the same qualities (fault tolerance, distributivity, security, peer autonomy, etc.) that emanate from the blockchain;
- Its formal character, which makes it verifiable using appropriate mathematical tools;
- The conformity of its execution model with the agent paradigm and service technology.

In view of all these benefits, we can say that the objectives set for this thesis have indeed been achieved. However, the proposed model is perfectible. For example, it can be modified to permit agents to respond incrementally to incoming requests as soon as any prefix of the extension of a bud is produced. This makes it possible to avoid the situation observed on figure 46 where the associated editor is informed of the evolution of the subtree resulting from C only when this one is closed. All the criticisms we can make of the proposed model in particular, and of this thesis in general, have been introduced in the general conclusion (page 134) of this manuscript.

III.8. Summary

In this chapter, we have proposed a new approach to facilitate workflow design and their execution in a distributed environment. The approach relies on the cooperative editing of a mobile artifact by several agents. The design of artifacts and their type (artifact type), the architecture and management protocols (choreography) of agents have been presented and discussed. Likewise, an illustration and an experimentation of the whole approach on a real example of an administrative business process (the peer-review process) were presented.

The proposed approach has many advantages, which we have outlined. In particular, it covers all aspects of process automation using BPM technology. In addition to the formal and demonstrative presentations, we have conducted a preliminary discussion of the presented approach in relation to existing work, and finally concluded that we have achieved the objective of this thesis.

GENERAL CONCLUSION

CONTENTS

| | |
|--|-----|
| Recall of this Thesis' Challenge and of our Methodological Choices . . . | 134 |
| A Critical Analysis of the Performed Work | 136 |
| Some Perspectives | 139 |

We hereby summarise the work reported and presented in this document. To this, we associate a critical analysis of our models and methodological choices. Finally, we present some research avenues that can be explored following the work in this thesis manuscript.

Recall of this Thesis' Challenge and of our Methodological Choices

In this thesis, we focused on the automation of business processes using the technological framework offered by the BPM domain. We have contributed to the ambition of making more accessible, administrative business processes automation through this technology. We were guided by the aim of increasing its success in business sectors using administrative processes as it has been in other sectors such as science, banking and insurance, which are governed by much more programmable processes. We thought that a first step towards achieving this great ambition was to rely on the current (most up-to-date) BPM paradigms and tools, to design and implement a new BPM framework that would be tailor-made for the management of administrative business processes. Having identified the artifact-centric BPM, structured cooperative edition, P2P computing, multiagent system and SOA concepts as being the hot topics in the implementation of BPM, we set the following goal for this thesis:

The proposal of a new artifact-centric framework, facilitating the modelling of administrative business processes and the completely decentralised execution of the resulting workflows; this completely decentralised execution being provided by a P2P system conceived as a set of agents communicating asynchronously by service invocation so that, the execution of a given workflow instance is technically assimilated to the cooperative edition of (mobile) structured documents called artifacts.

To achieve this goal, we have chosen to base our work on the structured document cooperative editing model developed by Badouel and Tchoupé a decade earlier. They proposed an approach based on grammatical models, to manage the lifecycle of a document collegially edited by actors on geographically distant sites. In their model, each actor has a potentially partial view (obtained by projection) of the edited document. The latter is used as an interface between the different actors of the system. When an actor receives a document he must know from its content, what he can do and/or what he has to do about it. The information contained in a document can only grow over its lifetime in the system. Since the system is distributed and under the assumption that several actors can contribute concurrently, it is possible that at any given time there may be several potentially partial replicas of the document in the system. Therefore, it was necessary not only to address the problems related to the coherence of views in order to ensure the feasibility of synchronisation/merge, and to ensure the system's convergence towards a coherent end-state, but also to provide algorithms for the merging of partial replicas. Badouel and Tchoupé did this brilliantly, making their work a solid foundation for modelling workflow systems.

We adapted Badouel and Tchoupé's document model to define an artifact model. Like them, we have therefore chosen to use grammars as our basic mathematical instrument. In the same vein, we made the choice to model workflow systems in which actors have potentially only a partial perception of the processes they execute. We believe that this configuration is relevant to many administrative business processes. For example, in a peer-review process, a reviewer does not necessarily need to know if another reviewer has been contacted for the expertise of the article entrusted to him; and even if so, he should not necessarily know if the latter has already returned his report, etc. Similarly, when organising a journey for a Head of State, not all actors (secret services, civil office, doctor, presidential guard, etc.) have access to the same information which may include for example, tasks to be executed, their dates and states of execution, etc.

A Critical Analysis of the Performed Work

Chronologically, we started by better understanding Badouel and Tchoupé's model in order to extend it so that it takes into account, conflicts detection and resolution. We then embarked on the construction of a workflow system based on this model by first proposing a generic architecture of such systems and an experimental system prototype based on it. We finished by proposing an artifact-centric framework for the completely decentralised management of administrative business processes. The results we can claim are the following:

An algorithm for reconciling partial replicas of a structured document: we gave a solution to the conflicts that arose when merging partial replicas of a structured document by developing reconciliation and control techniques adapted to the modelling of Badouel and Tchoupé. To address this problem, we have expressed it as that of the consensual merging of k updated partial replica $(t_{\mathcal{V}_i}^{maj})_{1 \leq i \leq k}$ (according to k views $(\mathcal{V}_i)_{1 \leq i \leq k}$) whose global model is given by a grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, A)$, which consists in: finding the largest documents $t_{\mathcal{S}}^{maj}$ conforming to \mathbb{G} such that, for any document t conforming to \mathbb{G} and admitting $t_{\mathcal{V}_i}^{maj}$ as projection along the view \mathcal{V}_i , $t_{\mathcal{S}}^{maj}$ and t are eventually updates each for other. The solution we have proposed is as follows: (1) We associate a tree automaton with *exit states* $\mathcal{A}^{(i)}$ for each update $t_{\mathcal{V}_i}^{maj}$ of a partial replica $t_{\mathcal{V}_i}$; this automaton recognises the trees (conform to the global model) for which $t_{\mathcal{V}_i}^{maj}$ is a projection. (2) We perform a *synchronous product* of the automata $\mathcal{A}^{(i)}$ with a commutative and associative operator noted \otimes that we define to obtain the consensual automaton $\mathcal{A}_{(sc)}$ generating the consensus documents: $\mathcal{A}_{(sc)} = \otimes \mathcal{A}^{(i)}$. (3) We obtain the consensus documents by generating the set of trees accepted by the automaton $\mathcal{A}_{(sc)}$.

