

Universidade do Minho
Escola de Engenharia

Nuno Miguel Almeida Luz

**Ontology-based Representation and
Generation of Workflows for Micro-Task
Human-Machine Computation**

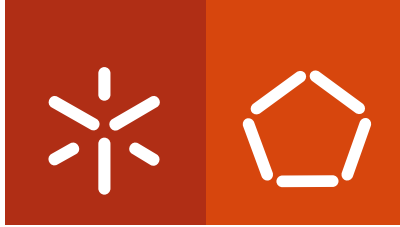
**The MAP Doctoral Program in Computer Science
of the Universities of Minho, Aveiro and Porto**



Universidade do Minho



January 2015



Universidade do Minho

Escola de Engenharia

Nuno Miguel Almeida Luz

Ontology-based Representation and Generation of Workflows for Micro-Task Human-Machine Computation

**The MAP Doctoral Program in Computer Science
of the Universities of Minho, Aveiro and Porto**



Universidade do Minho

Thesis submitted at the University of Minho for the degree of
Doctor of Philosophy (PhD) in Computer Science, under the
supervision of

Paulo Jorge Freitas de Oliveira Novais

and

Nuno Alexandre Pinto da Silva

January 2015

STATEMENT OF INTEGRITY

I hereby declare having conducted my thesis with integrity. I confirm that I have not used plagiarism or any form of falsification of results in the process of the thesis elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, _____

Full name: _____

Signature: _____

The more I read, the more I acquire, the more certain I am that I know nothing.

— Voltaire

ACKNOWLEDGMENTS

This research represents a long process of acquiring new skills and reaching for answers and questions alike in many different ways. Such a journey could never have led to the research presented in this thesis without the gracious support, patience and collaboration of several people. To all the people involved over the last four years, I am sincerely grateful.

I am most thankful to my supervisors Nuno Silva and Paulo Novais for their support and dedication. To Nuno Silva, for his interest and eagerness to question and understand even the smallest details. Our enlightening discussions and his advice and active participation were invaluable factors in the concretization of this thesis. To Paulo Novais for his constant attention to the progress of this research and for always pushing me towards reaching new milestones.

I would also like to thank all the friends that shared this journey with me. To Ricardo Anacleto and Carlos Pereira, for our multiple more and less productive but refreshing discussions, and to Nuno Oliveira for his invaluable advice in subjects I was trying to grasp. Also, to my parents, Belmiro and Glória, for their amazing efforts in always providing the best conditions for me and my brother, even in the most difficult circumstances.

The presented evaluation scenarios would never have been achieved without the participation of several people. Their effort and feedback are an important part of this work.

Finally, I would like to thank Soraia Ferreira, for her unconditional support. There are not enough words of appreciation for her patience, love and understanding during this long academic endeavour.

Nuno Miguel Almeida Luz: *Ontology-based Representation and Generation of Workflows for Micro-Task Human-Machine Computation* © January 2015

This work is funded by the ERDF (European Regional Development Fund) through the COMPETE Programme and by the Portuguese Government through the FCT (Portuguese Foundation for Science and Technology) within the doctoral grant SFRH/BD/70302/2010.

FCT Fundação para a Ciência e a Tecnologia

MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA



RESUMO

A crescente popularidade das plataformas de crowdsourcing de micro-tarefas levou ao aparecimento de novas abordagens baseadas em fluxos e workflows de micro-tarefas. Juntamente com estas novas abordagens, surgem novos desafios. A falta de estruturação dos dados das micro-tarefas torna difícil, por parte de quem solicita as tarefas, a inclusão de participantes máquina no processo de execução dos workflows. Outro desafio deve-se á falta de componentes que permitam o controlo do fluxo em workflows de micro-tarefas, embora estes componentes sejam comuns em abordagens de workflow tradicionais e em processos de negócio.

Nesta tese, é proposto um método para a representação, construção, instanciação e execução de workflows de tarefas em ambientes de computação pessoa-máquina, baseado em ontologias. A representação é capaz de capturar a estrutura e a semântica das operações e dos seus dados, ao mesmo tempo que se mantém próxima do nível conceptual humano. Os workflows são construídos em duas dimensões: a dimensão de domínio estático e a dimensão (da tarefa) dinâmica. Isto permite que os dados de entrada e de saída dos workflows possam ser descritos exclusivamente de acordo com uma ontologia de domínio, de forma completamente independente da representação do workflow. Para que possa ser efetuada a instanciação e a execução da representação do workflow, foi implementado um motor de workflows baseado no método proposto.

Para facilitar o papel do solicitador (ou *requester*) na criação de novas representações de workflows (ou *workflow-definitions*), um processo de construção semi-automático baseado em ontologias de domínio é também proposto. O processo foi implementado numa ferramenta de construção que permite a construção assistida, iterativa e visual de representações de workflows.

O método de representação e o processo de construção propostos são avaliados através de múltiplos cenários de aplicação em diferentes domínios.

ABSTRACT

The growing popularity of micro-task crowdsourcing platforms has led to new approaches based on workflows of micro-tasks. Along with these new approaches, new challenges have emerged. The unstructured nature of micro-tasks in terms of domain representation makes it difficult for task requesters to include machine workers in the workflow execution process. Also, the representation of these human-machine computation workflows lack the flow control components often found in traditional workflow and business process approaches.

In this thesis, a method for the representation, construction, instantiation and execution of human-machine computation task workflows through ontologies is proposed. The representation captures the structure and semantics of the tasks and their domain, while remaining close to the human conceptual level. Workflows are built according to two dimensions: the static domain dimension and the dynamic (task) dimension. This allows the input and the output of workflows to be described according to a domain ontology, completely independent from the workflow representation. The instantiation and execution of the represented workflow can be performed through the implemented workflow engine.

To aid the requester in the creation of new workflow representations (or workflow-definitions), a semi-automatic construction process based on domain ontologies is also proposed. The process has been implemented into a construction framework that allows the aided, iterative and visual construction of workflow-definitions.

The proposed method and construction process is evaluated through several application scenarios in different domains.

CONTENTS

i	PREAMBLE	1
1	INTRODUCTION	3
1.1	Problem Statement and Motivations	3
1.2	Hypothesis and Objectives	5
1.3	Approach	8
1.4	Outline	8
2	BACKGROUND	11
2.1	Human Computation and the Wisdom of Crowds	12
2.1.1	Definitions	13
2.1.2	Role and Application Domains	15
2.2	Workflows, Ontologies and the Semantic Web	17
2.2.1	Activity-based Workflows	18
2.2.2	Ontologies and Semantics in Workflows	20
2.3	Systematization of Human Computation Approaches	22
2.3.1	Entities and Relationships	23
2.3.2	The Process	25
2.3.3	Task-oriented Approaches	27
2.3.4	Workflow-oriented Approaches	30
2.3.5	Comparative Analysis	33
2.4	Ontology of Task-oriented Systems	34
2.4.1	Nature of Collaboration	37
2.4.2	Architecture	37
2.4.3	Worker Selection	39
2.4.4	Worker Assessment and Quality Control	40
2.4.5	Worker Motivation	40
2.4.6	Task Creation and Configuration	41
2.4.7	Task Management	41
2.4.8	Task Execution	42
2.4.9	Task Result Aggregation	43
2.5	Challenges	44
2.6	Summary	46

ii	THE COMPFLOW APPROACH	49
3	APPROACH OVERVIEW	51
3.1	Static and Dynamic Ontologies	52
3.2	Construction of the Workflow-Definition	52
3.3	Instantiation and Execution of the Workflow-Definition	53
3.4	Summary	53
4	ONTOLOGIES	55
4.1	Ontologies in Description Logics	55
4.1.1	Definition of Ontology	56
4.1.2	Definition of Knowledge Base	58
4.1.3	Interpretation of the DL Language	58
4.1.4	Domain Ontologies	60
4.2	The CompFlow Ontology	61
4.2.1	Jobs	62
4.2.2	Interfaces	63
4.2.3	Actors	63
4.2.4	Activities	64
4.2.5	Transitions	67
4.3	Summary	69
5	A METHOD FOR THE CONSTRUCTION OF WORKFLOW-DEFINITIONS	71
5.1	The Workflow-Definition Ontology	71
5.2	Task-Definitions	72
5.2.1	Definition	72
5.2.2	Types	75
5.2.3	Cardinalities	76
5.2.4	Example No. 1: Creating Individuals	78
5.2.5	Example No. 2: Optional Input or Output	79
5.2.6	Example No. 3: Concept Hierarchies	81
5.2.7	Example No. 4: Selecting or Filtering Individuals	82
5.2.8	Example No. 5: Updating Individuals	84
5.3	Event-Definitions	85
5.3.1	Definition	85
5.3.2	Example	87
5.4	Transition-Definitions	88
5.4.1	Definition	88
5.4.2	Types	89
5.4.3	Example No. 1: Flow Synchronization	91

5.4.4	Example No. 2: Flow Merge	91
5.4.5	Example No. 3: Flow Parallelization	92
5.4.6	Example No. 4: Flow Disjunction	92
5.4.7	Example No. 5: Flow Conditions	93
5.5	Workflow-Definitions	94
5.5.1	Definition	94
5.5.2	Dependencies on Task-Definitions	96
5.5.3	Inferring Transition-Definitions from Dependencies	97
5.5.4	Aggregation of Redundant Results	98
5.5.5	Example No. 1: Aggregation of Task-Definition Results	99
5.5.6	Example No. 2: Text Partition and Translation	102
5.6	Summary	104
6	INSTANTIATION AND EXECUTION OF WORKFLOW-DEFINITIONS	107
6.1	Task-Definition Instantiation and Execution	107
6.1.1	Definition of Task	108
6.1.2	Instantiation	110
6.1.3	Execution	113
6.2	Event-Definition Instantiation and Execution	113
6.2.1	Definition of Event	114
6.2.2	Instantiation	115
6.2.3	Execution	115
6.3	Transition-Definition Instantiation and Execution	115
6.3.1	Definition of Transition	116
6.3.2	Instantiation	117
6.3.3	Execution	117
6.4	Workflow-Definition Instantiation and Execution	117
6.4.1	Definition of Workflow	118
6.4.2	Instantiation	119
6.4.3	Event-Driven Instantiation	120
6.4.4	Execution	120
6.4.5	Example	120
6.5	Summary	126
7	ASSISTED CONSTRUCTION OF WORKFLOW-DEFINITIONS	129
7.1	The Layered Architecture	130
7.1.1	Command Layers	130
7.1.2	Command Contexts	131
7.2	Atomic Commands	132

7.2.1	The Task-Definition Context	134
7.2.2	The Event-Definition Context	143
7.2.3	The Transition-Definition Context	147
7.2.4	The Workflow-Definition Context	147
7.3	Pattern Commands	150
7.3.1	Follow-Role-CreateAndFill Pattern Commands	151
7.3.2	Partition Pattern Commands	154
7.3.3	Assembler Pattern Commands	158
7.3.4	Other Pattern Commands	161
7.4	Strategies	161
7.5	Summary	162
iii	POSTAMBLE	163
8	EVALUATION AND USE CASE SCENARIOS	165
8.1	CompFlow Construction Framework	165
8.1.1	Visual Workflow-Definitions	165
8.1.2	Construction Framework Architecture	167
8.2	CompFlow Engine	168
8.2.1	Engine Architecture	169
8.2.2	The Ontology Module	170
8.2.3	The Extensible Interface Module	171
8.2.4	The Interface Management Module	175
8.2.5	The Workflow and Task Modules	175
8.2.6	The Job Module	176
8.2.7	External Libraries	176
8.3	Use Case Scenarios	177
8.3.1	The Document Translation Scenario	177
8.3.2	The Ontology Alignment and Construction Scenario	185
8.3.3	The Catalan Constitution Refinement Scenario	196
8.3.4	Integration with External Projects	202
8.4	Summary	203
9	CONCLUSIONS	205
9.1	Contributions	206
9.2	Limitations	208
9.3	Future Work	209
	BIBLIOGRAPHY	211

LIST OF FIGURES

Figure 1	Integration of the dynamic (task) dimension and the static domain dimension in a workflow-definition.	6
Figure 2	Research areas involved in the presented survey and state of the art	12
Figure 3	Workflow management layers of action	18
Figure 4	Business process definition components according to the BPMN 2.0	20
Figure 5	Relationships between the concepts Micro-Task, Task and Complex Task	23
Figure 6	Entity relationship diagram of a common (complex) task CS and HC system	24
Figure 7	Process diagram of a common (complex) task CS and HC system	26
Figure 8	Overview of the CompFlow approach	51
Figure 9	The document ontology (TBox only) with a possible example ABox (or instantiation)	60
Figure 10	Overview of the CompFlow ontology	62
Figure 11	Actors in the CompFlow ontology	64
Figure 12	Abstract representation of a task according to the CompFlow ontology	65
Figure 13	Types of tasks in the CompFlow ontology	66
Figure 14	Events in the CompFlow ontology	67
Figure 15	Example of the operational TBox and ABox of a workflow	68
Figure 16	Transitions in the CompFlow ontology	68
Figure 17	Operational DL representation of a task-definition that creates new individuals	79
Figure 18	Operational DL representation of a task-definition with optional input and output data	80
Figure 19	Operational DL representation of a task-definition with hierarchies of input and output concepts	81
Figure 20	Operational DL representation of a task-definition with a filter operation	83

Figure 21	Operational DL representation of a task-definition with an update operation	84
Figure 22	Operational DL representation of an event-definition	88
Figure 23	Operational DL representation of a dependent task-definition that filters the assignments of a previous task-definition	100
Figure 24	Operational DL representation of a workflow-definition for partitioning and translating sections according to the document ontology	104
Figure 25	Overview of the task-definition instantiation and execution process	108
Figure 26	Overview of the RunningEvent event-definition instantiation and execution process	114
Figure 27	Overview of the transition-definition instantiation and execution process	116
Figure 28	Overview of the workflow-definition instantiation and execution process	118
Figure 29	Evolution of the operational ABox during the instantiation and execution of T ₁	122
Figure 30	Evolution of the operational ABox during the instantiation and execution of T ₂	123
Figure 31	New data in the operational ABox after the domain concept to input concept mapping step of T ₃	125
Figure 32	New data in the operational ABox after the unit and context association process of T ₃	126
Figure 33	New data in the operational ABox after the execution of T ₃	126
Figure 34	Overview of the CompFlow assisted construction process	129
Figure 35	Layered architecture of the operations in the assisted workflow-definition construction process	130
Figure 36	Example of a workflow-definition context hierarchy	131
Figure 37	Transitions between contexts in the assisted workflow-definition construction process	133
Figure 38	Result of the execution of a partition pattern command with four partition steps	155
Figure 39	Result of the execution of an assembler pattern command	159
Figure 40	The visual workflow-definition construction environment prototype	166

Figure 41	Implementation architecture of the CompFlow construction framework	167
Figure 42	Deployment-dependent and workflow-definition-dependent concepts in the CompFlow ontology	168
Figure 43	Implementation architecture of the CompFlow engine	170
Figure 44	TaskInterface mapping example	172
Figure 45	Screenshot of the LocalWebCrowdInterface	172
Figure 46	Overview of the document translation workflow-definition	178
Figure 47	Task-definitions in the document translation workflow-definition	178
Figure 48	Example assignment with the UI template of T2 in the document translation scenario	180
Figure 49	Example assignment with the UI template of T3 in the document translation scenario	181
Figure 50	Example assignment with the UI template of T4 in the document translation scenario	182
Figure 51	Answers to T3Q1, T3Q2, T3Q3 and T3Q4 in the document translation scenario	183
Figure 52	Answers to T4Q1, T4Q2, T4Q3 and T4Q4 in the document translation scenario	184
Figure 53	Overview of the ontology alignment and construction workflow-definition	186
Figure 54	Additional concepts in the ontology alignment and construction domain ontology	186
Figure 55	Task-definitions in the ontology alignment and construction workflow-definition (part 1)	187
Figure 56	Task-definitions in the ontology alignment and construction workflow-definition (part 2)	188
Figure 57	Assignment with the UI template of T1 in the ontology alignment and construction scenario	191
Figure 58	Assignment with the UI template of T2 in the ontology alignment and construction scenario	192
Figure 59	Assignment with the UI template of T3 in the ontology alignment and construction scenario	192
Figure 60	Assignment with the UI template of T4 in the ontology alignment and construction scenario	193
Figure 61	Overview of the Catalan constitution refinement process	198
Figure 62	Partial TBox of the Constitute project ontology	198

Figure 63	Task-definitions in the constitution refinement workflow-definition	199
Figure 64	Assignment with the UI template of T1 in the Catalan constitution refinement scenario	200
Figure 65	Assignment with the UI template of T2 in the Catalan constitution refinement scenario	201
Figure 66	Assignment with the UI template of T3 in the Catalan constitution refinement scenario	202

LIST OF TABLES

Table 1	Comparison of CS systems	35
Table 2	Terminologies employed by CS systems	36
Table 3	Classification of CS and HC systems	38
Table 4	Classification of CS systems according to the worker selection dimension	39
Table 5	Classification of CS systems according to the worker assessment and quality control dimension	40
Table 6	Classification of CS systems according to the task creation and configuration dimension	42
Table 7	Classification of CS systems according to the task management dimension	42
Table 8	Classification of CS systems according to the task execution dimension	43
Table 9	Classification of CS systems according to the task result aggregation dimension	44
Table 10	Syntax rule of the employed description logic language	56
Table 11	Additional expressions in the ontology definition	57
Table 12	The DL interpretation function definitions	59
Table 13	Additional functions for task-definitions	74
Table 14	Additional functions for event-definitions	86
Table 15	Additional functions for transition-definitions	89
Table 16	Additional functions for transitions	116
Table 17	Command sequence for the construction of the document translation workflow-definition	179
Table 18	Distribution of workers and assignments throughout each task in the document translation workflow	181
Table 19	Command sequence for the construction of the ontology alignment and construction workflow-definition (part 1)	189
Table 20	Command sequence for the construction of the ontology alignment and construction workflow-definition (part 2)	190
Table 21	Distribution of workers and assignments throughout each task in the ontology alignment and construction workflow	193

Table 22	Resulting alignment between the CMT and Conference ontologies in the ontology alignment and construction workflow	194
Table 23	Reference alignment between the CMT and Conference ontologies in the ontology alignment and construction workflow	195
Table 24	T ₄ results in the ontology alignment and construction workflow .	197

LIST OF ALGORITHMS

Algorithm 1	The assignment aggregation task-definition automatic construction process.	98
Algorithm 2	The domain concept to input concept mapping process in the task-definition instantiation.	110
Algorithm 3	The unit association process in the task-definition instantiation.	111
Algorithm 4	The context association process in the task-definition instantiation.	111
Algorithm 5	The assignment cloning process in the task-definition instantiation.	112
Algorithm 6	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 1.	135
Algorithm 7	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 2.	136
Algorithm 8	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 3.	136
Algorithm 9	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 4.	136
Algorithm 10	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 5.	137
Algorithm 11	Suggestion of Create-Input-Concept commands in a task-definition context: sub-set no. 6.	137
Algorithm 12	Suggestion of Create-Output-Concept commands in a task-definition context: sub-set no. 1.	139
Algorithm 13	Suggestion of Create-Output-Concept commands in a task-definition context: sub-set no. 2.	139
Algorithm 14	Suggestion of Create-Output-Concept commands in a task-definition context: sub-set no. 3.	140
Algorithm 15	Suggestion of Create-Relation commands in a task-definition context: sub-set no. 1.	141

Algorithm 16	Suggestion of Create-Relation commands in a task-definition context: sub-set no. 2.	141
Algorithm 17	Suggestion of Create-Internal-Dependency commands in a task-definition context: sub-set no. 1.	142
Algorithm 18	Suggestion of Create-Internal-Dependency commands in a task-definition context: sub-set no. 2.	142
Algorithm 19	Suggestion of Create-External-Dependency commands in a task-definition context.	143
Algorithm 20	Suggestion of Create-Input-Concept commands in an event-definition context: sub-set no. 1.	144
Algorithm 21	Suggestion of Create-Input-Concept commands in an event-definition context: sub-set no. 2.	145
Algorithm 22	Suggestion of Create-Input-Concept commands in an event-definition context: sub-set no. 3.	145
Algorithm 23	Suggestion of Create-Relation commands in an event-definition context.	146
Algorithm 24	Suggestion of Create-Internal-Dependency commands in an event-definition context.	147
Algorithm 25	Suggestion of Create-Transition-Definition commands in a workflow-definition context: sub-set 1.	149
Algorithm 26	Suggestion of Create-Transition-Definition commands in a workflow-definition context: sub-set 2.	150
Algorithm 27	Execution of a Follow-Role-CreateAndFill pattern command in a workflow-definition context.	152
Algorithm 28	Suggestion of Follow-Role-CreateAndFill pattern commands in a workflow-definition context.	153
Algorithm 29	Execution of a partition pattern command in a workflow-definition context.	155
Algorithm 30	Suggestion of partition pattern commands in a workflow-definition context: lexical verification.	156
Algorithm 31	Suggestion of partition pattern commands in a workflow-definition context: lexical and structural verification.	157

Algorithm 32	Execution of an assembler pattern command in a workflow-definition context.	158
Algorithm 33	Usage example of the Freemarker template engine data structure.	173
Algorithm 34	Usage example of the JavaScript data structure with the jQuery API.	174

ACRONYMS

ABox	Assertion Box
API	Application Programming Interface
AnBox	Annotation Box
BDO	Business Domain Ontology
BOWL	Business-OWL
BPDM	Business Process Definition Meta-model
BPMI	Business Process Management Initiative
BPMN	Business Process Modelling Notation
BPMNO	Business Process Modelling Notation Ontology
BPMO	Business Process Modelling Ontology
BPQL	Business Process Query Language
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CML	CrowdFlower Markup Language
CMT	Conference Management Toolkit
CRUD	Create, Read, Update, Delete
CS	Crowdsourcing
CSV	Comma Separated Values
DL	Description Logics
DoCO	Document Components Ontology
DOM	Document Object Model
DynEaaS	Dynamic Evaluation as a Service
EDOAL	Expressive and Declarative Ontology Alignment Language
GUI	Graphical User Interface
HC	Human Computation

HIT Human Intelligence Task

HTML Hyper-Text Markup Language

IP Internet Protocol

JAR Java ARchive

KB Knowledge Base

LOD Linked Open Data

OAEI Ontology Alignment Evaluation Initiative

OASIS Organization for the Advancement of Structured Information Standards

OWL Web Ontology Language

OWL-S OWL Semantic Markup for Web Services

OWL-T OWL Task

OWLIM OWL In Memory

RBox Role Box

REST REpresentational State Transfer

RMIT Royal Melbourne Institute of Technology

SBO Semantic Bridge Ontology

SDB SPARQL Database

TBox Terminological Box or Taxonomy

TDB Tuple Database

UI User Interface

UML Unified Modelling Language

W₃C World Wide Web Consortium

WfMC Workflow Management Coalition

WS-BPEL Web Services Business Process Execution Language

XPDL XML Process Definition Language

YAWL Yet Another Workflow Language

Part I

PREAMBLE

INTRODUCTION

Over the last decade, human computation has re-emerged as a relevant research field and as a form of collective intelligence. A relatively recent and popular form of human computation is micro-task crowdsourcing [23], which consists in assembling really small tasks and distributing them through a crowd of workers.

Several experiments in different domains have shown that micro-task crowdsourcing has great potential for solving large scale problems that are often difficult for computers to solve automatically, on their own [76]. These problems usually require a degree of creativity or just common sense plus some background knowledge [8, 67].

More recently, a special interest in employing crowdsourcing towards solving complex tasks has emerged [1, 27, 32, 37, 39, 62]. Following the trend of the current crowdsourcing platforms, which feature the execution of single micro-tasks, this interest has led to the emergence of new approaches built upon workflows of micro-tasks. The modelling of such workflows allows the crowdsourcing of a new kind of more complex tasks, which require the ordered and controlled execution of multiple types of micro-tasks.

Micro-task workflows present new challenges at different dimensions of the human-machine computation process. These challenges include micro-task specification in a way understandable to both human and machine workers, control of the micro-task flow, selection and assessment of worker performance, quality control, visualization, and reporting [1, 27]. While some of these challenges can be tackled through adapted solutions found in traditional workflow approaches (e. g. business processes and web service composition), others are specific to human-machine computation environments.

1.1 PROBLEM STATEMENT AND MOTIVATIONS

While some of the micro-tasks in a workflow are better performed by humans, others are better performed by a machine [35]. Besides their focus on exclusively human groups of workers, current micro-task workflow and non-workflow approaches rely

on unstructured (e. g. in natural language) or lightly structured (e. g. in the form of spreadsheets) input and output data. Furthermore, micro-task interfaces are built using mark-up languages that contain little or no meta-data. The unstructured nature of micro-tasks in terms of domain representation, therefore, makes it difficult for task requesters to include machine workers in the workflow execution process [59].

As shown from the recent developments in traditional workflow approaches and business process management, there is also a need for semantically-enriched representations of workflows and tasks that are shareable, extensible and allow the easy integration and assembly of different workflows and tasks [47, 49, 55]. However, although the dynamic (e. g. activities and tasks) and static (e. g. data objects) dimensions involved in workflows are conceptually entwined, the current approaches result in mostly separate and distinct representations for each of these dimensions.

In a broad sense, human-machine micro-task workflow challenges can be characterized in terms of four dimensions: (i) representation, (ii) human-machine interaction, (iii) quality control and (iv) management. Of particular interest, in the context of this thesis, are the challenges presented in:

- The representation of micro-task workflows (i), namely:
 - Full structure and semantics of the operations, input, and output of micro-tasks;
 - Partitioning and establishing different granularities for micro-tasks;
 - Controlling the flow of micro-tasks;
 - Achieving understandable and interpretable representations for both humans and machines;
 - Fully integrating the dynamic and the static domain dimensions;
 - Providing flexible, extensible and shareable representations;
- Human-machine interaction (ii), namely:
 - Specifying worker selection requirements for different phases of the workflow;
 - Identifying workers and requesting their participation in multiple micro-tasks;
 - Providing relevant contextual information.

1.2 HYPOTHESIS AND OBJECTIVES

Several research issues were raised in the context of micro-task workflows in human-machine environments. This thesis, however, focuses on the particular aspects of representation, construction and execution of micro-task workflows with an emphasis on human-machine interaction. The overarching goal and hypothesis is therefore to investigate if:

Workflows of micro-tasks can be represented and semi-automatically built through a formal and seamless full integration of semantically-enriched dynamic and static domain dimensions, in such a way that they are interpretable and executable by both human and machine actors in a human-machine execution environment.

This hypothesis raises several research questions and establishes multiple requirements.

RESEARCH QUESTION NO.1: *Can the structure and semantics of micro-tasks and micro-task workflows be fully represented in terms of their dynamic and static domain dimensions?*

Current micro-task representations lack the ability to properly describe the semantics of the operations and their full domain of impact. This mostly occurs because of the reduced complexity of opting for disconnected and partial input and output representations.

In this sense, one of the objectives is to devise a method that allows the formal representation of micro-task workflows through a seamless integration of both the dynamic (task) dimension and the static domain dimension, as depicted in [Figure 1](#). More precisely, abstract structures in the dynamic dimension are associated with a particular domain of knowledge (given by the static domain dimension) through a new conceptualization that becomes the workflow-definition (i. e. the representation of the workflow). The method must:

- Fully represent the structure and the semantics of Create, Read, Update, Delete (CRUD) micro-task operations, avoiding the need for specific implementations for each task;
- Fully represent the structure and the semantics of micro-task input and output data;

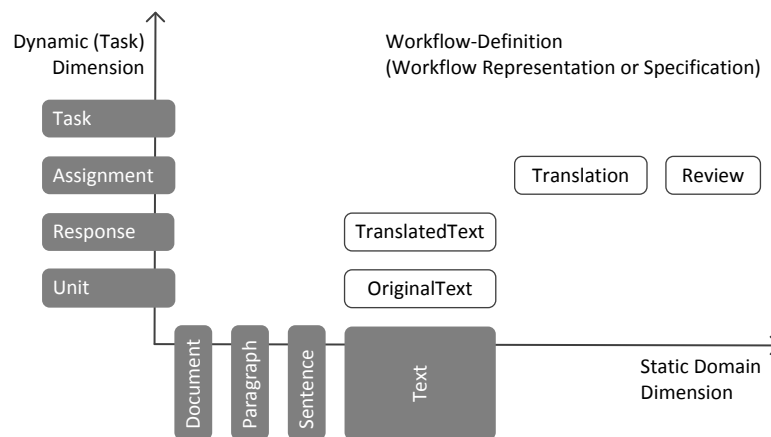


Figure 1: Integration of the dynamic (task) dimension and the static domain dimension in a workflow-definition.

- Provide an expressive mechanism for the representation of transitions (similar to those found in traditional business process approaches).

Overall, the proposed method must satisfy the overall requirements established for traditional workflow representations and provide a full semantic and structural representation of the micro-task operations, which is both interpretable by machines and close to the human conceptual level.

Another objective is to formally establish the automatic or semi-automatic instantiation and execution of workflow-definitions, by following the rules and semantics established by the proposed representation and construction method.

The proposed representation and construction method, along with all instantiation and execution processes, will be evaluated through the implementation of a workflow-definition instantiation and execution engine.

RESEARCH QUESTION NO.2: *Is there a way to incrementally build the micro-task workflow dynamic dimension based on the static domain dimension?*

Since the full representation of the dynamic and static nature of micro-tasks and micro-task workflows will most likely increase the complexity of manually constructing new workflow-definitions, new approaches that aid the requester and the creators of the workflow-definition are required.

The main objective is the definition of a process that facilitates (or assists in) the construction of workflow-definitions. More precisely, the process must:

- Focus on the iterative and assisted creation of the workflow-definition through an existing domain representation (the static domain dimension);
- Automate the construction process while still providing a high level of control to the creator of the workflow-definition;
- Provide a flexible approach to the construction of workflow-definitions that can be adapted to particular domains.

To evaluate the proposed construction process, a workflow-definition construction framework will be implemented.

RESEARCH QUESTION NO.3: *Can a workflow of micro-tasks, and the involved actors and interfaces, be represented through an unique and platform-independent representation mechanism, understandable and interpretable to both humans and machines?*

Both traditional workflow approaches and human-machine computation approaches have already established several requirements regarding the integration of computational efforts between humans and machines. However, the lack of semantics and the use of different representation mechanisms for the domain data and the workflow structure (the latter often with structurally and semantically weak relationships to the first), has led to several approaches that either tend to be far from the human conceptual level or too unstructured (far from being interpretable by machines).

From this research question, the following objectives regarding the micro-task and micro-task workflow representations are established:

- They must be shareable and extensible;
- They must be close to the human conceptual level, while retaining the necessary characteristics that make them interpretable and understood by machines;
- They must allow an expressive representation of the workers involved on each micro-task;
- They must allow an expressive representation of the interfaces that deliver the micro-tasks to each worker.

The implementation of both the instantiation and execution engine, along with the implementation of the construction framework, will provide relevant data for the assessment of this research question. Furthermore, several experiments in different scenarios will be conducted using these implementations.

1.3 APPROACH

As stated by Obrst et al. [52], ontologies “represent the best answer to the demand for intelligent systems that operate closer to the human conceptual level”. Domain ontologies are not only able to describe the domain knowledge, but also to describe micro-task workflows and the data flowing through them in a way understandable to both humans and machines. Furthermore, ontologies are flexible, extensible and shareable representations of knowledge.

The growing popularity of ontologies in the context of the Linked Open Data (LOD) and the Semantic Web has led to the emergence of multiple ontologies in several different domains. These ontologies are mostly used to establish a consensus about a particular domain of knowledge and can be used in different application scenarios such as the integration and alignment of enterprise data [2, 46]. The structure and semantics found in domain ontologies can also be used to represent the data involved in the execution of micro-task workflows, i. e. to represent the static domain dimension.

In this sense, the use of Description Logics (DL) ontologies is proposed for the representation of workflows. Each workflow-definition is an ontology that seamlessly integrates the dynamic and static domain dimensions. Micro-tasks are defined as extensions of the static domain dimension that provide a full representation of the data involved in the micro-task.

The extensible nature of ontologies allows the static domain dimension to be re-used in several workflow-definitions. Also, the wide set of domain ontologies found throughout the Web can be used to iteratively build new workflow-definitions.

1.4 OUTLINE

The defined objectives of this thesis are addressed throughout the document, which is structured as follows:

- Part I (Preamble):
 - Chapter 1 (the current chapter) outlines the work presented in this thesis document, along with the motivations and research questions;
 - Chapter 2 presents an analysis of the current state of the art, identifying challenges regarding human-computation systems, micro-task and workflow representations for human-machine computation;

- Part II (The CompFlow Approach):
 - Chapter 3 provides an overview of the proposed approach, entitled CompFlow;
 - Chapter 4 introduces DL ontologies and their role in the CompFlow approach;
 - Chapter 5 describes the proposed method for building workflow-definitions;
 - Chapter 6 formally defines the workflow-definition instantiation and execution process;
 - Chapter 7 introduces the semi-automatic iterative workflow-definition construction process based on the previously described method;
- Part III (Postamble):
 - Chapter 8 provides a description of the implemented workflow construction, instantiation and execution environment, along with three evaluation scenarios;
 - Chapter 9 presents the conclusions, which include a summary of the contributions, a discussion of the obtained results, and future work directions.

BACKGROUND

Since the advent of artificial intelligence, researchers have been trying to create machines that emulate human behaviour. This has led to multiple branches of artificial intelligence such as multi-agent systems, reasoning and negotiation. Back in the 1960s however, Licklider [35] believed that machines and computers were just part of a scale which weights humans on one side, and machines on the other. His vision was that machines and humans should work together performing complementary roles [35, 59].

It was only recently that relevant research emerged and brought humans into machine affairs. The early contributors to this retake on Lickliders' vision might as well be the social Web and the harnessing of collective intelligence [19]. These proved that humans have great complementary abilities that are relative to machines, and that they can act as guided computational units.

As a part of collective intelligence [59], human computation [76] re-emerged as a relevant research field. Shortly after, the term crowdsourcing was coined [23], leading to yet another field of research that is highly connected to human computation.

After almost one decade of active research into human computation and crowdsourcing, several approaches and business models based on crowdsourcing have emerged, managing and distributing work to the crowd [11, 59, 79]. In this sense, crowdsourcing (currently the most popularized term) can be seen from two different perspectives: a business domain-specific perspective, and a technical domain-independent perspective.