A software architecture for the implementation of workflow systems: we proposed a generic architecture that could facilitate the implementation of workflow systems as modelled by Badouel and Tchoupé. The proposed architecture is composed of three tiers: *clients*, a *central server* and *administration tools*. These three tiers are interconnected around a middleware that facilitates service-oriented interfacing between them.

TinyCE v2: based on the proposed system architecture, we have built a workflow system prototype referred to as TinyCE v2. The latter was coded in Java and Haskell following a cross-fertilisation protocol that we presented in this manuscript,

and it allowed us to test all the proposed algorithms related to the reconciliation of documents' partial replicas.

A grammar-based language for the artifact-centric modelling of administrative processes: we proposed a new tool (a language) that allows to specify any administrative business process using a triplet $\mathbb{W}_f = (\mathbb{G}, \mathcal{L}_{P_k}, \mathcal{L}_{\mathcal{A}_k})$ composed of: a grammatical model (GMWf) \mathbb{G} , a list of actors \mathcal{L}_{P_k} and a list of accreditations $\mathcal{L}_{\mathcal{A}_k}$. The GMWf is an attributed grammar used to describe all the tasks (by means of its symbols or sorts) of the studied process and the precedence of execution between them (by means of its productions); it is used as artifact type. The list of accreditations provides information on the role played by each actor involved in the process execution; it is through accreditations that one is able to model the potentially partial perceptions that different actors have on the processes and their data.

A distributed workflow system and a fully decentralised execution model of administrative business processes: we proposed a multiagent-like distributed system in which autonomous software agents, based on a same and unique architecture presented in this manuscript, can exchange artifacts to communicate through service invocations, so as to orchestrate the fully decentralised execution of a given administrative business process instance modelled using the proposed grammar-based language. Each time a given agent receives an artifact, it executes a unique five-step protocol allowing it to identify tasks that are ready to be executed on its site, to allow their execution by the local actor and to diffuse if necessary, the updated artifact. To ensure the successful completion of this execution model on a given process case, the initial configuration of agents must be such as to guarantee the coherence of their respective accreditations (views). We have therefore proposed projection algorithms to derive agent-specific models that allow them to control their actions in order to ensure the system's convergence towards a business goal state. We have also investigated some mathematical properties of these algorithms.

P2PTinyWfMS: we have finally produced a prototype of a distributed system providing an artifact-centric management of administrative workflows according to our models. In this one, we have implemented all our algorithms in Java language and we have tested them on some process examples with convincing results.

The work we have done and presented in this manuscript is not perfect: neither in the applied methodology, nor in its presentation, and even less in its scientific contributions. The first criticisms that we can make of it are the following:

The diversity of our algorithms' presentation formats: we didn't just use pseudo code to present our algorithms. Sometimes we used code (Haskell and Java) and other times we wrote them as an arbitrary and ordered set of instructions written in natural language; in these cases we have neglected the more frequently used and more precise mathematical notations. This methodological choice of presentation can indeed be confusing for the reader. We justify it, however, by our desire to be precise, concise and as simple to understand as possible. We have written each of our algorithms in all the formats used in this manuscript before selecting for each of them, the format that seemed to us the clearest and simplest to present and understand.

Conflict management: we have chosen to use a single conflict resolution strategy: that of rejecting conflicting contributions and asking for new ones from contributors. This seems to us rather restrictive but it was a necessary (not necessarily wise) choice for a complete automation of the process of merging partial replicas of a structured document. However, in practice, it would be more appropriate to propose, following the example of Git, several conflict management strategies using a participant as an actor (coordinator) of this resolution.

Insistence on manipulating user views: although we have already justified the choice to take user views into consideration in our models by explaining their contribution to both security and accuracy, it is no less true that they do not always have a positive impact on our work. They have made it a little more complex and have led us to make slightly restrictive assumptions such as the non-recursive GWMf assumption that guarantees their projection. We could make the use of these views more flexible by restricting it to only censorship and not to the complete deletion of sensitive data; this would certainly allow us to overcome some assumptions.

The weak study of our execution model's properties: apart from the isolated study of the properties of a few of its algorithms, we did not study some properties of our decentralised workflow execution model as it is usual in similar BPM studies. We probably pay the consequences of our not completely formal and uncommon (but specific to artifact-centric models, it is one of their often mentioned

limits) presentation that mixes the artifact modelling with its execution. A clear separation of these two aspects would certainly allow us to better study them in an isolated and more conventional way.

Still as theoretical as ever: obviously, we have the ambition to produce concrete systems that can be used in production environments. We are still a long way from that. For the two types of systems studied in this thesis, we have only produced prototypes that allow us to provide experimental proof of concepts. Theoretical studies on the Badouel and Tchoupé model being already quite advanced, it would be time to start implementing these concrete environments.

Some Perspectives

The perspectives presented here are classified into categories according to their priority, to better guide the reader wishing to continue the work. The categories includes :

- **Short-term:** to indicate that the perspective is almost unavoidable and its results will be a real plus to the overall vision we have; it is therefore necessary to work on it as soon as possible;
- **Mid-term:** to indicate that the perspective's results will be practical and usable but not a necessity;
- **Long-term:** to indicate that the perspective is optional and its theoretical results would only help give credibility to our work.