Regardless of the research and development perspective, the application of human-machine computation to several domains is spreading. Alongside emerges a special interest in employing crowdsourcing towards solving more complex tasks, which establish requirements regarding the representation of tasks that cannot be satisfied through single and independent micro-tasks. This interest has led to several approaches being built upon workflows of micro-tasks [1, 27, 32, 37, 39, 62].

As depicted in Figure 2, this chapter presents a survey of crowdsourcing human computation systems from a technical domain-independent perspective, with a focus on solving micro-tasks and complex tasks. Since an emphasis is given to workflows

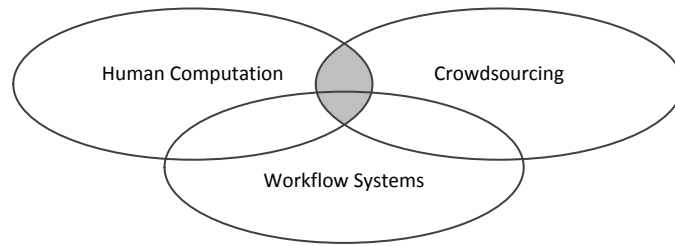


Figure 2: Research areas involved in the presented survey and state of the art. The analysed systems belong to the gray area.

of micro-tasks, an overview of workflow representation approaches is also given. First, a discussion of the concepts of collective intelligence, human-machine computation and crowdsourcing is presented, along with their role and applications in several different domains. Afterwards, an overview of workflow representation approaches is presented, followed by a detailed analysis of several micro-task and workflow-based crowdsourcing systems. This analysis is given according to scientific publications and the empirical analysis of the respective online system, if available. Finally, a systematization of the micro-task and complex task crowdsourcing systems along with current challenges is presented.

2.1 HUMAN COMPUTATION AND THE WISDOM OF CROWDS

Humans are innately social and the intrinsic aspects of human cooperation have been the subjects of great research efforts [58]. Humans not only tend to form a clustered structure of relationships (social circle), but also extract individual benefits from them [34]. Social circles have a great impact on our lives, influencing our ideas and behaviour [31]. Not so long ago, the information that a person had access to was mostly the information flowing inside his/her social circle. Nowadays, our social circle also acts as a filter for all the vast amounts of information and choices delivered to us every day [45] by other means (e. g. social media, internet).

The emergence of the social Web has brought new powerful Web applications that connect people on a global scale, and allow them to reap the benefits of social life from online virtual environments in a global scale. Along with their huge popularity, online social networks allow the retrieval of significant amounts of important social data, which can be used to promote social benefits [34].

One of the most straightforward benefits we extract from society comes from asking our friends for an opinion or advice [45]. It is possible to apply a similar mechanism to

online social networks by automatically filtering data, and providing the user with relevant and personalized results according to the opinions coming from his/her online social circle. The difference is that, unlike humans alone, the introduction of machines allows that procedure to be performed for millions of items, covering a wide social circle.

Online social networks also destroy geographical barriers, thus promoting the combination of behaviours and ideas on a global scale [58]. This combination is often referred to as collective intelligence [38].

An interesting example of the importance of collective intelligence is what Porter [58] regards as the Amazon Effect. To explain the Amazon Effect, he describes a usability study where people were asked to buy a product at a certain online store. A lot of people wanted to go to Amazon first, and when they were asked why, they answered that they would like to do some research on the product, even if they were not buying it on Amazon.

2.1.1 Definitions

Brabham [5] follows Surowieckis' [70] view on the wisdom of crowds (often referred to in the context of the Web as collective intelligence), which states that it emerges from aggregating individual solutions, instead of averaging them (as in the case of Amazons' review system). This view is particularly relevant in the context of problem solving, where aggregating individual solutions often leads to a better solution than any of the best originally proposed individual solutions. Following this view, Brabham argues that crowdsourcing is a model achieved through the Web that is "capable of aggregating talent, leveraging ingenuity while reducing costs and time formerly needed to solve problems".

Crowdsourcing is a term popularized by Howe [23, 24] that emerged in the context of a paradigm shift in business models. This shift originated from companies that started to provide outsourcing services relying on anonymous communities or crowds throughout the Web (e. g. iStockPhoto¹, InnoCentive² and Amazons' Mechanical Turk³). By 2006, these communities were growing into incredible valuable work forces capable of performing several specific tasks in exchange for small monetary rewards.

¹<http://www.istockphoto.com>

²<http://www.innocentive.com>

³<https://www.mturk.com>

Since then, several similar definitions for the term crowdsourcing have been given. According to Howe [24], “crowdsourcing is the act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large group of people”. Doan et al. [11] define crowdsourcing as a system that “enlists a crowd of humans to help solve a problem defined by the system owners, and if in doing so, it addresses” the challenges of recruiting and retaining users, defining which contributions can be made by users, combining these contributions and evaluating user performance.

Quinn and Bederson [59] not only provide a definition for crowdsourcing, but also compare it to terms such as human computation, social computing and collective intelligence. They aggregate several definitions found in literature and state that crowdsourcing is a form of collective intelligence that overlaps human computation.

The term human computation dates back to the early years of artificial intelligence, in the 1960s, where it was envisioned that machines and humans should work together performing complementary roles [35, 59]. Still, the vision of human-machine collaboration only started to be properly explored after 2005, the year Von Ahn [76] published his doctoral thesis entitled Human Computation. Von Ahn [76] proposes the use of human algorithm games to harness the distributed processing power of humans to perform specific tasks. In accordance, human computation can be defined as a computational process that involves humans and their cooperation in order to solve problems that computers cannot yet solve [59]. This definition is complemented by stating that “human computation does not encompass online discussions or creative projects where the initiative and flow of activity are directed primarily by the participants’ inspiration, as opposed to a predetermined plan designed to solve a computational problem”.

Quinn and Bederson [59] argue that while human computation requires humans to act as managed units that merely perform a computation, crowdsourcing requires several humans to cooperate in a process by performing a computation or a creative task that is not always managed by computers (e. g. Wikipedia).

Crowds have an important role in human computation. Besides providing good amounts of computational power, applicable in tasks that machines can barely solve with efficiency and efficacy, they can also be used for redundancy.

With the evolution and absorption of crowdsourcing and human computation into market-places and businesses, it can be observed that while Human Computation (HC) is a term that is mostly used by the scientific community, Crowdsourcing (CS) is a term highly employed in the business world.

2.1.2 *Role and Application Domains*

Several experiments in different domains have shown that CS and HC have great potential for solving large-scale problems that are often difficult for computers to solve automatically, on their own [76]. These problems usually require a degree of creativity or just common sense plus some background knowledge [8, 67]. The interpretation and recognition of images and natural language are two examples of these kinds of problems.

One of the applications of CS lies in harnessing geographical information. Recently, the production of geo-referenced data, maps, and atlases has moved from mapping agencies and corporations to non-expert users [18]. Some of the services that allow this include Flickr, Googles' MyMaps, OpenStreetMap, and Wikimapia. Following this trend, Goodchild and Glennon [18] discuss the applications of CS to the harnessing of geographical information for disaster response. They argue about the importance of quality in harnessing geographical information and present an analysis of non-expert user generated geographical information from occurrences of wildfires in Santa Barbara, California. Although further research is needed, there is great potential for quickly generating and spreading disaster-related information through a CS system.

The potential and importance of CS in harnessing geographical information has also been successfully noted and put into practice by Safecast⁴, a project that emerged one week after the earthquake that led to the Fukushima Japanese nuclear accident. Safecast is a "global sensor network for collecting and sharing radiation measurements to empower people with data about their environments". In order to collect data, different types of radiation sensors are distributed through volunteers that later use them to collect geo-referenced radiation measurements during their travels. The results are collected and published by Safecast, which provides free access to the data.

Several CS-based businesses have emerged since the advent of CS platforms. While some maintain their own community of workers (e. g. MicroWorkers⁵, ShortTask⁶), others interact with one or more CS platforms (e. g. CrowdFlower⁷) offering their services in designing and managing projects and tasks, and obtaining reliable results. Brabham [5] discusses several successful applications of CS as business models, which include Threadless, InnoCentive, and iStockPhoto. Threadless crowdsources the design process of t-shirts by promoting online competition. InnoCentive has a different focus, as

⁴<http://blog.safecast.org/>

⁵<https://microworkers.com>

⁶<http://www.shorttask.com>

⁷<http://crowdfLOWER.com>

it crowdsources the research and development of scientific problems as challenges. The last, iStockPhoto, sells photographs, animation, and video clips produced by its crowd of artists. Interestingly, surveys of the iStockPhoto crowd show that the main motivations behind their time and effort are not only monetary but also enjoyment and the development of individual skills [6].

In 2010, Dawson and Alexandrov published a diagram depicting the landscape of CS⁸. They distinguish CS systems through thirteen categories, enumerating several domains where CS has been applied. The presented categories are: crowdsourcing aggregators (e. g. CrowdFlower), content markets (e. g. iStockPhoto), prediction markets (e. g. Crowdcast), question answering (e. g. Yahoo! Answers), innovation prizes (e. g. XPrize), service marketplaces (e. g. Freelancer), distributed innovation (e. g. InnoCentive), crowdfunding (e. g. KickStarter), competition platforms (e. g. 99 Designs), content-rating (e. g. Delicious), idea platforms (e. g. IdeaScale), data sharing (e. g. Dead Cell-zones), reference content (e. g. Wikipedia), cycle sharing (e. g. SETI@Home) and micro-tasks (e. g. Mechanical Turk and ShortTask). Among the application domains featured by these CS systems are business ideas, 3D and graphic design, data analysis, research, tagging, translation, writing and editing, reviewing and software development.

From all the CS systems and common types of CS tasks enumerated by Dawson and Alexandrov, only some qualify as HC systems. This is the case for CS systems under the micro-tasks category. However, although CS systems like Mechanical Turk and ShortTask provide a platform for building any type of tasks, some specific types of tasks have become widely popular for being particularly adequate for micro-task representation, and for being easily accepted by workers. These specific types of tasks are often (as presented in CloudCrowd⁹) writing, editing, categorization, searching, data entry and translation tasks.

In this sense, most CS micro-task systems feature the creation of task templates that can be used to request multiple similar tasks. These systems often provide a predefined set of templates for commonly requested types of tasks. Some of the predefined templates provided by Mechanical Turk and ShortTask include:

- Categorization and classification;
- Data verification (e. g. provide the correct spelling);
- Data extraction (e. g. finding a website address);

⁸<http://crowdsourcingresults.com/competition-platforms/crowdsourcing-landscape-discussion>

⁹<http://www.cloudcrowd.com>

- Moderation and tagging of multimedia content (e. g. tagging images or videos with adult content);
- Transcription from multimedia content (e. g. audio, video and images);
- Sentiment analysis and surveys;
- Search relevance (e. g. evaluate the relevance of search results).

Around these CS systems that manage their own community of workers, CS-oriented businesses have started to emerge. Although these businesses tend to provide services with a tendency towards solving complex tasks, they still share many similarities with common CS micro-task system templates. For instance, MobileWorks¹⁰, a CS-oriented company, groups its services into categories such as digitalization of documents, categorization and classification, researching, and harnessing feedback (e. g. through surveys).

Some of these application domains can be easily modelled and managed with single and independent CS micro-tasks. Recently, however, a special interest in employing CS towards solving more complex tasks has emerged [1, 27, 32, 37, 39, 62]. This interest has led to several approaches being built upon workflows of micro-tasks.

2.2 WORKFLOWS, ONTOLOGIES AND THE SEMANTIC WEB

Workflow approaches are often a typical manifestation of human-machine computation, where humans are guided to act as computational units and their efforts are integrated with the effort of machine algorithms and routines.

The Workflow Management Coalition (WfMC) defines a workflow as “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [33, 75]. Similar definitions describe a workflow as “a collection of tasks organised to accomplish some business process” [50] and even as a process that “supports the coordination and collaboration of people that implement a process” [17].

A common classification of workflows (or business processes) fits them into three different categories: material processes, information processes and business processes [17]. Material processes represent human tasks that are rooted in the physical world. Information processes relate to automated or partially automated tasks (performed by humans and machines). Finally, business processes represent business activities

¹⁰<https://www.mobileworks.com>

implemented as material or information processes. Notice that information processes, in particular but not exclusively, fit into the description of human computation since they can be guided by machines and seen as multiple small human computation steps.

2.2.1 Activity-based Workflows

Workflow and workflow management approaches can be classified as communication-based (modelling the communication steps and commitments between humans), activity-based (modelling work) and hybrid (those that can be classified into both the former categories) [50]. In the context of this work, the focus is given to activity-based approaches since they are closely related to CS workflows.

Extensive studies and surveys have been published regarding activity-based workflows and business processes [17, 29, 50, 75]. Georgakopoulos et al. [17] describe the workflow management process as involving three main layers of action analogous to those presented in Figure 3. The layers of action are (1) process conceptualization, (2) workflow specification and (3) workflow implementation.

The conceptualization of a workflow or process (1) is the act of analyzing a particular domain and understanding the inherent workflow at the conceptual level. The workflow specification (2) consists in the representation of the workflow through a model or specification language. The resulting workflow-definition is the structured and concrete specification of the process. Finally, the implementation (3) is the act of creating and executing a workflow according to a particular workflow-definition.

Following (more or less) this definition and management layers of action, a variety of ways to formal and informal workflow representations have been approached so far. These include the representation of workflows through Petri nets and high-level Petri nets [73, 72], event-driven process chains [66], Unified Modelling Language (UML) activity diagrams [12] and, lately, workflow patterns [61]. While event-driven process chains and UML activity diagrams share many similarities to Petri nets and usually depend on a representation language, the workflow pattern approach is closer to the

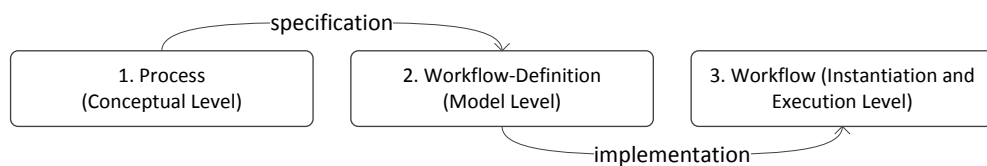


Figure 3: Workflow management layers of action.

conceptual level, focusing on the structural and semantic aspects of workflows. This latter approach emerged from the necessity to deal with the diversity of languages for workflow-definitions [13].

In terms of representation and execution standards, several approaches have been developed by consortia such as the [WfMC](#), the Business Process Management Initiative ([BPMI](#)), the Organization for the Advancement of Structured Information Standards ([OASIS](#)) and the World Wide Web Consortium ([W3C](#)). Overall, they are classified into four different categories [29]:

- Graphical standards - allow the visual representation of business processes with diagrams;
- Execution standards - represent the deployment and automation of business processes;
- Interchange standards - facilitate the portability of business process definitions;
- Diagnosis standards - provide administrative and monitoring capabilities.

Amongst graphical standards are the [UML](#) and the Business Process Modelling Notation ([BPMN](#)). The [BPMN](#), in particular, has been proposed by the [BPMI](#) and establishes a set of components and a notation for the representation of business processes (see [Figure 4](#)). According to the [BPMN 2.0](#), the main workflow components are events, activities (representing tasks and processes) and gateways (control the flow convergence and divergence). They are connected through flow control components and usually fit into swimlanes. Swimlanes organize flow objects and may represent the set (role) of participants that must perform the contained set of activities.

Languages such as the Business Process Definition Meta-model ([BPDM](#)) and the XML Process Definition Language ([XPDL](#)) are able to represent business processes using the [BPMN](#).

The Web Services Business Process Execution Language ([WS-BPEL](#)) has been proposed by [OASIS](#) as a workflow execution standard. The [WS-BPEL](#) is a Web service composition language that describes interactions between a given service and its environment as a composition of communication actions [53]. Alternatively, Yet Another Workflow Language ([YAWL](#)) [74] is an academic execution standard based on Petri nets with formal semantics that focuses on the representation of workflows. [YAWL](#) follows a pattern-based approach, including several control flow patterns (some analogous to the components present in the [BPMN](#)) classified into six different categories: basic control flow patterns, structural patterns, state-based patterns, advanced branching and synchronization patterns, cancellation patterns and multiple instance patterns.

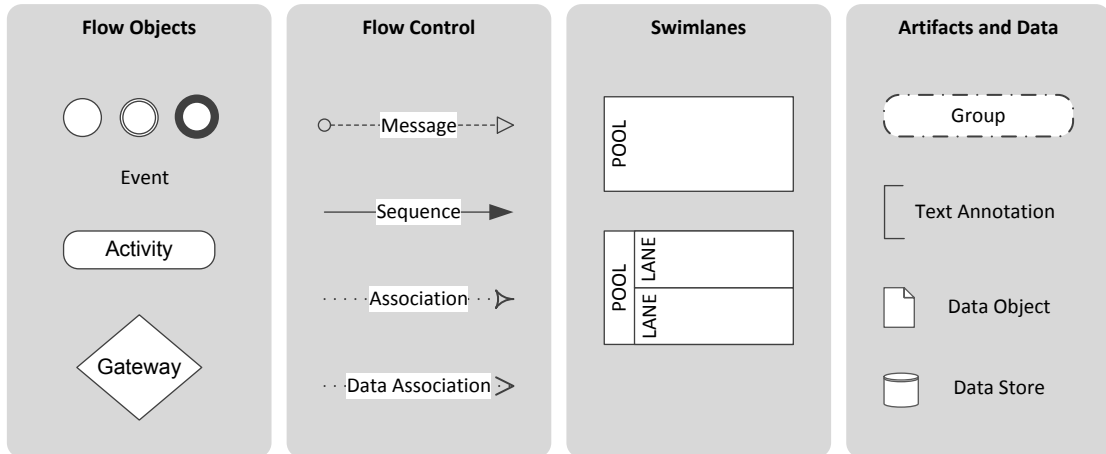


Figure 4: Business process definition components according to the BPMN 2.0.