Here are now some interesting avenues for the continuation of our work that come to mind:

A Language for the Specification of Administrative Workflow based on Attributed Grammars (LSAWfP) (priority: short-term): it is obvious that process modelling is a crucial phase of BPM (Dumas et al., 2018). Despite the many efforts made in producing process modelling tools, existing tools (languages) are not commonly accepted. They are mainly criticised for their inability to specify both the tasks making up the processes and their scheduling (their lifecycle models), the data they manipulate (their information models) and their organisational models. Process modelling in these languages often results in a single task graph; such a graph can quickly become difficult to read and maintain. Moreover, these

languages are often too general (they have a very high expressiveness); this makes their application to specific types of processes complex: especially for administrative processes. One can generalise the artifact specification model presented in this thesis, in order to provide a new language for administrative processes modelling that allows designers to specify the lifecycle, information and organisational models of such processes using a mathematical tool based on a variant of attributed grammars. Therefore, the approach imposed by the new language will certainly require the designer to subdivide his process into scenarios, then to model each scenario individually using a simple task graph (an annotated tree) from which a grammatical model will be further derived. At each moment then, the designer will manipulate only a scenario of the studied process: this seems more intuitive and modular because it will allow to produce task graphs that will be more refined and therefore, more readable and easier to maintain.

A Scenario-Oriented Scheme for Administrative Business Processes Modelling (priority: mid-term): in the BPM community, researchers and professionals in the field have little interest in the "how" to model business processes to the benefit of the "with what" to model them. As a result, there is a plethora of workflow modelling languages but very few methods (Dumas et al., 2018). The question on the method to be used to successfully model a given process is however crucial when we know that BPM reduces the automation of the said process to its specification in a particular workflow language: a well carried-out specification produces a quality workflow system. Because the modelling language that can be extracted from the work of this thesis seems to be adapted to a process modelling philosophy centered on the notion of scenario, it would be interesting to propose a method that would accompany it. This one would present the steps to be followed to succeed in its scenario-oriented modelling of administrative workflow systems.

Verification of workflows specified using our models (priority: short-term): one of the BPM activities is the formal analysis/verification of the specifications produced using a given workflow language. Proposing and/or adapting a verification method for workflows designed using our models seems to be an interesting avenue of research especially since several similar works have done the same (Badouel et al., 2015; Van Der Aalst, 1997, 2000). To this end, it will certainly be necessary to deeply study our models in order to highlight their mathematical properties. These properties will then make it possible to identify and present the

criteria that must be verified by a specification in order to be qualified as correct (sound).

A tool to help in the specification of administrative processes with a scenario-oriented approach (priority: short-term): since the models we have proposed are new, it would be wise to propose a tool to assist in their use in practice. In addition to being a guide, such a tool should simplify the creation of process models and possibly validate them according to pre-established correction criteria. Moreover, it will be able to provide several DSL for saving the specifications as well as several modules for exporting them in more conventional notations (BPMN, YAWL, etc.).

A framework to generate administrative business processes' specific decentralised execution simulators (priority: mid-term): as implemented, the WfMS prototype P2PTinyWfMS can be used as a foundation for the production of a tool that generates simulators of the completely decentralised execution of administrative processes specified using our models. The new framework can be based on the models found in (Tchembé, Tchoupé Tchendji, and Matene Kakeu, 2019); then, it will be implemented to generate a simulation environment tailored to a given administrative process. Still in a generative logic, another approach would be to use the recent concepts of model-driven engineering to produce a simulator based on integrated development environments such as Eclipse: one could for example use a GEMOC approach (Bousse et al., 2016; Combemale, Barais, and Wortmann, 2017).

Extension of our decentralised execution model for recursive GMWf and monitoring support (priority: short-term): we already mentioned this in the discussion section of chapter III. It would be interesting to introduce into our multiagent system, new types of agents to monitor the execution of processes. In addition, we could also make the use of views more flexible by modifying their impact on projection operations. For example, we could redefine the artifact projection operation so that it no longer erases nodes but censors them; that is, it replaces them with symbols that help in the specification of control flows (restructuring symbols for instance) carrying no information on the process. This will preserve the possibility of offering only a potentially partial view of the processes and their data to actors while allowing the use of recursive symbols in GMWf.

Concrete implementation of WfMS supporting our models (priority: short-term): eventually, it will also be necessary to propose implementations of the WfMS presented here. Naturally, the implemented system will have to cover all phases and activities of the BPM lifecycle while focusing on the automation of administrative business processes. A study of recent BPM systems to ensure interoperability and an openness of the implemented system on the cloud will certainly be interesting avenues to explore.

Study of each proposed formal tool's properties in order to identify more precisely the class of workflows to which they apply (priority: long-term): another, more theoretical, line of research would be to formally analyse the proposed BPM approach to characterise the class of workflows to which it can actually be applied. We assumed that we were only interested in administrative workflows; however, the proposed models are quite general and could well be applied to other classes of workflows. The study carried out in this perspective should therefore determine these classes and the conditions under which the proposed model automates them.

BIBLIOGRAPHY

- Almeida, E. T., Luntz, J. E., and Tilbury, D. M. (2005). Modular finite state machines implemented as event-condition-action systems. *IFAC Proceedings Volumes*, 38(1), 373–378.
- Aoyama, M. (1998). New age of software development: How component-based software engineering changes the way of software development. In *1998 international workshop on cbse* (pp. 1–5).
- Assaf, M. A. (2016). Towards an integration system for artifact-centric processes. In *Proceedings of the 2016 on sigmod'16 phd symposium* (pp. 2–6).
- Assaf, M. A., Badr, Y., and Amghar, Y. (2017). A continuous query language for stream-based artifacts. In *International conference on database and expert systems applications* (pp. 80–89).
- Assaf, M. A., Badr, Y., El Khoury, H., and Barbar, K. (2018). Generating database schemas from business artifact models. *I.J. Information Technology and Computer Science*, 2, 10–17. doi: 10.5815/ijitcs.2018.02.02
- Badouel, E., Hélouët, L., Kouamou, G.-E., and Morvan, C. (2014). A Grammatical Approach to Data-centric Case Management in a Distributed Collaborative Environment. *CoRR*, abs/1405.3223. Retrieved from <http://arxiv.org/abs/1405.3223>
- Badouel, E., Hélouët, L., Kouamou, G.-E., Morvan, C., and Fondze Jr, N. R. (2015). Active workspaces: distributed collaborative systems based on guarded attribute grammars. *ACM SIGAPP Applied Computing Review*, 15(3), 6–34.
- Badouel, E., and Tchoupé Tchendji, M. (2008). Merging Hierarchically-Structured Documents in Workflow Systems. *Electronic Notes in Theoretical Computer Science*, 203(5), 3–24.
- Baier, C., and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

- Balasubramaniam, S., and Pierce, B. C. (1998). What is a file synchronizer? In *Proceedings of the 4th annual acm/ieee international conference on mobile computing and networking* (pp. 98–108).
- Barais, O. (2005). *Construire et maîtriser l'évolution d'une architecture logicielle à base de composants* (PhD Thesis, Lille 1). Retrieved from <http://www.theses.fr/2005LIL10072> (Thèse de doctorat dirigée par Duchien, Laurence Informatique Lille 1 2005)
- Bell, G., Hey, T., and Szalay, A. (2009). Beyond the data deluge. *Science*, 323(5919), 1297–1298.
- Berliner, B., et al. (1990). Cvs ii: Parallelizing software development. In *Proceedings of the usenix winter 1990 technical conference* (Vol. 341, p. 352).
- Berstel, J., and Boasson, L. (2000). Xml grammars. In *International symposium on mathematical foundations of computer science* (pp. 182–191).
- Boaz, D., Limonad, L., and Gupta, M. (2013). *Bizartifact: Artifact-centric business process management, june 2013*. Retrieved from <https://sourceforge.net/projects/bizartifact/>, accessed 12 December 2019
- Böhme, T., and Rahm, E. (2004). Supporting efficient streaming and insertion of xml data in rdbms. In *Diweb* (pp. 70–81).
- Booch, G., Rumbaugh, J., Jacobson, I., et al. (2000). *Le guide de l'utilisateur uml* (Vol. 3). Eyrolles.
- Börger, E. (2012). Approaches to modeling business processes: a critical analysis of bpmn, workflow patterns and yawl. *Software & Systems Modeling*, 11(3), 305–318.
- Boukhedouma, S. (2015). *Adaptation et restructuration de modèles de processus workflow dans un contexte inter-organisationnel* (PhD Thesis). Université de Nantes.
- Bousse, E., Degueule, T., Vojtisek, D., Mayerhofer, T., Deantoni, J., and Combe-male, B. (2016). Execution framework of the gemoc studio (tool demo). In *Proceedings of the 2016 acm sigplan international conference on software language engineering* (pp. 84–89).
- Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., et al. (2000). *Extensible markup language (xml) 1.0*. W3C recommendation October.

- Brown, C. (2005). Workflow Management in Staffware. In *Process-aware information systems: Bridging people and software through process technology*. Wiley.
- Bryant, B. R., Gray, J., and Mernik, M. (2010). Domain-specific software engineering. In *Proceedings of the fse/sdp workshop on future of software engineering research* (pp. 65–68).
- Burkhart, T., Werth, D., and Loos, P. (2012). Context-sensitive business process support based on emails. In *Proceedings of the 21st international conference on world wide web* (pp. 851–856).
- Chaâbane, M. A., Bouzguenda, L., Bouaziz, R., and Gargouri, F. (2007). Spécification des Processus Workflows Évolutifs Versionnés. *Schedae, prépublication*, 2(11), 21–29.
- Combemale, B., Barais, O., and Wortmann, A. (2017). Language engineering with the gemoc studio. In *2017 ieee international conference on software architecture workshops (icsaw)* (pp. 189–191).
- Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (1997). Tree automata techniques and applications. <http://www.grappa.univ-lille3.fr/tata>.
- Cunningham, W. (1995). *Wikiwikiweb history*. Retrieved from <http://c2.com/cgi/wiki?WikiHistory>
- Dadam, P., Reichert, M., and Kuhn, K. (2000). Clinical workflows - the killer application for process-oriented information systems? In *Bis 2000* (pp. 36–59). Springer.
- Damaggio, E., Deutsch, A., and Vianu, V. (2012). Artifact systems with data dependencies and arithmetic. *ACM Transactions on Database Systems (TODS)*, 37(3), 1–36.
- Damaggio, E., Hull, R., and Vaculín, R. (2013). On the equivalence of incremental and fixpoint semantics for business artifacts with guard–stage–milestone lifecycles. *Information Systems*, 38(4), 561–584.
- Das, S., Kochut, K., Miller, J. A., Sheth, A. P., and Worah, D. (1997). Orbwork: A reliable distributed corba-based workflow enactment system for meteor2. *University of Georgia*.
- Davie, A. J. (1992). *Introduction to functional programming systems using haskell*

(Vol. 27). Cambridge University Press.

DeLoach, S. A., Wood, M. F., and Sparkman, C. H. (2001). Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(03), 231–258.

Deutsch, A., Hull, R., and Vianu, V. (2014). Automatic verification of database-centric systems. *ACM SIGMOD Record*, 43(3), 5–17.

Divitini, M., Hanachi, C., and Sibertin-Blanc, C. (2001). Inter-organizational workflows for enterprise coordination. In *Coordination of internet agents* (pp. 369–398). Springer.

Dumas, M. (2015). From models to data and back: the journey of the bpm discipline and the tangled road to bpm 2020. In *Proceedings of the 13th international conference on business process management*. Springer.

Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2018). *Fundamentals of business process management, second edition*. Springer.

Dumas, M., Van Der Aalst, W. M. P., and Ter Hofstede, A. H. M. (2005). *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons.