In order to interchange workflows, standards such as the [BPD](#) and the [XPD](#) [13] are available. In terms of diagnosis, there are standards such as the Business Process Query Language ([BPQL](#)). The [BPQL](#) is a real-time language for querying business process instances that provides an infrastructure for the execution and deployment of business processes.

Despite the amount of different approaches to activity-based workflows (or business processes), standards and representations tend to rely on the same conceptual view of a workflow.

2.2.2 *Ontologies and Semantics in Workflows*

As mentioned by Ko et al. [30], current standards are not able to describe the semantics of how tasks are linked, how they can be decomposed, their relationships to business documents or domain data, and which actors may participate in tasks. These drawbacks can all be tackled through the use of ontologies for the representation of workflow definitions.

The use of ontologies in workflow specifications emerged along with the growing popularity of the semantic Web and the migration of multiple services to the cloud. Consequently, several approaches that regard tasks as semantic Web service operations have emerged.

Semantic Web services provide an “agent-independent declarative API capturing the data and metadata associated with a service together with specifications of its properties and capabilities, the interface for its execution, and the prerequisites and

consequences of its use” [49]. Since semantic Web services are described using ontologies or an ontology-based language, their description shares the common properties of ontologies to facilitate sharing, re-use, composition and mapping. This enables the integration of automated agents in the processes of service discovery, execution, composition, interoperation, orchestration and choreographies [49, 10].

Several ontologies have been proposed to satisfy the demand for semantic Web service representations [55]. One of the most popular is the OWL Semantic Markup for Web Services (*OWL-S*). *OWL-S* is based on the Web Ontology Language (*OWL*) and describes Web services in terms of their profiles (what they do), model (how they work) and groundings (how they can be accessed) [47].

Another approach is that of Di Francescomarino et al. [10], which propose the use of semantic annotations on top of already existing business process specifications. The workflow representation is formalized as business process knowledge base through two ontologies: the Business Process Modelling Notation Ontology (*BPMNO*) and the Business Domain Ontology (*BDO*). The *BPMNO* provides a structural formalization of the *BPMN*. The *BDO* represents any domain ontology describing the typical operations found in a particular domain of knowledge. Additional constraints, that establish the correspondences between the two ontologies, are also part of the business process knowledge base. The aim is not the execution of workflows, but to allow formal and automated reasoning on top of the elements of business processes (e.g. checking for consistency violations and building domain-specific queries).

Natschläger [51] propose an ontology that provides a thorough representation of the *BPMN* 2.0 specification. Although it can be easily extended, due to the inherent features of ontologies, it is intended for the representation of workflows through individuals of the defined concepts. I.e., the concrete representations of workflows are typically found at the instance level, instead of at the model/ontological level. Since it provides a conceptual view of the *BPMN* 2.0 specification, it also “allows a much faster understanding of *BPMN*”.

Cabral et al. [7] propose the use of the Business Process Modelling Ontology (*BPMO*) in order to represent workflows. The ontology captures domain-independent aspects and provides a set of concepts and roles inspired on standards such as the *BPMN* and the *WS-BPEL*.

Notice that the *BPMO* focuses on pure domain-independent aspects and does not integrate the semantics of the domain into the representation. Capturing these semantics

seamlessly remains a complex achievement. The Business-OWL (BOWL) [30], however, provides some inspiration towards capturing these semantics.

The BOWL is an ontology for the representation of hierarchical task networks that gets closer to the domain-oriented conceptual view of a workflow-definition. It describes business processes as hierarchies of decomposable business tasks, focusing on the domain-specific aspects while capturing at the same time the dynamic aspects of a task.

Following the same conceptual view, the OWL Task (OWL-T) [71] is an ontology and language for defining task templates (or task-definitions) that is placed at the top of the semantic Web Service stack. OWL-T contemplates four types of tasks: (i) atomic tasks (directly completed by an operation of a service), (ii) composite tasks (completed by a composition of several operations or services), (iii) simple tasks (either an atomic or composite task) and (iv) complex tasks (a set of one or more simple tasks).

Each task template in OWL-T has several attributes and properties such as *hasInput*, *hasPreCondition*, *hasOutput* and *hasPostCondition*. The *hasInput* and *hasOutput* property typically links to a domain ontology concept. Although the OWL-T allows the representation of task-definitions at the schema-level, it remains decoupled from domain representations since they rely on different languages. Furthermore, the specification of input and output is limited to the scope of a concept.

Overall, OWL-T provides an abstraction on top of the current semantic Web service representations and ontologies.

2.3 SYSTEMATIZATION OF HUMAN COMPUTATION APPROACHES

From a business perspective, there is great interest in accomplishing specific business tasks efficiently and effectively in terms of time and monetary costs. Studies in this context tend to focus on the domain-specific details of the task, giving special concern to user motivation and quality-control aspects [6, 18, 25, 26, 54, 78].

A technical and domain-independent perspective, on the other hand, puts the emphasis on methods, techniques and frameworks for solving problems (in general) that machines alone are not yet able to solve [59, 76]. In some cases, problem-solving can be achieved by replacing machines with humans in certain computation steps where humans usually perform better (human computation). In this context, the focus is on the creation and deployment of mechanisms that efficiently and effectively facilitate the crowdsourcing and human computation process.

So far, most work on providing a survey and classification of crowdsourcing systems [11, 59, 79] adopted a business and user perspective. This section presents a systematization of task-oriented CS and HC systems from a technical domain-independent perspective.

2.3.1 Entities and Relationships

Throughout this thesis, the terms micro-task, task and complex task will be heavily employed. In order to reduce ambiguity, Figure 5 defines the relationships between each concept.

Two types of tasks are considered: micro-tasks and complex tasks. While micro-tasks are atomic computation operations, complex tasks are (linked) sets of micro-tasks (e. g. workflows of micro-tasks) with a specific purpose.

Depending on the HC or CS system, several terms from different terminologies may be employed. These terminologies refer to the entities that are present in the CS and HC process. For this systematization the following terms and entities are considered:

- Worker - a person that solves tasks;
- Community - a set of workers;
- Requester - an entity that submits jobs;
- Job - a complex task or workflow of tasks;
- Task - the specification of a task or micro-task, which may be instantiated a multiple number of times;
- Unit - an instance of a task;
- Reference Unit - an instance of a task with a known answer;
- Assignment - an assignment of a unit to a single worker;
- Answer - the given solution of a worker to a specific assignment;

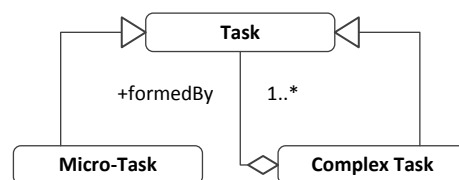


Figure 5: Relationships between the concepts Micro-Task, Task and Complex Task. Both Micro-Task and Complex Task are sub-concepts of Task.

- Qualification - a validated worker skill or expertise in a specific domain;
- Credibility - a measurement of worker performance in completed assignments;
- Workflow - the continuity of work by passing the output of one task as the input of another.

Regardless of their application domain, several subtypes of tasks were identified:

- Partition Task - a task that consists in the partitioning of a complex task into a workflow of simpler tasks;
- Aggregation Task - a task that consists in the aggregation of multiple answers given to another task or job;
- Qualification Task - a task that must be successfully completed in order to obtain a qualification;
- Grading Task - a task that consists of assessing the results of qualification tasks and usually given to highly credible and qualified workers.

The relationships between these entities are represented in the relational diagram in Figure 6. Notice that this diagram represents a generalization and a conceptual model of the analysed systems. It does not take into account implementation details.

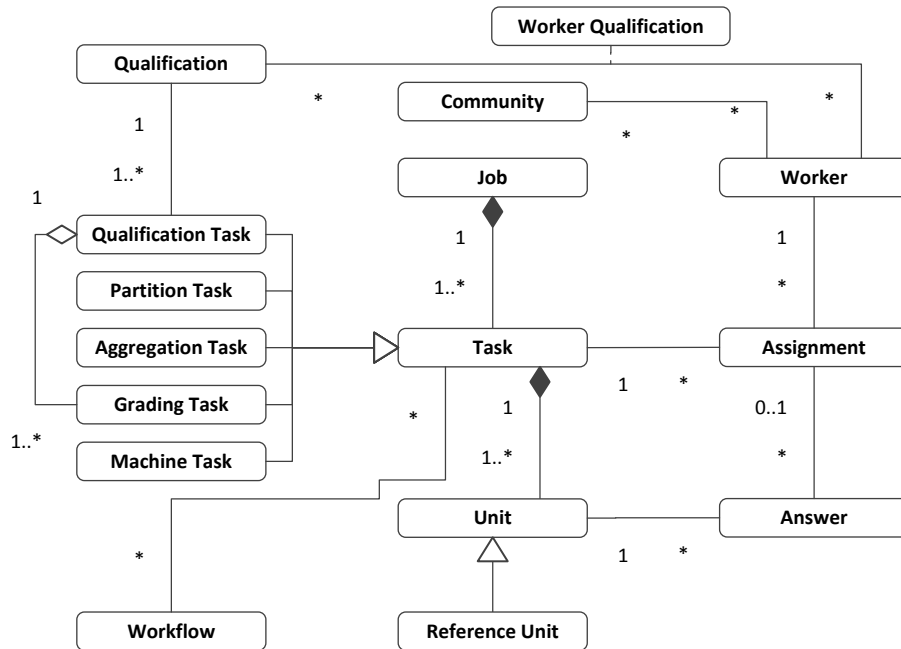


Figure 6: Entity relationship diagram of a common (complex) task CS and HC system.

2.3.2 The Process

There are three active entities in a task-oriented CS and HC process: the requester, the worker, and the system itself. Each of these entities has a different role in the process. The worker selects a task, solves it, and later receives (or not) a reward according to his/her performance. The requester and system flow of actions, however, is more complex. In Figure 7, a systematization of the whole process is provided for each of the three active entities. The depicted process is the result of the analysis of the presented state of the art CS and HC systems. Notice that when solving complex human computation tasks, current systems focus on managing workflows of simple tasks.

The overall process has three phases: the design phase, the online phase, and the conclusion phase. For each of these phases, the job can be found in different states. In the case of the design phase, the job is always in the *not ordered* state. During the online phase, the job may be either *running* or *paused*. Finally, during the conclusion phase, the job reaches its final state, which can be either *finished* or *cancelled*.

The design phase is an exclusive interaction between the requester and the system, where the requester must configure the job, design each task, and build the task workflow.

In the online phase the job is set to run. It is during this phase that the worker must act and solve the task. The requester, on the other hand, can pause the job to modify its configuration and change parts of the workflow (as in the crash and rerun strategy), resuming its execution afterwards.

Just like the job, each worker task in the workflow has an associated state. Before it is reached, the worker task is set as *not ordered*. When its execution starts, the worker task is set to *running* with the possibility of being *paused*. Finally, it can be either *cancelled* or *finished*.

Five main steps are performed during the execution of a worker task: (i) task distribution and worker selection, (ii) assignment to workers, (iii) assignment assessment, and optionally, (iv) result aggregation and (v) worker rewarding. The first three steps are executed in a loop, where the task may be re-assigned to a specific group of workers in the same community. Also, for each task, multiple assignments can be given to several workers, each requiring an assessment. When all the required assignments are solved, workers can be optionally rewarded automatically according to the results in the assignment assessment. All assignments of the same task may also be aggregated in order to provide a final answer.

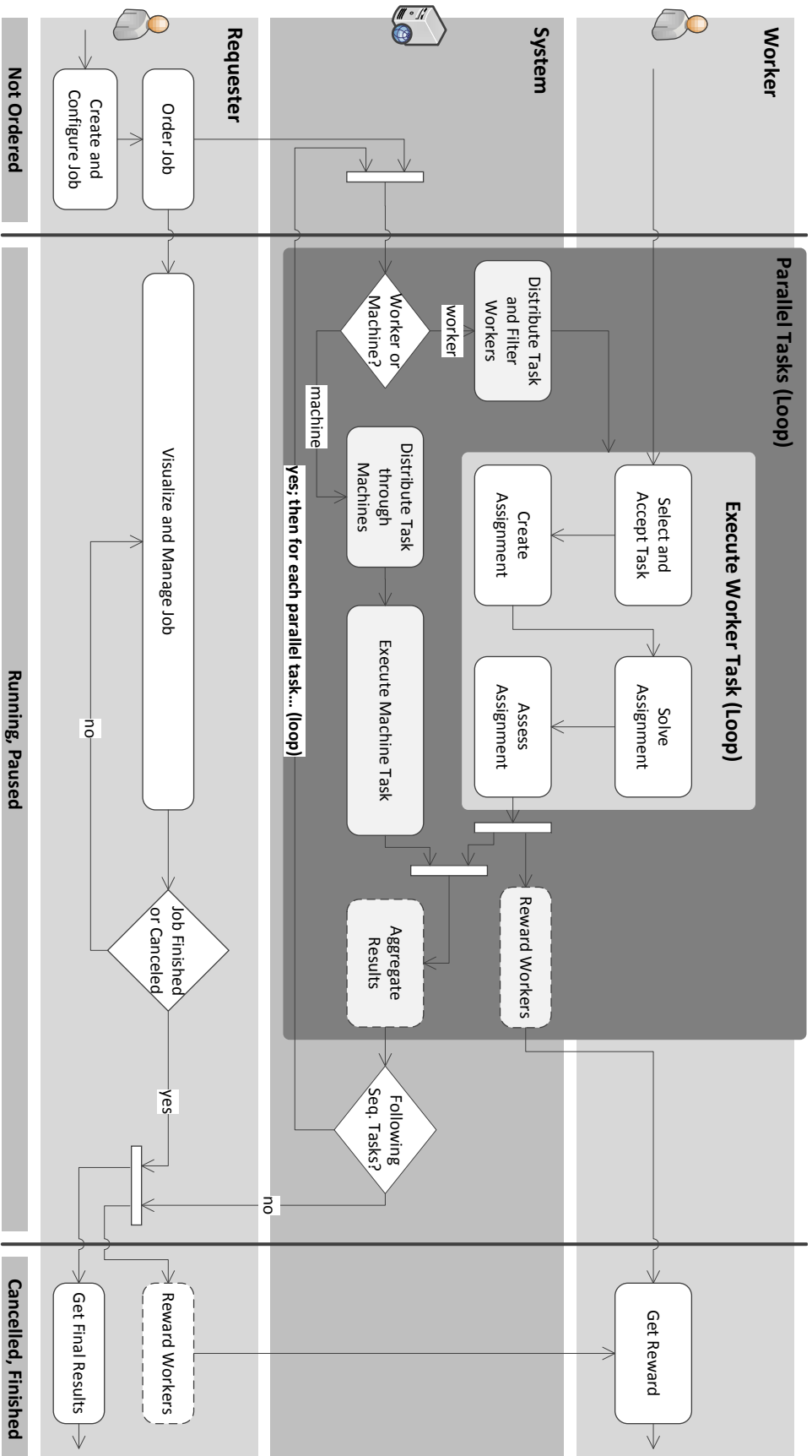


Figure 7: Process diagram of a common (complex) task CS and HC system. Dashed steps may, or may not, exist in different systems.

For each task in the workflow, the previously described process is followed. More specifically, these tasks can run either concurrently, for parallel tasks in the workflow, or sequentially. If the output of two parallel tasks is required as the input of the following task, the system must synchronize and wait for both parallel tasks to be completed before advancing to the next task.

After the workflow is finished, the conclusion phase starts, in which the requester can review and reward workers according to their performance. The complete results of the execution of the workflow are also made available to the requester.

2.3.3 *Task-oriented Approaches*

The following analysis describes micro-task CS systems that overlap with HC. The analysis is empirical since there are no available publications describing these systems.

Mechanical-Turk

Mechanical Turk is an online labour market and pioneer of the CS platforms that specialize in micro-tasks. Each requester (employer) can create Human Intelligence Task (HIT), which represent multiple micro-tasks that will be executed by workers in exchange for a small monetary reward [54].

Requesters can specify several qualification requirements for workers. These requirements often contain test forms that workers must complete in order to assess their qualifications. Additionally, profile-related requirements can be enforced such as the country of residence and accuracy in previously solved tasks.

In order to use Mechanical Turks' work force, a requester must create a new project that will wrap all the data regarding the task. A project represents a HIT template, containing common parameters (e. g. name, description, keywords, rewards per assignment, assignments per HIT, allotted time per assignment) and the layout design in Hyper-Text Markup Language (HTML). A project can be created from scratch or from a pre-defined project template.

Afterwards, a set of HIT can be submitted to Mechanical Turk using the created project. Along with this request, a Comma Separated Values (CSV) file must be uploaded. Each row of the CSV represents a HIT and contains the required data to present the HIT using the layout design defined in the project. Finally, the requester can approve

(the reward is given) or disapprove (the reward is not given) the results of worker assignments.

Mechanical Turk does not include an aggregation mechanism. According to the project configuration, the assignment results given directly to the requester may contain multiple answers to each [HIT](#).

As a worker, several assignments for the same [HIT](#) can be accepted. If a specific qualification is required for the [HIT](#), the corresponding qualification test must be passed first.

CrowdFlower

CrowdFlower is a micro-task [CS](#) platform that distributes micro-tasks over several [CS](#) channels such as Mechanical Turk and Crowd Guru. Currently, more than fifty channels are supported.

A requester starts by submitting a job to CrowdFlower. Similarly to projects in Mechanical Turk, a job is a template and an aggregation of multiple micro-tasks. These micro-tasks, instead of [HIT](#), are called units.

Several workers can work on the same unit. The result of this work for a single unit is called judgement. In this sense, one unit can have multiple judgements given by multiple workers (even one worker can give two or more judgements to the same unit).

Some of these units can be gold units. Gold units come with a reference judgement (correct answer) and are used to validate the work done by workers. If a worker gives a wrong judgement to a gold unit, their confidence degree will decrease until their judgements are not accounted for in the aggregated job results.

Besides the specification of common parameters (e. g. keywords, judgements per worker, judgements per Internet Protocol ([IP](#)) address, allowed countries, judgements per unit), a CrowdFlower job contains the user interface layout defined in CrowdFlower Markup Language ([CML](#)). The [CML](#) is a language that provides an abstraction over [HTML](#) objects and allows interaction with the unit data, which is also uploaded to CrowdFlower through a [CSV](#) file.

After supplying all the required data (units, gold units, [CML](#) form, and other configuration parameters) the requester can order judgements. In this order, one or more distribution channels can be selected (e. g. Mechanical Turk, Crowd Guru, SurveyHunt, Earn The Most).

When the job is finished, a full report is given. Additionally, aggregated, source, gold unit and worker reports are supplied in [CSV](#) format. The aggregation mechanism of CrowdFlower is based on a majority voting scheme that may exclude judgements according to the supplied gold units.

A worker participates in a job by giving judgements over sets of units presented on a single page. If allowed, a single worker can submit as many judgements as units. Taking into account the existence of gold units, this amount may even exceed the amount of units.

ShortTask

ShortTask is an online labour market similar to Mechanical Turk. The terminology, however, is slightly different. Workers are called solvers, and requesters are called seekers.

A seeker can create a task template, which can be used to order multiple tasks (equivalent to [HIT](#) in Mechanical Turk). These tasks will then be assigned to solvers.

Although the terms task and assignment are used, their application is not coherent. Furthermore, even though there is a parameter for selecting the amount of assignments in the task template, requesting multiple answers for one task when ordering was not possible while experimenting with ShortTask.

MicroWorkers

MicroWorkers is a [CS](#) platform that focuses solely on the distribution of a micro-task amongst multiple workers. Unlike in Mechanical Turk and CrowdFlower, the process of building the job is significantly less structured and simpler.

Instead of an interface for solving tasks structured with a markup language, the worker is given a set of instructions in natural language. These instructions are fixed for each job, meaning that submission will result in multiple workers performing the same task.

Given these limitations, jobs in MicroWorkers mostly involve work such as filling surveys, searching, rating, clicking, bookmarking, commenting, downloading, and installing applications. These are often associated with forums and websites such as Google, YouTube, Facebook and Twitter.

Probably due to the nature of its jobs, MicroWorkers does not implement any specific worker selection mechanism besides the possibility of targeting specific countries. Also, no automatic worker assessment is performed. Employers may instead ask for the submission of proof regarding the task completion.

CloudCrowd

CloudCrowd is a CS platform originally implemented as a Facebook application. Unlike the previously described platforms, users can only register as workers.

Comparatively, CloudCrowd is more selective regarding workers and their expertise. In order to work on projects, workers need to get credentials. These credentials are given after successfully solving specific tests with multiple levels of difficulty. Additionally, a credibility score is given to the worker. The credibility score value can increase or decrease according to the worker credentials and work feedback (e. g. incorrect answers).

In order to assess the credibility of workers, CloudCrowd uses check tasks. Check tasks are tasks with known answers, similar to gold units in CrowdFlower.

Workers can browse a list of projects and credentials, grouped by the following types: writing, editing, categorization, research, data entry, translation, and other. For each project, the required credibility and credentials are presented, along with the monetary reward and availability. The availability represents the amount of tasks available to be solved in the project.

From the analysis of the CloudCrowd worker interface, it is unknown if the same task is given to multiple workers or if there is any aggregation mechanism available to requesters.

2.3.4 *Workflow-oriented Approaches*

The following analysis describes systems that perform the CS of complex tasks. For some of these systems the implementation was not available, leading to a study based only on related scientific publications.

Turkit

Mechanical Turk focuses on independent tasks that can be executed in parallel. However, it does not support the creation of workflows of dependent tasks. TurKit is an Application Programming Interface (API), built on top of Mechanical Turk, for running iterative tasks that provides an environment for the creation of workflows that connect multiple dependent tasks [37].

Using the crash and rerun model, it provides an abstraction over the specificities and synchronization issues of Mechanical Turk, allowing the developer to focus on imperative ordinary function calls.

The crash and rerun model follows the premise that it is cheap to rerun an entire program up to the point where it crashed, as long as it runs locally. For remote and costly operations (e. g. HIT requests) the results must be stored in a database so that they will be accessible in future reruns.

The TurKit API is implemented in Java and allows the implementation of Mechanical Turk workflows in a JavaScript crash and rerun environment. This environment features a set of function directives for an easy and incremental implementation of scripts.

The TurKit provides a development environment that is available both as a stand-alone application and as a Web application. The Web application runs on the Google App Engine.

Turkomatic

Turkomatic follows a divide-and-conquer approach to plan work featuring micro-task workflows partially designed by workers. It works over Mechanical Turk and is defined as “a crowdsourcing interface that consults the crowd to design and execute workflows based on user requests” [32].

Workers start by dividing the requested task into subtasks that will be solved by other workers. This process can be iterative, generating a tree of subtasks. The final results are later combined by workers into an adequate solution.

Turkomatic allows the requester to manage the generated workflow and its execution through a visual workflow editor.

CrowdForge

CrowdForge is a general purpose framework for distributed processing that provides scaffolding for complex human computation tasks [27]. The approach features a set of task coordination strategies that allow multi-level and dynamic partitioning of tasks, the specification of task workflows, quality control tasks and aggregation of results.

Finding inspiration in Googles' Map Reduce framework, CrowdForge defines three types of subtasks: (i) partition tasks, (ii) map tasks and (iii) reduce tasks. Partition tasks divide a larger task into smaller subtasks. In map tasks, one or more workers process a task. Finally, in reduce tasks the processing results of multiple workers are merged into a single output.

The CrowdForge prototype presented in [27] consists in a Web application allowing the design of complex tasks, along with a back-end server that interacts with Mechanical Turk. The system manages a workflow of Mechanical Turk HIT templates, which can represent partition map or reduce tasks.

CrowdWeaver

TurKit, CrowdForge and Jabberwocky provide an environment for the design and execution of complex CS tasks through structured languages and non-visual representations. With the current trend on micro-task workflows, work on their management and visualization has started to emerge. Kittur et al. [28] state that one of the major issues faced by employers working with crowds lies in the complexity of linking tasks and forming workflows. In this sense, they identify several challenges in visually managing CS workflows and present a system for visualization and management of complex tasks, entitled CrowdWeaver.

CrowdWeaver works on top of CrowdFlower and features the creation and monitoring of task workflows, the management and reuse of templates with human and machine tasks, the tracking and notification of crowd factors such as price and quality, and support for real-time experimentation. The interface provides a mental representation of the task workflow, which can be saved and re-used in further instantiations of the workflow.

Several machine tasks, which mainly manipulate input and output, are supported. These include divide, concatenate, pair, and permute. Additionally, custom machine tasks are allowed.

Jabberwocky

Similarly, Jabberwocky [1] also employs the MapReduce approach in a framework featuring a high-level abstraction task modelling language. It allows the modelling of complex tasks and workflows in which the advantages of multiple worker communities can be harnessed. These communities can be local or found in social networks and other CS systems. In the case of local communities, workers can be identified during the CS process. For social network communities, expertise data may be extracted from the social network API.

Jabberwocky is formed by three layers: the (i) base layer is called Dormouse, followed by the (ii) ManReduce layer, with the (iii) Dog layer on top. The Dormouse layer provides an abstraction over human (crowd workers) and machine computational units. Unlike other CS frameworks such as Mechanical Turk and CrowdFlower, each computational unit registered under Dormouse can be uniquely identified during workflow executions. Besides featuring its own worker community, Dormouse can crowdsource tasks to external CS platforms.

The ManReduce layer is a programming framework, written in Ruby, responsible for facilitating complex data processing tasks in Dormouse. It features map and reduce steps which can be computed by either humans or machines.

The top layer, called Dog, represents an abstraction scripting language for modelling tasks. Dog works over the low-level ManReduce framework and focuses on reusability, maintainability and ease-of-use.

2.3.5 *Comparative Analysis*

While featuring the integration of a wide range of CS platforms, CrowdFlower also features the CML, an abstraction language for building task graphical user interfaces. Other analysed platforms only support direct HTML usage.

According to our analysis, Jabberwocky represents a highly complete CS platform compared to the others. It adds several features to CrowdForge, such as:

- Worker profiles and social networks;
- Abstraction over human and machine computational units;
- Multiple worker sources (e.g. e-mail groups, social networks, external CS platform communities).

CloudCrowd has a rigorous and complete assessment system compared to most CS platforms, incorporating the benefits of both Mechanical Turks' and CrowdFlowers' assessment systems.

Table 1 contains an overview and comparison of the analysed CS systems. The *relies on* column indicates if the system is able to relay the CS process to other systems. The *complex tasks* column describes, if available, how complex tasks are formed and handled. The third column, *task methodologies*, enumerates different methodologies employed by the system in designing, maintaining, and executing tasks. The *worker assessment* column refers to how the CS system evaluates the quality of the work performed by workers. Finally, the *aggregation* column describes, if available and applicable, automatic result aggregation procedures that merge the work of several workers.

The terminology employed in the CS domain often varies from platform to platform. Table 2 wraps the terminology used in four different CS platforms.

The first concept found in Table 2 represents a wrapper of the whole CS process. It contains all the data required for the execution of the task or task workflow, including the input and output data. A task (e. g. tag an image) can have several units of work (e. g. the actual images to be tagged). This concept is represented in the second row of Table 2. The following concept depicts assignments of specific units to workers (the ones who solve the task). For each assignment, the worker must submit an answer (fourth row concept).

In some cases, for quality control purposes, units are submitted by the requester with an already known correct answer. This kind of unit is present in the fifth row of Table 2. Furthermore, the requester (the one who requires a certain task to be solved) can specify expertise requirements that workers must satisfy in order to be electable for solving the task. The terminology for this concept can be found in the last row of Table 2.

2.4 ONTOLOGY OF TASK-ORIENTED SYSTEMS

Although classifications of CS systems already exist [11, 59, 79], they often focus on dimensions that are domain-specific or of interest from a business standpoint. In this sense, a new systematization of these classifications is provided, integrating an analysis of the previously described systems.

The presented classification, in Table 3, focuses on technical and domain-independent dimensions of systems that fit under both the CS and HC definitions. It is built

SYSTEM	RELIES ON	COMPLEX TASKS	TASK METHODS	WORKER ASSESSMENT	AGGREGATION
MTurk	Self	No	Templates	Qualification Tests	Manual
CrowdFlower	Several	No	Templates	Gold Units	Yes
ShortTask	Self	No	Templates	Manual	Manual
MicroWorkers	Self	No	Templates	Manual	N/A
CloudCrowd	Self	-	-	Credential Tests and Credibility	-
CrowdForge	MTurk	Workflows	Map Reduce	(MTurks')	Yes
Jabberwocky	Several	Workflows	Map Reduce	User Profiles	Yes
Turkomatic	MTurk	Workflows	Divide and Conquer	(MTurks')	Yes (Workers)
Turkit	MTurk	Workflows	Crash and Rerun	(MTurks')	Yes (Workers)

Table 1: Comparison of CS systems.

MEANING	MTURK	CROWDFLOWER	SHORTTASK	CLOUDCROWD	MICROWORKERS
Wrapper of CS data and tasks	Project	Job	Task Template	Project	Campaign or Job
Unit of the task to be solved	HIT	Unit	Task	Task	Task
Assignment of a unit to a worker	Assignment	-	Assignment	-	-
Answer given to an assignment	Answer	Judgement	-	Answer	-
Reference unit with correct answer	N/A	Gold Unit	N/A	Check Task	N/A
The worker	Worker	Worker	Solver	Worker	Worker
The requester	Requester	Requester	Seeker	-	Employer
Worker expertise requirement	Qualification	N/A	N/A	Credentials	N/A

Table 2: Terminologies employed by CS systems.

as a poly-hierarchy of categories, meaning that a CS system can fit under a number of these categories. As a reference for the classification, the following dimensions for task-oriented systems were identified:

- Nature of collaboration [11];
- Architecture [11];
- Worker selection;
- Worker assessment and quality control;
- Worker motivation;
- Task creation and configuration;
- Task management;
- Task execution;
- Task result aggregation.

2.4.1 *Nature of Collaboration*

The explicit and implicit categories present in the *nature of collaboration* dimension are introduced by Doan et al. [11]. In explicit CS systems the users are aware that they are working towards solving specific tasks. Some examples of explicit CS systems include Mechanical Turk, CrowdFlower and ShortTask.

Implicit CS systems usually retrieve work behind a system with a different purpose. These include games such as the ESP Game [76], Tag-a-Tune [59], and Foldit [9]. ReCAPTCHA [36] is also an example of an implicit CS system where users work as book and document digitizers by filling Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA) forms. Since CS games fit under the implicit category, the *game* sub-category has been included in the classification.

2.4.2 *Architecture*

The architecture dimension encompasses the *stand-alone* and *piggyback* categories, which are also introduced by Doan et al. [11]. A stand-alone CS system is independent of other CS systems. It has its own worker community and does not require the distribution of tasks among other systems (e. g. Mechanical Turk).

DIMENSION	CATEGORY	SUB-CATEGORY
Nature of Collaboration	Explicit	
	Implicit	Game
Architecture	Stand-alone	
	Piggyback	
Worker Selection	Community Scope	Multiple Community
		Single Community
	Worker Scope	Assessment Filtering
		Expertise Test Filtering Profile (Social) Filtering ↙ Demographic Filtering
Worker Assessment and Quality Control	Manual	
	Automatic	Reference Units Assessment Tasks
Worker Motivation	Monetary	
	Enjoyment	
	Reputation	
Task Creation and Configuration	Instantiation	Manual
		Templates
	UI Construction	Plain Text
		HTML Proprietary Language
	Data Structure	Plain Text CSV, JSON (No Schema)
Task Management	Non-Visual	
	Visual	Construction Monitoring
Task Execution	Simple	
	Complex (Workflows)	Crash and Rerun
		Divide and Conquer ↙ Partition-Map-Reduce
Task Result Aggregation	Manual	
	Automatic	Voting Scheme Assessment-based Crowdsourced

Table 3: Classification of CS and HC systems according to different dimensions (↙ means that the following is a sub-category of the above category).

A piggyback CS system relies on external worker communities. They usually focus on the process of building the task and distributing work between other CS systems. One of the best examples of such a system is CrowdFlower.

The given definition for these two categories is not disjoint. In this sense, a CS system, such as Jabberwocky, fits under both categories.

2.4.3 Worker Selection

When a task is published, it becomes available to a certain amount of workers. These workers usually belong to at least one community, and may be filtered according to different characteristics such as their profile information, demographics (e. g. country of residence), or expertise. In this sense, six types of worker selection strategies that act in two different scopes (the community scope, and the worker scope) were identified (see Table 4).

The community scope encompasses CS systems that either rely on multiple communities of workers or on a single worker community. These two categories are disjoint.

In the worker scope there are four categories: *assessment filtering*, *expertise test filtering*, *demographic filtering*, and *profile (social) filtering*. In assessment-based filtering, the task is distributed between workers that satisfy requirements regarding reputation and performance in previously solved tasks. Expertise test filtering consists in providing test tasks that assess worker expertise in a specific domain of knowledge, or in performing a specific type of task. The measurement of expertise allows the enforcement of a minimum expertise value for the worker to be able to participate.

CATEGORY	SUB-CATEGORY	EXAMPLES
Community Scope	Multiple Communities	Jabberwocky
	Single Community	MTurk, ShortTask
Worker Scope	Assessment Filtering	MTurk, CloudCrowd
	Expertise Test Filtering	MTurk, CloudCrowd
	Profile (Social) Filtering	
	↙ Demographic Filtering	CrowdFlower

Table 4: Classification of CS systems according to the worker selection dimension (↙ means that the following is a sub-category of the above category).

Demographic filtering selects workers that satisfy certain statistical characteristics of attributes such as age and country of residence. Profile (social) filtering represents a superset of demographic filtering. It enforces requirements regarding the worker personal and social profile, which may include demographic attributes. One example is the enforcement of a minimum or maximum value for social worker relationship closeness [77].