Eder, J., and Gruber, W. (2002). A meta model for structured workflows supporting workflow transformations. In *East european conference on advances in databases and information systems* (pp. 326–339).

Ellis, C. A., Gibbs, S. J., and Rein, G. (1991). Groupware: some issues and experiences. *Communications of the ACM*, 34(1), 39–58.

Estanol, M., Queralt, A., Sancho, M. R., and Teniente, E. (2012). Artifact-centric business process models in uml. In *International conference on business process management* (pp. 292–303).

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2), 114–131.

Fakas, J. G., and Karakostas, B. (2004). A Peer to Peer (P2P) Architecture for Dynamic Workflow Management. *Information & Software Technology*, 46(6), 423–431.

Ferber, J. (1997). *Les systèmes multi-agents: vers une intelligence collective*.

InterEditions.

Gazzé, D., La Polla, M. N., Marchetti, A., Tesconi, M., and Vivaldi, A. (2012). Workmail: collaborative document workflow management by email. In *International conference on cooperative design, visualization and engineering* (pp. 14–23).

Georgakopoulos, D., F. Hornick, M., and P. Sheth, A. (1995). An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2), 119–153.

Geppert, A., and Tombros, D. (1998). Event-Based Distributed Workflow Execution with EVE. In *Middleware'98* (pp. 427–442).

Giannetti, J., and Lord, M. (2015). Une plateforme web pour soutenir la réécriture collaborative: Etherpad. *Formation et profession*, 23(1), 71–73.

Grigori, D. (2001). *Éléments de flexibilité des systèmes de workflow pour la définition et l'exécution de procédés coopératifs* (PhD Thesis). Université Henri Poincaré - Nancy 1.

Grudin, J. (1994). Computer-supported cooperative work: History and focus. *Computer*, 27(5), 19–26.

Hauder, M., Pigat, S., and Matthes, F. (2014). Research challenges in adaptive case management: a literature review. In *2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations* (pp. 98–107).

Heineman, G. T., and Councill, W. T. (2001). *Component-based software engineering: Putting the pieces together*. Ivica Crnkovic, George T. Heineman.

Houlding, D. I. (2004, May). *System and method for delivering web services using common object request broker architecture*. Google Patents. (US Patent 6,735,771)

Hull, R. (2017). Blockchain: Distributed event-based processing in a data-centric world. In *Proceedings of the 11th ACM international conference on distributed and event-based systems* (pp. 2–4).

Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F. T., ... others (2011). Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In *Proceedings of the 5th*

acm international conference on distributed event-based system (pp. 51–62).

Hull, R., Narendra, N. C., and Nigam, A. (2009). Facilitating workflow inter-operation using artifact-centric hubs. In *Service-oriented computing* (pp. 1–18). Springer.

Hull, R., Su, J., and Vaculin, R. (2013). Data management perspectives on business process management: tutorial overview. In *Proceedings of the 2013 acm sigmod international conference on management of data* (pp. 943–948).

Hurwitz, J. S., Bloor, R., Kaufman, M., and Halper, F. (2009). *Service oriented architecture (soa) for dummies*. John Wiley & Sons.

IMA, I. o. M. A. (2002). *Implementing Automated Workflow Management*. Institute of Management Accountants.

Imine, A. (2006). *Conception Formelle d'Algorithmes de Réplication Optimiste vers l'Édition Collaborative dans les Réseaux Pair-à-Pair* (PhD Thesis). Université Henri Poincaré - Nancy 1.

Jennings, N. R. (1996). Using Intelligent Agents to Manage Business Processes. In *Intelligent agents and their applications, iee colloquium on (digest no: 1996/101)* (pp. 5–1).

Johansen, R. (1988). *Groupware: Computer support for business teams*. The Free Press.

Juve, G., Chervenak, A. L., Deelman, E., Bharathi, S., Mehta, G., and Vahi, K. (2013). Characterizing and Profiling Scientific Workflows. *Future Generation Comp. Syst.*, 29(3), 682–692.

Kanzow, S. (2004). *Approche pour l'Ordonnancement Distribué de Workflows dans le Contexte d'Entreprises Virtuelles: une Méthodologie basée Multi-Agents pour la Planification et l'Éxecution de Processus Distribués* (PhD Thesis). Université Paris XII - Val de Marne.

Kaur, N., McLeod, C. S., Jain, A., Harrison, R., Ahmad, B., Colombo, A. W., and Delsing, J. (2013). Design and simulation of a soa-based system of systems for automation in the residential sector. In *2013 iee international conference on industrial technology (icit)* (pp. 1976–1981).

Kiepuszewski, B., Ter Hofstede, A. H. M., and Bussler, C. J. (2000). On structured workflow modelling. In *International conference on advanced information systems*

engineering (pp. 431–445).

Krishnakumar, N., and Sheth, A. (1995). Managing heterogeneous multi-system tasks to support enterprise-wide operations. *Distributed and Parallel Databases*, 3(2), 155–186.

Krishnan, R., Munaga, L., and Karlapalem, K. (2001). Xdoc-wfms: A framework for document centric workflow management system. In *International conference on conceptual modeling* (pp. 348–362).

Lahcen, A. A., and Parigot, D. (2012). A Lightweight Middleware for Developing P2P Applications with Component and Service-Based Principles. In *15th IEEE international conference on computational science and engineering, CSE 2012, paphos, cyprus, december 5-7, 2012* (pp. 9–16).

Lamma, E., Riguzzi, F., Storari, S., Mello, P., and Montali, M. (2007). Learning decserflow models from labeled traces. Oregon State University.

Leymann, F. (2010). Bpel vs. bpmn 2.0: Should you care? In *International workshop on business process modeling notation* (pp. 8–13).

Leymann, F., and Roller, D. (1994). Business Process Management with Flowmark. In *Spring COMPCON 94, digest of papers, san francisco, california, usa, february 28 - march 4, 1994* (pp. 230–234).