2.4.4 Worker Assessment and Quality Control

Assessing the performance of workers in specific tasks or in general is an important dimension of CS. Worker assessment can be employed in different phases and in other dimensions of the CS process such as in worker selection and task result aggregation, often having a significant impact on the quality of the final task results.

According to the current state of the art, assessment strategies are either manual (performed by the requester) or automatic (performed by the CS system).

As presented in Table 5, two automatic assessment strategies were identified: *reference units*, and *assessment tasks*. An assessment through reference units consists of using test tasks with known answers to evaluate worker performance. Another strategy is to crowdsource the assessment of the answers to other workers through an assessment task.

2.4.5 Worker Motivation

For users to participate and remain as workers, they require some kind of motivation, usually given by the CS system or the requester.

CATEGORY	SUB-CATEGORY	EXAMPLES
Manual		ShortTask, MicroWorkers
Automatic	Reference Units	CrowdFlower
	Assessment Tasks	Turkomatic, CrowdForge

Table 5: Classification of CS systems according to the worker assessment and quality control dimension.

Most [HC](#) and [CS](#) systems provide small monetary rewards. Others, however, use different means of motivating workers. While in [CS](#) games the motivational aspect is usually recreation and enjoyment, reputation has also been proven a viable source of motivation [58].

In this sense, three worker motivation strategies are identified as the following non-disjoint categories:

- Monetary (e. g. MTurk and CrowdFlower);
- Enjoyment (e. g. Tag-a-Tune and the ESP Game);
- Reputation (not so widely exploited since workers often remain in anonymity).

2.4.6 *Task Creation and Configuration*

The task creation and configuration dimension encloses all the special characteristics of the task creation and configuration process. So far, it mainly consists of the definition of the User Interface ([UI](#)), the upload of data, and the configuration of the task request through parameters such as monetary reward per unit and target worker residence countries.

Referring to this dimension, a classification is presented in [Table 6](#) that takes into account how the task instance is created, how the construction of the worker [UI](#) is performed, and which data structures are consumed by the [CS](#) system. None of the included categories and sub-categories are disjoint.

Most [CS](#) systems allow both the manual and template-based instantiation of the task. Also, the [CSV](#) format is commonly used in [CS](#) systems to allow requesters to upload task data.

2.4.7 *Task Management*

So far, the management of tasks has been performed through simple, mostly textual, Web [UI](#). With the advent of several complex task [CS](#) systems, however, a need for visually building and managing task workflows has emerged.

Turkomatic is one of the first [CS](#) systems introducing task visualization by presenting a complex task construction environment where the requester is able to visually build a workflow of tasks. More recently, CrowdWeaver takes complex task visualization a

CATEGORY	SUB-CATEGORY	EXAMPLES
Instantiation	Templates	MTurk, ShortTask
	Manual	CrowdFlower, MTurk
UI Construction	Proprietary Language	CrowdFlower
	HTML	MTurk
	Plain Text	MicroWorkers
Data Structure	CSV, JSON (No Schema)	CrowdFlower, MTurk
	Plain Text	MicroWorkers

Table 6: Classification of CS systems according to the task creation and configuration dimension.

step further by providing an environment not only for visually building workflows of tasks but also for monitoring their progress.

Taking into account the current state of the art, four categories for the task management dimension are presented in [Table 7](#).

In this dimension, only the *non-visual* and *visual* categories are disjoint. The *construction* category refers to CS systems that provide a visual UI for building tasks. The *monitoring* category, on the other hand, refers to CS systems that allow the visual monitoring of the task progress in real time.

2.4.8 Task Execution

The task execution dimension refers to how the task is structured and processed by the CS system. Under this dimension, a task can either be simple or complex (a workflow of simple tasks). For complex tasks, several strategies are employed. These are present in the classification with the sub-categories shown in [Table 8](#).

CATEGORY	SUB-CATEGORY	EXAMPLES
Non-Visual		CrowdFlower, MTurk
Visual	Construction	CrowdWeaver, Turkomatic
	Monitoring	CrowdWeaver

Table 7: Classification of CS systems according to the task management dimension.

CATEGORY	SUB-CATEGORY	EXAMPLES
Simple		CrowdFlower, MTurk
Complex	Crash and Rerun	Turkit
	Divide and Conquer	Turkomatic
	↙ Partition-Map-Reduce	CrowdForge, Jabberwocky

Table 8: Classification of CS systems according to the task execution dimension (↙ means that the following is a sub-category of the above category).

The *crash and rerun* category includes systems that allow modifying and restarting the execution of the workflow without additional CS costs. Divide and conquer approaches try to split the task into smaller units before requesting a final solution. Partition-map-reduce are a specific case of divide and conquer approaches that include a result aggregation phase.

2.4.9 Task Result Aggregation

Often, the same task is solved by multiple workers, leading to several solutions for the same task. Despite providing redundancy, it requires the aggregation and processing of the resulting data in order to reach a final and unique solution.

The aggregation of results can be either manual (performed by the requester of the task) or performed automatically using an aggregation strategy. Commonly used aggregation strategies include the application of voting schemes and filtering through worker assessment. CrowdFlower employs both of these aggregation strategies. It uses a majority voting scheme to select results that are most likely to be correct, excluding workers with poor assessment values (those that gave wrong answers to check tasks).

The results of the algorithms employed during worker selection can also provide important data for result aggregation. Both CloudCrowd and Mechanical Turk implement mechanisms to assess expertise (qualifications or credentials) in order to filter workers. Giving more prominence to the answers of workers with higher degrees of expertise could lead to better results.

Another approach is to crowdsource the result aggregation as an aggregation task. This strategy is employed in systems that use the partition-map-reduce task execution strategy.

Table 9 presents the categories identified for the task result aggregation dimension.

CATEGORY	SUB-CATEGORY	EXAMPLES
Manual		ShortTask
Automatic	Voting Scheme	CrowdFlower
	Assessment-based	CrowdFlower, CloudCrowd
	Crowdsourced	CrowdForge, Jabberwocky

Table 9: Classification of CS systems according to the task result aggregation dimension.

2.5 CHALLENGES

Since the appearance of the first CS systems for micro-tasks, many more have emerged. Their continuous use has led to several experiments and studies that often focus on user motivation and quality control [80].

User motivation has been addressed in several ways, from providing enjoyment and relying on altruism, to giving monetary rewards [15]. The latter has been the subject of some criticism, due to the creation of cheap labour marketplaces [21]. In the specific case of task CS systems, the current monetary rewards are not sufficient to be a primary source of income, and often they are not enough to serve as a motivational factor [48, 54]. In these cases, workers actually participate out of altruism, curiosity, or simply to keep themselves busy. Still, motivating and retaining workers over time remains a challenge for any CS system.

The advent of CS of complex tasks has brought new challenges to different dimensions of the CS process, including specification, flow control, quality control, and visualization. In the specific case of quality control, and due to the presence of a micro-task workflow structure, a deviation or error in one task can accumulate with those of the following tasks.

Besides quality control, the flow control assumes special relevance. In fact, the CS of complex tasks establishes requirements regarding worker selection that are often discarded in simple micro-task CS systems. One of these requirements is worker identity. The fact that most CS systems regard the micro-task as the top-level unit of work, leads to loss of identity information from one micro-task to the other. This can easily become a challenge when the worker, as an identifiable individual, is required to participate in different steps of the task workflow.

Another challenge when dealing with complex tasks is the aggregation and visualization of results. At some point, since multiple micro-tasks are involved in a workflow,

tracing a specific result and asserting the causes and facts that led to it becomes a necessity. Currently, both the results presented by most CS systems and the input requested from workers are poorly structured and, in some cases, in natural language. Since the output of a micro-task might be the input given to the execution of a computer algorithm micro-task, both the specifications (structure) of the task and the task domain of knowledge must be understandable and interpretable by both humans and machines. Furthermore, there is a necessity for a specification that is able to capture iterative steps formed by complex flow conditions.

Enforcing structured machine-readable data also facilitates the implementation of strategies for tracing workflow results. Still, special care is needed so that such a structure does not increase the complexity and cumbersome nature of solving tasks from the worker's perspective.

Overall, the following main challenges were identified:

- The representation of micro-task workflows, namely:
 - Full structure and semantics of the operations, input and output of micro-tasks;
 - Partitioning and establishing different granularities for micro-tasks;
 - Controlling the flow of micro-tasks;
 - Understandable and interpretable representations for both humans and machines;
 - Fully integrated dynamic and static domain dimensions;
 - Flexible, extensible and shareable representations;
- Human-machine interaction, namely:
 - Specifying worker selection requirements for different phases of the workflow;
 - Identifying workers and requesting their participation in multiple micro-tasks;
 - Providing relevant contextual information;
- Quality control, namely:
 - Assessing and reducing the impact of low quality answers across the workflow;

- Using (social) profile information to assess the worker’s expertise and relevance to a particular micro-task;
- Defining metrics and measures to assess the performance of workers;
- Visualization and reporting, namely:
 - Providing visual representations of micro-task and workflow structures;
 - Assembling the final results and providing graphically represented statistics.

2.6 SUMMARY

Over the last decade, **HC** and **CS** have been merged to create multiple systems for problem solving on a wide scale. These systems are starting to facilitate the man-computer symbiosis envisioned by Licklider [35] in the 1960s.

The increasing popularity of **CS** has led to a variety of commercial and non-commercial applications in different domains. Among these applications, a small set of **CS** platforms oriented towards problem (task) solving has emerged. These platforms regard the problem (or job) as sets of micro-tasks that can be solved redundantly in one step by multiple users. In complex jobs where a workflow of micro-tasks is required, new challenges emerge. This new trend in **CS** has led to some interesting workflow-oriented systems such as CrowdForge and Jabberwocky.

Despite the fact that workflow and **CS** approaches have emerged from different demands and requirements, an analysis of the current state of the art leads to the conclusion that they share many commonalities and tendencies.

While workflow-definition approaches started with highly structured and strict representations of activities and processes easily handled by machines and execution engines, crowdsourcing is popular for its relatively loose and unstructured representation of tasks and their input and output data. Although this later representation is preferable to humans, it is difficult to interpret and understand by machines. The trend, however, for both crowdsourcing and workflow activity-based approaches has been to provide a workflow representation mechanism that is both understandable by machines and as closer to the human conceptual level as possible.

Ontologies as a formal representation mechanism are currently “the best answer to the demand for intelligent systems that operate closer to the human conceptual level” [52]. While providing structure and semantics, ontologies can also leverage:

- Template re-use and extensibility;
- Semi-automatic generation of worker interfaces;
- Contextual information in complex workflows (tackling flow and quality control);
- Justifications for results using common reasoning algorithms (tackling result traceability in flow control);
- Semantics for further automatic processing of data (tackling complex task specification of the task and the task domain).

Detailed representations (ontologies) of micro-tasks and their dependencies through the specification of their domain (input, output and context) can be analysed to present relevant contextual information to workers and detailed structured reports to requesters [43]. Although this is not particularly interesting in single task CS systems, it is relevant when dealing with complex task workflows and their input/output dependencies.

Part II

THE COMPFLOW APPROACH

APPROACH OVERVIEW

Analogous to the analysed **CS** and **HC** background, an approach to the ontology-based representation, instantiation and execution of workflows of micro-tasks (simply referred to as tasks from now on) is proposed. This approach is entitled, **CompFlow**.

As depicted in **Figure 8**, the **CompFlow** approach envisages two steps:

- i) Task-definition (the task representation) and workflow-definition (the workflow representation) construction (i.e. the conceptualization and implementation step);
- ii) The workflow-definition instantiation and execution.

During the workflow-definition construction (i), the creator (i.e. the actor building the representation) must clearly represent the activities involved in the workflow through a semantic model of the input and output data, i.e. a workflow-definition ontology must be created. The workflow-definition ontology is built as an extension of both a domain ontology and the **CompFlow** ontology.

During the workflow-definition instantiation and execution (ii), workflow-definition ontologies can be instantiated multiple times and executed by any workflow engine that is able to interpret the **CompFlow** ontology and apply the ground rules established

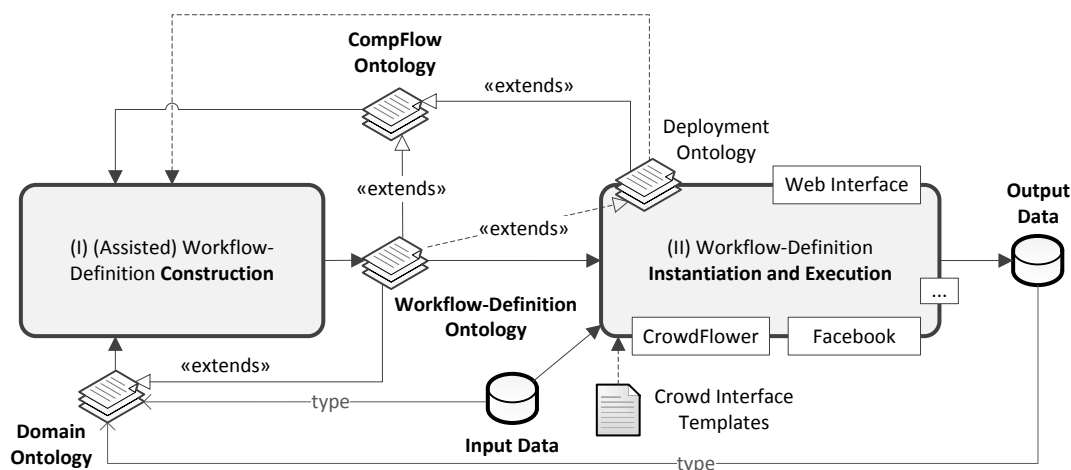


Figure 8: Overview of the **CompFlow** approach (dashed connections are optional).

by the proposed method. Besides the workflow-definition ontology, this step is fed with an input dataset and a set of crowd interface templates (required only for human workers).

3.1 STATIC AND DYNAMIC ONTOLOGIES

There are three main ontologies in the CompFlow approach: the domain ontology (representing the static domain dimension), the CompFlow ontology (representing the dynamic task and workflow dimension), and the workflow-definition ontology (integrating both of the previous dimensions).

A workflow-definition ontology is the representation or model of a workflow, which must always extend the CompFlow ontology and any available domain ontology.

The domain ontology, on the one hand, represents static knowledge about a particular domain (e. g. the input and output of tasks), typically lacking the representation of actions or processes that modify or increase this knowledge. A great and growing amount of these kind of ontologies can be found throughout the Semantic Web.

The CompFlow ontology, on the other hand, contains a high-level abstract representation of the dynamic features that can be associated with any domain of knowledge. More specifically, the CompFlow captures the abstract structure and semantics required to represent elements such as workflows, tasks, events and transitions.

As exemplified in [Figure 1](#), dynamic features can be attributed to static domain concepts such as *Text* through the concepts found in the CompFlow ontology, i. e. *Unit* and *Response*. Thus, the two dimensional concepts *OriginalText* and *TranslatedText* emerge with additional semantics: *OriginalText* represents the input of a translation task and *TranslatedText* represents its output.

3.2 CONSTRUCTION OF THE WORKFLOW-DEFINITION

The construction of a workflow-definition ontology can be completely manual or performed by the creator with the assistance of a semi-automatic construction process.

In order to create workflow-definitions, a method that establishes a set of ground rules based on DL is proposed. This method is referred to as *the workflow-definition method*. The workflow-definition method exploits the expressiveness and semantics of DL to represent tasks, workflows, and their input and output data, using a single rep-

resentation language. Consequently, the construction of a workflow-definition is the construction of an ontology, following the workflow-definition method.

The semi-automatic construction process employs the workflow-definition method. It analyses the structure and semantics of the current workflow-definition ontology and its domain ontology to assist and propose further construction steps to the creator.

3.3 INSTANTIATION AND EXECUTION OF THE WORKFLOW-DEFINITION

The instantiation and execution of a workflow-definition requires a CompFlow-based workflow engine implementation.

CompFlow-based workflow engines may integrate and dispatch the execution of the tasks to external micro-task execution communities such as Mechanical Turk and CrowdFlower. Alternatively, they can provide their own task resolution interfaces, which may interact with external social networks such as Facebook and LinkedIn.

Optionally, an extension of the CompFlow ontology, associated with the instantiation and execution engine deployment (i. e. a deployment ontology) may be used by workflow-definition ontologies to represent the dynamic task and workflow dimension. The main purpose of the deployment ontology is to extend the CompFlow ontology with new concepts (e. g. new types and groups of workers and interfaces) that are dependent on a particular deployment of the instantiation and execution engine.

For workflow-definitions containing task representations that must be solved by human workers, crowd interface templates must be supplied along with the workflow-definition ontology. These templates are used to (i) present the task data to the worker, and (ii) to retrieve the submitted response.

3.4 SUMMARY

This chapter provides an overview of the CompFlow approach, which is fully described in the following chapters. In this sense, the presentation of the CompFlow approach starts with a formal introduction to ontologies and DL. Afterwards, each of the steps considered in the CompFlow approach are analysed and discussed in detail.

ONTOLOGIES

Although several definitions have been given to ontology, in the context of this work an ontology is defined as “a formal, explicit specification of a shared conceptualization” [69]. A conceptualization is a set of entities (e.g. objects, concepts, relations or roles) presumed to exist in a particular domain. It is formal because it is supported by unambiguous formal logics, explicit since it makes domain assumptions explicit for reasoning and understanding, and shared for its ability to provide consensus.