Liu, R., and Kumar, A. (2005). An analysis and taxonomy of unstructured workflows. In *International conference on business process management* (pp. 268–284).

Lohmann, N., and Nyolt, M. (2011). Artifact-centric modeling using bpmn. In *International conference on service-oriented computing* (pp. 54–65).

Lohmann, N., and Wolf, K. (2010). Artifact-centric choreographies. In *International conference on service-oriented computing* (pp. 32–46).

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., . . . Zhao, Y. (2006). Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10), 1039–1065.

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., Metz, R., and Hamilton, B. A. (2006). Reference model for service oriented architecture 1.0. *OASIS standard*, 12(S 18).

Manna, Z., and Pnueli, A. (2012). *Temporal verification of reactive systems: safety*.

Springer Science & Business Media.

Marchetti, A., Tesconi, M., and Minutoli, S. (2005). Xflow: An xml-based document-centric workflow. In *International conference on web information systems engineering* (pp. 290–303).

McCready, S. (1992). There is more than one Kind of Workflow Software. *Computerworld*, 2.

Medina-Mora, R., Winograd, T., Flores, R., and Flores, F. (1993). The Action Workflow Approach to Workflow Management Technology. *The Information Society*, 9(4), 391–404.

Meilin, S., Guangxin, Y., Yong, X., and Shangguang, W. (1998, October). Workflow management systems: a survey. In *Icct'98. 1998 international conference on communication technology. proceedings (iee cat. no. 98ex243)* (Vol. 2, pp. 6–12).

Mending, J., Weber, I., Van Der Aalst, W. M. P., Brocke, J. v., Cabanillas, C., Daniel, F., ... Zhu, L. (2018). Blockchains for business process management - challenges and opportunities. *ACM Trans. Management Inf. Syst.*, 9(1), 4:1–4:16.

Mens, T. (2002). A State-of-the-Art Survey on Software Merging. *Journal of IEEE Transactions on Software Engineering*, 28(5), 449–462.

Meyer, B. (2000). *Conception et programmation orientées objet*. Eyrolles.

Miao, Y., Zhou, J., and Tao, L. (2011). Realization of key technologies for enterprise collaboration office automation system based on workflow. *Computer Technology and Development*, 21(3), 90–93.

Miller, J. A., Palaniswami, D., Sheth, A. P., Kochut, K., and Singh, H. (1998). WebWork: METEOR₂'s Web-Based Workflow Management System. *J. Intell. Inf. Syst.*, 10(2), 185–215.

Model, B. P. (2011). Notation (BPMN) version 2.0. *OMG Specification, Object Management Group*, 22–31.

Nigam, A., and Caswell, N. S. (2003). Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3), 428–445.

Oster, G. (2005). *Réplication optimiste et cohérence des données dans les environnements collaboratifs répartis* (PhD Thesis). Université Henri Poincaré - Nancy I.

Ouyang, C., Dumas, M., Ter Hofstede, A. H. M., and Van Der Aalst, W. M. P. (2006). From bpmn process models to bpel web services. In *2006 IEEE International Conference on Web Services (ICWS'06)* (pp. 285–292).

Papazoglou, M. P. (2003). Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the fourth international conference on web information systems engineering, 2003. wise 2003.* (pp. 3–12).

Penichet, V. M. R., Marin, I., Gallud, J. A., Lozano, M. D., and Tesoriero, R. (2007). A classification method for cscw systems. *Electronic Notes in Theoretical Computer Science*, 168, 237–247.

Piccinelli, G., Finkelstein, A., and Williams, S. L. (2003). Service-oriented workflow: The dysco framework. In *null* (p. 291).

Pilato, C. M., Collins-Sussman, B., and Fitzpatrick, B. W. (2008). *Version control with subversion: next generation open source version control.* O'Reilly Media, Inc.

Reichert, M., and Dadam, P. (1998). Adept flex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2), 93–129.

Reichert, M., Rinderle, S., and Dadam, P. (2003). Adept workflow management system. In *International conference on business process management* (pp. 370–379).

Rowland, F. (2002). The Peer-Review Process. *Learned Publishing*, 15(4), 247–258.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. E., et al. (1991). *Object-oriented modeling and design* (Vol. 199). Prentice-hall Englewood Cliffs, NJ.

Saito, Y., and Shapiro, M. (2005). Optimistic replication. *ACM Computing Surveys (CSUR)*, 37(1), 42–81.

Sarin, S. K. (1996). Workflow and Data Management in InConcert. In *Proceedings of the twelfth international conference on data engineering, february 26 - march 1, 1996, new orleans, louisiana, USA* (pp. 497–499).

Schimdt, K., and Bannon, L. (1992). Taking cscw seriously: Supporting articulation work. *Computer Supported Cooperative Work (CSCW)*, 1(1).

Sun, Y., Xu, W., and Su, J. (2012). Declarative choreographies for artifacts. In *International conference on service-oriented computing* (pp. 420–434).

Sycara, K. P. (1998). Multiagent systems. *AI magazine*, 19(2), 79–79.

Tchembé, M. X., Tchoupé Tchendji, M., and Matene Kakeu, A. L. (2019, December). An ad-hoc social network generation approach. In *CRI'2019 - Conférence de Recherche en Informatique*. Yaoundé, Cameroon.

Tchoupé Tchendji, M. (2009). *Une Approche Grammaticale pour la Fusion des Réplicats Partiels d'un Document Structuré: Application à l'Édition Coopérative Asynchrone* (PhD Thesis). Université de Rennes I (France), Université de Yaoundé I (Cameroun).

Tchoupé Tchendji, M. (2010). Fertilisation croisée d'un langage fonctionnel et d'un langage objet: application à la mise en oeuvre d'un prototype d'éditeur coopératif asynchrone. In *Cari 2010 proceedings* (Vol. 1, pp. 541–549).

Tchoupé Tchendji, M., Djeumen D., R., and Atemkeng T., M. (2017). A Stable and Consistent Document Model Suitable for Asynchronous Cooperative Edition. *Journal of Computer and Communications*, 5(08), 69.

Tchoupé Tchendji, M., and Zekeng Ndadji, M. M. (2016). Réconciliation par consensus des mises à jour des répliques partielles d'un document structuré. In *Cari 2016 proceedings* (Vol. 1, pp. 84–96).

Tchoupé Tchendji, M., and Zekeng Ndadji, M. M. (2017). Tree Automata for Extracting Consensus from Partial Replicas of a Structured Document. *Journal of Software Engineering and Applications*, 10(05), 432–456.

Tlili, M. (2011). *Infrastructure p2p pour la réplication et la réconciliation des données* (PhD Thesis). Université de Nantes.

Van Der Aalst, W. M. P. (1997). Verification of workflow nets. In *International conference on application and theory of petri nets* (pp. 407–426).

Van Der Aalst, W. M. P. (1998). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(01), 21–66.

Van Der Aalst, W. M. P. (2000). Workflow verification: Finding control-flow errors using petri-net-based techniques. In *Business process management* (pp. 161–183). Springer.

Van Der Aalst, W. M. P. (2011). *Process mining: discovery, conformance and*

enhancement of business processes. Springer.

Van Der Aalst, W. M. P. (2013). Business process management: a comprehensive survey. *ISRN Software Engineering*, 2013.

Van Der Aalst, W. M. P. (2015). Business process management as the "killer ap" for petri nets. *Software & Systems Modeling*, 14(2), 685–691.

Van Der Aalst, W. M. P., Barthelmess, P., Ellis, C. A., and Wainer, J. (2001). Procllets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, 10(04), 443–481.

Van Der Aalst, W. M. P., Jörg, D., and Oberweis, A. (2003). *Business process management: Models, techniques, and empirical studies*. Springer.

Van Der Aalst, W. M. P., La Rosa, M., and Santoro, F. M. (2016). Business process management: Don't forget to improve the process ! *Business & Information Systems Engineering*, 58(1), 1–6.

Van Der Aalst, W. M. P., Mans, R., and Russell, N. C. (2009). Workflow support using procllets: Divide, interact, and conquer. *IEEE Data Eng. Bull.*, 32(3), 16–22.

Van Der Aalst, W. M. P., and Pesic, M. (2006). Decserflow: Towards a truly declarative service flow language. In *International workshop on web services and formal methods* (pp. 1–23).

Van Der Aalst, W. M. P., and Ter Hofstede, A. H. M. (2005). Yawl: yet another workflow language. *Information systems*, 30(4), 245–275.

Van Der Aalst, W. M. P., and Ter Hofstede, A. H. M. (2012). Workflow patterns put into context. *Software & Systems Modeling*, 11(3), 319–323.

WfMC. (1995). *Wfmc Standards: the Workflow Reference Model, Version 1.1*. <http://www.aiim.org/wfmc/mainframe.htm>. Retrieved from <http://www.aiim.org/wfmc/mainframe.htm>

White, S. (2005). Using bpmn to model a bpel process. *BPTrends*, 3(3), 1–18.

Wilm, J., and Frebel, D. (2014). Real-world challenges to collaborative text creation. In *Proceedings of the 2nd international workshop on (document) changes: modeling, detection, storage and visualization* (pp. 1–4).

Wu, L., and Buyya, R. (2012). Service level agreement (sla) in utility computing systems. In *Performance and dependability in service computing: Concepts*,

techniques and research directions (pp. 1–25). IGI Global.

Yan, J., Yang, Y., and Raikundalia, G. K. (2006). SwinDeW-a P2P-Based Decentralized Workflow Management System. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 36(5), 922–935.

Yongyi, Z., Youjie, F., and Hui, L. (2009). Research on service-oriented workflow management system architecture. In *2009 ninth international conference on hybrid intelligent systems* (Vol. 2, pp. 369–372).

Zekeng Ndadji, M. M., and Tchoupé Tchendji, M. (2018). A Software Architecture for Centralized Management of Structured Documents in a Cooperative Editing Workflow. In *Innovation and interdisciplinary solutions for underserved areas* (pp. 279–291). Springer.

Zur Muehlen, M., and Recker, J. (2013). How much language is enough? theoretical and practical use of the business process modeling notation. In *Seminal contributions to information systems engineering* (pp. 429–443). Springer.

A

APPENDIX

IMPLEMENTATION OF SOME IMPORTANT ALGORITHMS PRESENTED IN THIS THESIS

We had thought to present in this appendix, a Haskell implementation of the projection algorithms proposed in chapter III of this thesis. However, these are far too voluminous and their presentation here will not be very readable. We have therefore decided to present only the main data types here. We have hosted the rest of the produced Haskell code on the public Git repository accessible via this link: <https://github.com/MegaMaxim10/my-thesis-projection-algorithms>.

Haskell Type for Tags

Let's start by defining the tags for the node types (sequential or parallel). More clearly, in a given artifact, a node A is tagged with `Seq` (resp. `Par`) when its sub-artifacts are executed in sequence (resp. potentially in parallel), i.e. the production used for its extension is a sequential (resp. parallel) one. A node with at most one sub-artifact is always tagged with `Seq`.

```
1 data ProductionTag x = Seq x | Par x deriving (Eq, Show)
```

The `untagProduction` function below clears a given symbol of its tag (`Seq` or `Par`):

```
1 untagProduction :: ProductionTag x -> x
2 untagProduction (Seq x) = x
3 untagProduction (Par x) = x
```

Definition of tags (*closed*, *locked*, *unlocked* or *upstair*) for symbols

In an artifact: a closed node is tagged `Closed`, an unlocked bud is tagged `Unlocked`, a locked bud is tagged `Locked` and an upstairs bud is tagged `Upstair` (only found after expansion).

```
1 data NodeTag x = Closed x | Locked x | Unlocked x | Upstair x deriving (Eq, Show)
```