Notice that some of the benefits of using ontologies as a workflow representation mechanism are already explicitly mentioned in this definition, i.e.:

- They allow the application of reasoning algorithms;
- They are easily shared;
- They provide consensus;
- They capture the structure and semantics of data while retaining the ability to be interpretable by both humans and machines.

This chapter defines ontologies through DL and establishes all the related formalisms required to define the CompFlow approach. According to this formal representation, a domain ontology example is given. Finally, a discussion and overview of the CompFlow ontology is presented.

4.1 ONTOLOGIES IN DESCRIPTION LOGICS

The CompFlow approach is formally defined through a DL language that is able to express role subsumptions and role transitive closures. Concept descriptors (C , D), role descriptors (R) and nominal descriptors (a , b) are defined according to the syntax rule in Table 10.

$C, D \longrightarrow$	$A \mid$	(atomic concept)
	$\top \mid \perp \mid$	(universal and bottom concepts)
	$a, b, \dots \mid$	(one-of concept)
	$C \sqcup D \mid$	(union)
	$\forall R.C \mid$	(universal quantification)
	$\exists R.C \mid$	(existential quantification)
	$= nR.C \mid \geq nR.C \mid \leq nR.C \mid$	(quantified cardinality restriction)
	$R : a$	(fills nominal restriction)
$R, S \longrightarrow$	$AR \mid$	(atomic role)
	$R^+ \mid$	(role transitive closure)
	$R^- \mid$	(inverse role)
	$R \sqcup S$	(role union)
$a, b \longrightarrow$	$n \mid$	(individual)
	v	(datatype value)

Table 10: Syntax rule of the employed description logic language.

4.1.1 Definition of Ontology

An ontology is defined as a 4-tuple $O := (DL, TBox, RBox, AnBox)$, containing descriptors according to the syntax rule in Table 10, where:

- DL is the set of all *concept*, *role* and *nominal* descriptors;
 - $DLC \subseteq DL$ is the set of all *concept* descriptors;
 - $DLAC \subseteq DLC$ is the set of all *atomic concept* descriptors, $C \longrightarrow A$;
 - $DLR \subseteq DL$ is the set of all *role* descriptors;
 - $DLAR \subseteq DLR$ is the set of all *atomic role* descriptors, $R \longrightarrow AR$;
 - $DLn \subseteq DL$ is the set of all *individual* descriptors, $a \longrightarrow n$;
 - $DLv \subseteq DL$ is the set of all *datatype value* descriptors, $a \longrightarrow v$;
- $TBox$ is the *terminological box* containing equivalence and subsumption relationships between concept descriptors, $X \longrightarrow C \sqsubseteq D \mid C \equiv D$, where $C \in DLC$ and $D \in DLC$;
- $RBox$ is the *role box* containing equivalence and subsumption relationships between role descriptors, $X \longrightarrow R \sqsubseteq S \mid R \equiv S$, where $R \in DLR$ and $S \in DLR$;

- *AnBox* is an *annotation box* containing assertions of the form $X \longrightarrow R(C, v)$, where $C \in DL$, $R \in DLR$ and $v \in DLv$.

The Terminological Box or Taxonomy (**TBox**) and the Role Box (**RBox**) contain the vocabulary of the knowledge domain, consisting on concepts and roles. The vocabulary is defined through terminological and role axioms such as subsumptions (\sqsubseteq) and equalities (\equiv). The Annotation Box (**AnBox**) contains assertions about concept and role descriptors.

The expressions in [Table 11](#) are part of this definition, where $\lfloor X \rfloor_\rho$ is a DL axiom X in the **TBox** ρ .

Likewise, the expression $\lfloor X \rfloor_\rho$ can also be used to represent axioms inside a **RBox** ρ (where X must follow the syntax allowed in the **RBox**) or inside an **AnBox** ρ (where X must follow the syntax allowed in the **AnBox**).

$(C \xrightarrow{R} D)_\rho \Rightarrow$
$\lfloor C \sqsubseteq \exists R.D \rfloor_\rho \vee \lfloor C \sqsubseteq \forall R.D \rfloor_\rho \vee (\exists n : \lfloor C \sqsubseteq = nR.D \rfloor_\rho \vee \lfloor C \sqsubseteq \geq nR.D \rfloor_\rho \vee \lfloor C \sqsubseteq \leq nR.D \rfloor_\rho)$, which defines if a role R restriction with domain C and range D exists
$(C \xrightarrow[X]{R} D)_\rho \Rightarrow$
$\lfloor C \sqsubseteq X \rfloor_\rho \wedge (\lfloor X \equiv \exists R.D \rfloor_\rho \vee \lfloor X \equiv \forall R.D \rfloor_\rho \vee (\exists n : \lfloor X \equiv = nR.D \rfloor_\rho \vee \lfloor X \equiv \geq nR.D \rfloor_\rho \vee \lfloor X \equiv \leq nR.D \rfloor_\rho))$
$minC(C \xrightarrow{R} D)_\rho =$
$n : \lfloor C \sqsubseteq = nR.D \rfloor_\rho \vee \lfloor C \sqsubseteq \geq nR.D \rfloor_\rho \vee (\lfloor C \sqsubseteq \exists R.D \rfloor_\rho \wedge n = 1) \vee n = 0$
$maxC(C \xrightarrow{R} D)_\rho =$
$n : \lfloor C \sqsubseteq = nR.D \rfloor_\rho \vee \lfloor C \sqsubseteq \leq nR.D \rfloor_\rho \vee n = \infty$
$exactC(C \xrightarrow{R} D)_\rho =$
$n : \lfloor C \sqsubseteq = nR.D \rfloor_\rho \vee (\lfloor C \sqsubseteq \geq nR.D \rfloor_\rho \wedge \lfloor C \sqsubseteq \leq nR.D \rfloor_\rho) \vee n = 0$
$sameR(C1 \xrightarrow{RC} C2, D1 \xrightarrow{RD} D2)_{(\rho_C, \rho_D)} \Rightarrow$
If $\lfloor C1 \sqsubseteq \forall RC.C2 \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq \forall RD.D2 \rfloor_{\rho_D}$ and
If $\lfloor C1 \sqsubseteq \exists RC.C2 \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq \exists RD.D2 \rfloor_{\rho_D}$ and
$\forall n : \lfloor C1 \sqsubseteq = nRC.C2 \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq = nRD.D2 \rfloor_{\rho_D}$ and
$\forall n : \lfloor C1 \sqsubseteq \geq nRC.C2 \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq \geq nRD.D2 \rfloor_{\rho_D}$ and
$\forall n : \lfloor C1 \sqsubseteq \leq nRC.C2 \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq \leq nRD.D2 \rfloor_{\rho_D}$ and
$\forall a : \lfloor C1 \sqsubseteq RC : a \rfloor_{\rho_C}$ then $\lfloor D1 \sqsubseteq RD : a \rfloor_{\rho_D}$,
which defines a set of rules that enforce the same role restrictions between two pairs of concepts in different TBoxes

Table 11: Additional expressions in the ontology definition.

If an ontology, O_1 , extends or imports (\triangleright) another ontology, O_2 , then O_2 is contained in O_1 , i. e. if $O_1 \triangleright O_2$ then $DL_{O_2} \subseteq DL_{O_1}$, $TBox_{O_2} \subseteq TBox_{O_1}$, $RBox_{O_2} \subseteq RBox_{O_1}$ and $AnBox_{O_2} \subseteq AnBox_{O_1}$. The relation \triangleright is transitive.

4.1.2 Definition of Knowledge Base

A knowledge base is an ontology with an Assertion Box (**ABox**). It is defined as a 3-tuple $KB := (ADL, ABox, O)$, where:

- ADL is a set of nominals according to the syntax rule in [Table 10](#);
 - $ADLn \subseteq ADL$ is the subset of individuals, $a \rightarrow n$;
 - $ADLv \subseteq ADL$ is the subset of datatype values, $a \rightarrow v$;
- $ABox$ is the **ABox**, containing assertions of the form $X \rightarrow C(n) \mid R(n, a)$, where $C \in DLC_O$, $R \in DLR_O$, $n \in ADLn$, and $a \in ADL$;
- O is the ontology of the knowledge base.

The expression $\lfloor X \rfloor_\rho$ may be used to represent axioms inside an **ABox** ρ , where X must follow the syntax allowed in the **ABox**.

The **ABox** contains “assertions about named individuals in terms of” the **TBox** [3], such as $C(a)$ and $R(a, b)$. $C(a)$ means that a belongs to the interpretation of C (i. e. a is an individual of C). $R(a, b)$ means that b is a filler of the role R for a (i. e. a is related to b through R).

An **ABox** can be considered as a possible instantiation of the **TBox**. In analogy to relational databases, the **TBox** would be the relational database schema and the **ABox** would be the rows or data inside each table. However, unlike relational databases, which follow a closed-world assumption, the interpretation of an **ABox** follows an open-world assumption. Thus, it cannot be assumed that the knowledge in a Knowledge Base (**KB**) is complete.

4.1.3 Interpretation of the DL Language

A **DL** interpretation is a tuple $I := (\Delta^I, \cdot^I)$ where Δ^I is a non-empty set, defining the domain of the interpretation, and \cdot^I is an interpretation function that assigns to every atomic concept A a set $A^I \subseteq \Delta^I$, and to every atomic role R a binary relation

$R^I \subseteq \Delta^I \times \Delta^I$. The interpretation function is extended to concept and role descriptors by the inductive definitions shown in Table 12.

An interpretation I is a model of an ontology O if it satisfies the axioms in the TBox, the RBox, and the AnBox, of O , i.e.:

- If $C \sqsubseteq D$ then $C^I \subseteq D^I$;
- If $R \sqsubseteq S$ then $R^I \subseteq S^I$;
- If $C \equiv D$ then $C^I = D^I$;
- If $R \equiv S$ then $R^I = S^I$;
- If $R(C, a)$ then $(C^I, a^I) \in R^I$.

A model of an ontology is also a model of the ontology's TBox, RBox and AnBox.

A concept, C , is *satisfiable* (i. e. non-contradictory) with respect to a TBox, ρ , if there exists a model of ρ , such that C is non-empty (i. e. does not subsume \perp) [3].

An interpretation I is a model of a knowledge base KB with an ontology O , if it is a model of O and if it satisfies the axioms in the ABox of KB , i.e.:

- If $C(n)$ then $n^I \in C^I$;
- If $R(n, a)$ then $(n^I, a^I) \in R^I$.

A^I	\subseteq	Δ^I
R^I	\subseteq	$\Delta^I \times \Delta^I$
$(R^+)^I$	$=$	$\bigcup_{n>0} (R^n)^I, (R^n)^I = \{(a, b) \mid \exists b : (b, c) \in R^I \wedge (a, b) \in (R^{n-1})^I\}$
$(R^-)^I$	$=$	$\{(b, a) \in \Delta^I \times \Delta^I \mid (a, b) \in R^I\}$
$(R \sqcup S)^I$	$=$	$R^I \cup S^I$
\top^I	$=$	Δ^I
\perp^I	$=$	\emptyset
$(\{a, b, \dots\})^I$	\subseteq	Δ^I
$(C \sqcup D)^I$	$=$	$C^I \cup D^I$
$(\forall R.C)^I$	$=$	$\{a \in \Delta^I \mid \forall b : (a, b) \in R^I \Rightarrow b \in C^I\}$
$(\exists R.C)^I$	$=$	$\{a \in \Delta^I \mid \exists b : (a, b) \in R^I \wedge b \in C^I\}$
$(= nR.C)^I$	$=$	$\{a \in \Delta^I \mid \{b \mid (a, b) \in R^I \wedge b \in C^I\} = n\}$
$(\geq nR.C)^I$	$=$	$\{a \in \Delta^I \mid \{b \mid (a, b) \in R^I \wedge b \in C^I\} \geq n\}$
$(\leq nR.C)^I$	$=$	$\{a \in \Delta^I \mid \{b \mid (a, b) \in R^I \wedge b \in C^I\} \leq n\}$
a^I	\in	Δ^I
$(R : a)^I$	$=$	$\{d \in \Delta^I \mid (d, a^I) \in R^I\}$

Table 12: The DL interpretation function definitions.

An **ABox**, A , is *consistent* with respect to a **TBox**, T , and a **RBox**, R , if there is an interpretation that is a model of A , T and R [3].

4.1.4 Domain Ontologies

The static structure and semantics of a particular domain of knowledge are captured by domain ontologies in the form of concepts, their relations and constraints. These can be analysed and employed in the semi-automatic construction of workflow-definition ontologies.

Consider the document ontology and example **ABox** presented in Figure 9 as a knowledge base. The graph structure of the **TBox** defines the known concepts and roles of individuals in the **ABox**. Following the restrictions specified in this structure, the incremental filling of the **ABox** is possible through the execution of several atomic operations represented through task-definitions. In the specific case of the document ontology, an initial **ABox** with English sections may be supplied as input to the workflow, resulting in translated Portuguese sections. Since the ontology contains the semantics for the sub-division of sections, some of the task-definitions may consider their sub-division into smaller units (e. g. paragraphs, sentences).

The ontology in Figure 9 contains the following **TBox**:

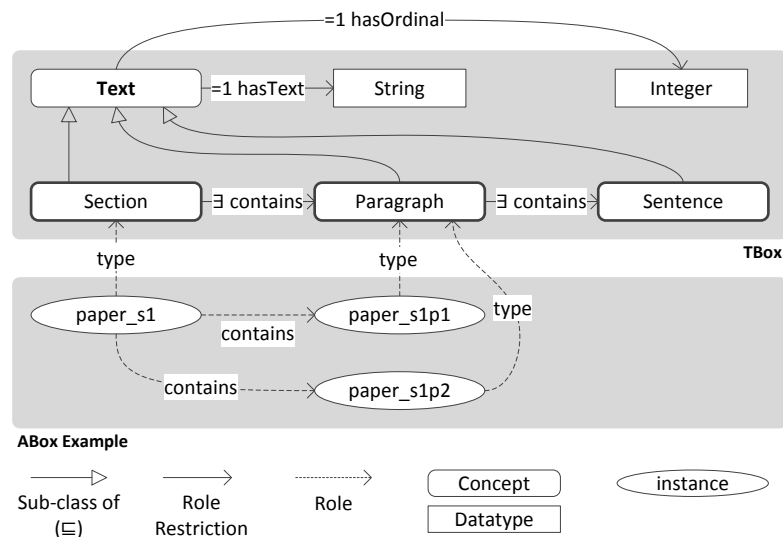


Figure 9: The document ontology (TBox only) with a possible example ABox (or instantiation). The TBox is an adaptation from the DoCO¹.

¹DoCO: <http://purl.org/spar/doco/>

- $Text \sqsubseteq = 1hasText.String$;
- $Text \sqsubseteq = 1hasOrdinal.Integer$;
- $Section \sqsubseteq Text$;
- $Section \sqsubseteq \exists contains.Paragraph$;
- $Paragraph \sqsubseteq Text$
- $Paragraph \sqsubseteq \exists contains.Sentence$;
- $Sentence \sqsubseteq Text$.

The **ABox**, on the other hand, contains the following axioms:

- $Section(paper_s1)$;
- $Paragraph(paper_s1p1)$;
- $Paragraph(paper_s1p2)$;
- $contains(paper_s1, paper_s1p2)$;
- $contains(paper_s1, paper_s1p1)$.

4.2 THE COMPFLOW ONTOLOGY

The CompFlow ontology, as depicted in [Figure 10](#), defines the basic concepts and roles required to represent workflows. It captures concepts and lessons learnt from widely known workflow specification languages and approaches such as the [XPDL](#) and [BPMN](#) [22]. Furthermore, it incorporates concepts that aid in the crowdsourcing, distribution and delivery of tasks.

Formally, the CompFlow ontology or its extension (the deployment ontology) is defined as $OCF = (DL_{OCF}, TBox_{OCF}, RBox_{OCF}, AnBox_{OCF})$, which includes:

- $DDT_{OCF} \subseteq DL_{OCF} = \{Unit, Response, UnitContext, ResponseContext\}$, i. e. the set of abstract concepts that represent the domain (input and output) of an activity (the domain definition types);
- $DDR_{OCF} \subseteq DL_{OCF} = \{hasUnit, hasUnitContext, hasResponse, hasResponseContext\}$, i. e. the set of top-level roles that restrict the domain (input and output) of an activity (the domain definition roles);
- $TDT_{OCF} \subseteq DL_{OCF} = \{CreateAndFillTask, FillTask, SelectionTask, MajorityVotingFilterTask\}$, which is a set of task types;

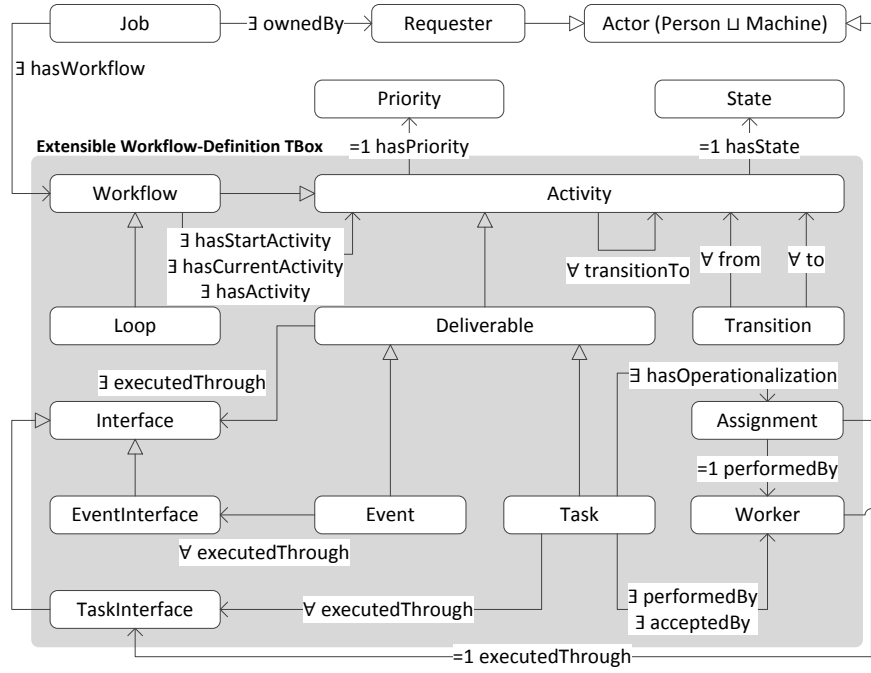


Figure 10: Overview of the CompFlow ontology.