The `untagNode` function below clears a given symbol of its tag (`Closed`, `Unlocked`, `Locked` or `Upstair`):

```
1 untagNode :: NodeTag x -> x
2 untagNode (Closed x) = x
3 untagNode (Locked x) = x
4 untagNode (Unlocked x) = x
5 untagNode (Upstair x) = x
```

Definition of tags for symbol types (structuring or standard)

The symbols of a given artifact t are either those of the grammatical model \mathbb{G} denoting t , or (re)structuring symbols introduced to preserve some important properties of our model (mainly, the form of productions used in GMWf): in this case, the symbols of \mathbb{G} are said to be standard and are tagged with `Standard` while the (re)structuring symbols are tagged with `Structural`.

```
1 data SymbolTag x = Structural x | Standard x deriving (Eq, Show)
```

As the previous "untag" functions, the `untagSymbol` function below clears a given symbol of its tag (`Structural` or `Standard`):

```
1 untagSymbol :: SymbolTag x -> x
2 untagSymbol (Structural x) = x
3 untagSymbol (Standard x) = x
```

Haskell Type for Artifacts

Recursively, we consider that an artifact is given by its root node (`nodeLabel`) and the list of its sub-artifacts (`sonsList`) tagged either by `Seq` (to indicate that

they are executed in sequence) or by `Par` (to indicate that they are potentially executed in parallel). We do not consider empty artifacts. The corresponding Haskell type is as follows:

```

1 data Artifact a = Node {
2     nodeLabel:: a,
3     sonsList:: ProductionTag [Artifact a]
4     } deriving Eq

```

Here is an example of artifact encoded in this type. It corresponds to the target artifact art_1 in the figure 35:

```

1 art1 = Node (Closed "Ag") (
2     Seq [
3         Node (Closed "A") (
4             Seq [
5                 Node (Closed "B") (Seq []),
6                 Node (Closed "D") (Seq [])
7             ])
8     ])

```

Haskell Type for GMWf

Let's start by presenting a type for productions: a production is given by its left hand side (`lhs`) consisting of one symbol and by its right hand side (`rhs`) consisting of several symbols.

```

1 data Production symb = Prod {lhs:: symb, rhs:: [symb]} deriving Eq

```

Finally, a GMWf is given by the set of symbols and the set of productions constituting it. The productions are tagged either by `Seq` or by `Par`:

```

1 data GMWf a = GMWf {
2     symbols:: [a],
3     productions:: [ProductionTag (Production a)]
4     } deriving (Eq, Show)

```

These are the main data types that we have defined and which are manipulated by the different projection functions that are available in our Git repository¹. They are included with some test cases that one will be able to immediately experiment.

1. Our Git repository: <https://github.com/MegaMaxim10/my-thesis-projection-algorithms>



LIST OF PUBLICATIONS ISSUED FROM THE WORK PRESENTED IN THIS THESIS

Journal Papers

Published

1. Milliam Maxime Zekeng Ndadji, Maurice Tchoupé Tchendji, Clémentin Tayou Djamegni and Didier Parigot. "*A new Domain-Specific Language and Methodology based on Scenarios, Grammars and Views, for Administrative Processes Modelling.*" ParadigmPlus, Volume 1, Number 3, 2020, 1-22.
2. Maurice Tchoupé Tchendji and Milliam Maxime Zekeng Ndadji. "*Tree Automata for Extracting Consensus from Partial Replicas of a Structured Document.*" Journal of Software Engineering and Applications 10.05 (2017): 432-456.

Under Review

1. Maurice Tchoupé Tchendji, Milliam Maxime Zekeng Ndadji and Didier Parigot. "*A Grammatical Approach for Administrative Workflow Design and their Distributed Execution using Structured and Cooperatively Edited Mobile Artifacts.*" Software and Systems Modeling, Springer (**submitted**).
2. Milliam Maxime Zekeng Ndadji, Maurice Tchoupé Tchendji, Clémentin Tayou Djamegni and Didier Parigot. "*A Projection-Stable Grammatical Model for the Distributed Execution of Administrative Processes with Emphasis on Actors' Views.*" Journal of King Saud University - Computer and Information Sciences, Elsevier (**submitted**).

Communications in Conferences

Published

1. Milliam Maxime Zekeng Ndadji, Maurice Tchoupé Tchendji, Clémentin Tayou Djamegni and Didier Parigot. "A Grammatical Model for the Specification of Administrative Workflow using Scenario as Modelling Unit." H. Florez and S. Misra (eds) Applied Informatics. ICAI 2020. Communications in Computer and Information Science, vol 1277, Springer, Cham, 2020. pages 131-145.
2. Milliam Maxime Zekeng Ndadji, Maurice Tchoupé Tchendji, Clémentin Tayou Djamegni and Didier Parigot. "A Language for the Specification of Administrative Workflow Processes with Emphasis on Actors' Views." Gervasi O. et al. (eds) Computational Science and Its Applications - ICCSA 2020. ICCSA 2020. Lecture Notes in Computer Science, vol 12254, Springer, Cham, 2020. pages 231-245.
3. Milliam Maxime Zekeng Ndadji, Maurice Tchoupé Tchendji and Didier Parigot. "A Projection-Stable Grammatical Model to Specify Workflows for their P2P and Artifact-Centric Execution." CRI'2019 - Conférence de Recherche en Informatique. Dec 2019, Yaoundé, Cameroon. (hal-02375958).
4. Milliam Maxime Zekeng Ndadji and Maurice Tchoupé Tchendji. "A Software Architecture for Centralized Management of Structured Documents in a Cooperative Editing Workflow." Innovation and Interdisciplinary Solutions for Underserved Areas. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering (LNICST), Springer, Cham, 2018. pages 279-291.
5. Maurice Tchoupé Tchendji and Milliam Maxime Zekeng Ndadji. "Réconciliation par consensus des mises à jour des répliques partielles d'un document structuré." CARI 2016 Proceedings, volume 1, 2016. pages 84-96.