- $EDT_{OCF} \subseteq DL_{OCF} = \{InstantiationEvent, RunningEvent\}$, which is a set of event types;
- $RDT_{OCF} \subseteq DL_{OCF} = \{BasicTransition, ConditionalTransition, SynchronizationTransition, MergeTransition, ParallelTransition, DisjunctTransition\}$, which is a set of transition types;
- $PD_{OCF} \subseteq DL_{OCF} = \{lowest, low, medium, high, highest\}$, i.e. the set of priority values;
- $SD_{OCF} \subseteq DL_{OCF} = \{notStarted, inProgress, paused, cancelled, finished, withError\}$, i.e. the set of state values.

4.2.1 Jobs

The *Job* concept represents a workflow execution environment created and owned by a *Requester*, which may contain more than one *Workflow*. Each job has an operational *ABox*, $OABox \subseteq ABox_{KBJ}$, inside a knowledge base, $KBJ = (ADL_{KBJ}, ABox_{KBJ}, O_{KBJ})$, shared by all its workflows, where O_{KBJ} is an ontology that imports all the workflow-definition ontologies of the job. The partial *TBox* that describes the contents inside the operational *ABox* is called an operational *TBox*.

4.2.2 Interfaces

The execution of a workflow requires interaction with external actors and services during the execution of *Event* and *Task* activities. While an *Event* is typically listened for, and arrives through an *EventInterface*, a *Task* must be delivered to and retrieved from workers through a *TaskInterface*. Thus, interfaces represent logical and/or physical components through which the interaction with workers (machine or person/human) is performed (e. g. a Web service interface, a graphical user Web interface).

The ability to represent different types of interfaces enables the specification of distinct interfaces, commonly used on user-centric environments [41]:

- Simple, where a single medium or modality is used. For instance, tasks can be delivered to workers through a visual interface, a sound interface, or simply through a Web interface (the common case for crowdsourcing applications);
- Multi-modal, i. e. capable of merging and coordinating multiple mediums and modalities in a single interface.

Accordingly, and of particular interest in the crowdsourcing scenario, different types of user interface implementations (such as a game interface or a mobile interface) can be used to distribute tasks through human workers.

Likewise, an event occurrence can arrive through different types of interfaces, such as publish-subscribe interfaces [14], or event queues.

4.2.3 Actors

Two main types of actors are considered by the CompFlow ontology: the *Requester* and the *Worker*. Also, actors are either *Person* (human) or *Machine* (see Figure 11 for an illustration).

Each *Actor* may belong to several *ActorGroup*. Actor groups allow requesters to associate and filter groups of actors for participation in particular tasks. An *ActorGroup* may include a wide set of attributes, including social network analysis clustering measurements (e. g. clusterability), improving the control of the requester over the selection of workers. Inclusively, each *ActorGroupMembership* may feature a wide set of actor specific attributes and measurements (e. g. centrality and prestige measures).

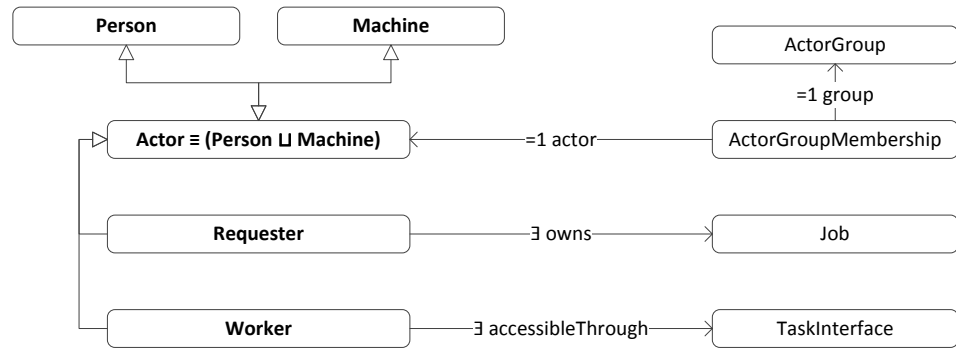


Figure 11: Actors in the CompFlow ontology.

4.2.4 Activities

Activities are the inter-connected components that form a workflow. There are three main types of activities: the *Workflow*, the *Task* and the *Event*. Among these, *Deliverables*, which include *Task* and *Event*, represent a group of activities that require worker or external interaction through some kind of *Interface*.

Each activity has a *State* and a *Priority*. The *State* of the activity is established during the execution of the workflow it belongs to. The possible state values are *notStarted*, *inProgress*, *paused*, *finished*, *cancelled* and *withError*. The *Priority* is assigned by the requester or the creator of the workflow-definition, and provides the instantiation and execution engine with a measure that can be employed to schedule activities in situations of concurrency. The possible priority values are *lowest*, *low*, *medium*, *high* and *highest*.

4.2.4.1 Tasks

A task is a set of assignments and operations on top of operational input data (**ABox**), which must be performed by workers.

All the operations involved in the execution of a task are performed on top of the data in the operational **ABox**. Thus, the input of the tasks comes from the operational **ABox** and the output of the tasks goes into the operational **ABox**. Analogously, the operations involved in a task, along with its input and output data, are described by its operational **TBox**.

As depicted in Figure 12, the representation of a task involves multiple concepts and roles in the CompFlow ontology.

The concepts inside the operational **TBox** of the task representation include:

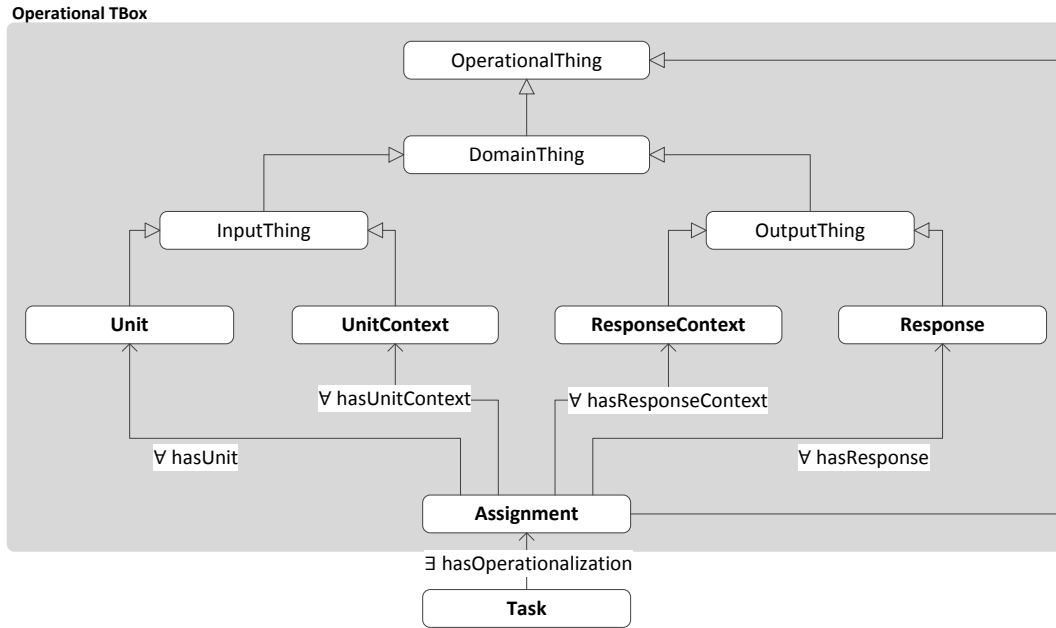


Figure 12: Abstract representation of a task according to the CompFlow ontology.

- The *Assignment* concept, which represents the actual operationalization of the task;
- Input concepts:
 - The *Unit* concepts, which represent the input unit of work given to the worker;
 - The *UnitContext* concepts, which represent relevant contextual input data that must be presented along with the unit (and possibly related to it);
- Output concepts:
 - The *Response* concepts, which represent the top-level response or output given by the worker;
 - The *ResponseContext* concepts, which represent additional output given by the worker, usually related to the response.

Each work unit (represented by the *Unit* concept) is assigned to a worker through an *Assignment*. The same unit may be assigned to different workers, resulting in different solutions to the same problem.

A task is classified, as depicted in [Figure 13](#), according to its inherent atomic operations.

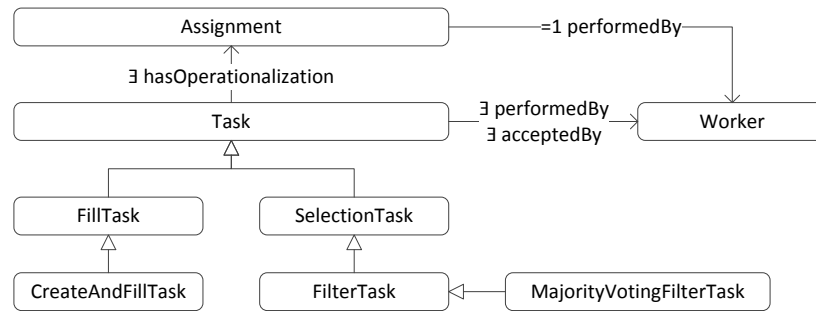


Figure 13: Types of tasks in the CompFlow ontology.

The type of the task captures the semantics, the structure, and many operational features of the task. The two main types of tasks are *SelectionTask* and *FillTask*. While a *FillTask* fills or updates datatype values of individuals in the operational *ABox*, a *SelectionTask* consists in selecting or pinpointing individuals inside the operational *ABox*.

The *FilterTask* is a special *SelectionTask* that applies a filtering or voting strategy and removes the least selected or unselected individuals from the operational *ABox*.

Different types of *FilterTask* may be represented through different sub-concepts. Currently, a majority voting strategy, which can be applied by either or both human and machine workers, is included. *FilterTask* are particularly important for tasks that contain more than one assignment per unit, i. e. where the same unit is assigned to more than one worker. In these situations, the execution of the task may result in the inconsistency (see Section 4.1.3) of the operational *ABox*. In order to avoid these situations, tasks with more than one assignment per unit are often followed by an associated *FilterTask*.

A *FillTask* only fills datatype role values for already existent individuals of a given concept. A *CreateAndFillTask*, on the other hand, is a special type of *FillTask* that creates a new individual, for a given concept, and fills its datatype role values.

Workers are associated with assignments and tasks through the *acceptedBy* and *performedBy* roles. The *acceptedBy* role associates a worker that has accepted to participate in a particular task. The *performedBy* role associates a worker that has performed a particular assignment.

4.2.4.2 Events

An *Event*, as depicted in Figure 14, is an external occurrence that either triggers the continuation of a running workflow (a *RunningEvent*) or triggers the execution of a new workflow (an *InstantiationEvent*).

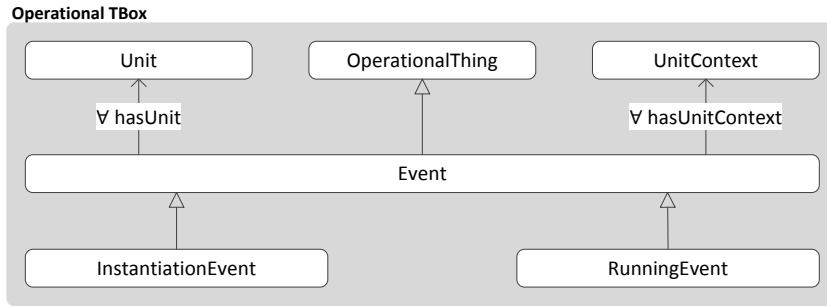


Figure 14: Events in the CompFlow ontology.

Events are listened and received through an *EventInterface*. They may bring and result in additional domain data inside the operational *ABox*, which can be used by the following activities. These data, as in the representation of a task, are represented through *Unit* and *UnitContext* concepts. Thus, the whole event representation (or event-definition) belongs to the operational *TBox*.

4.2.4.3 Workflows

Workflows are graphs of activities linked through transitions, which establish a process that delivers, for a given input dataset, a specific result dataset.

Each workflow contains one or more start activities. Workflows that are part of (or activities of) another workflow are called sub-workflows.

In some cases, the iterative execution of a certain group of activities is required. In these situations, the activities are grouped into a sub-workflow defined as a *Loop*. The representation of a *Loop* requires that a condition for stopping the iterative process is given through either a constant integer value that establishes the maximum amount of iterations, or a rule that is executed on top of the operational *ABox*.

The operational *TBox* of a workflow (its partial representation) is the union of the operational *TBoxes* of all its activities. Notice that the operational *ABox*, as depicted in [Figure 15](#), only contains individuals of domain concepts and operational *TBox* concepts.

4.2.5 Transitions

The flow of activities in a workflow is established through transitions. Transitions establish a connection between origin activities (through the *from* role) and destination activities (through the *to* role). As illustrated in [Figure 16](#), there are six main types of

transitions. According to the set of incoming activities, a transition is classified as a *BasicTransition*, a *MergeTransition* or a *SynchronizationTransition*. According to the set of outgoing activities, a transition is classified as a *ParallelTransition* or a *DisjunctTransition*. Furthermore, if a transition establishes one or more conditions onto the operational *ABox* that must be fulfilled in order to continue its execution, then it is a *ConditionalTransition*.

A transition is a *BasicTransition* if it has a single incoming activity and no enforced conditions. A *SynchronizationTransition*, on the other hand, waits for two or more incoming activities before proceeding. A *MergeTransition* is a transition where there is a set of incoming activities from which only one of those activities needs to be finished in order to proceed. The expressivity of *DL* allows situations where a transition contains both synchronizations and merges (see the *DL* definition in Section 5.4).

A *ParallelTransition* triggers all activities in a set of outgoing activities. On the other hand, a transition is a *DisjunctTransition* if there is a set of outgoing activities from which only one of those activities is triggered.

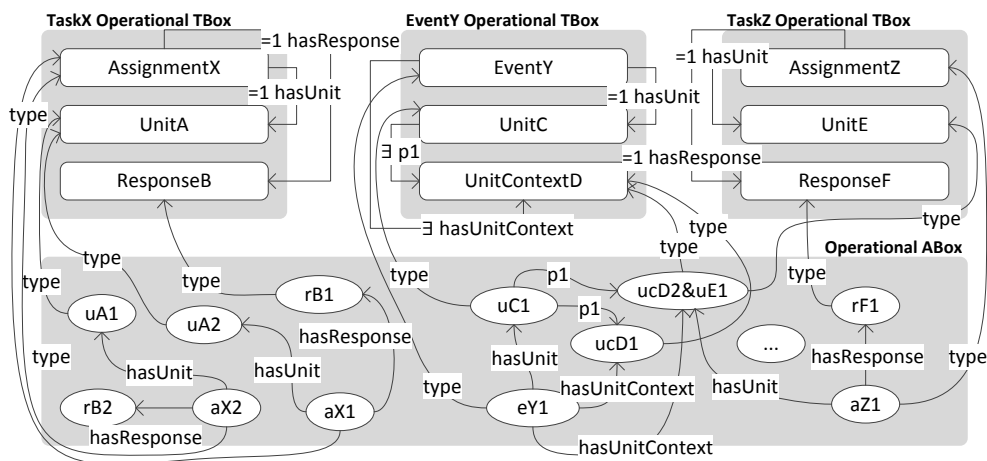


Figure 15: Example of the operational TBox and ABox of a workflow.

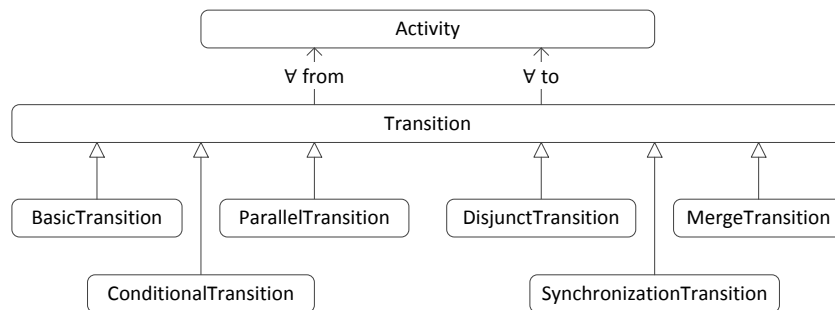


Figure 16: Transitions in the CompFlow ontology.

4.3 SUMMARY

Since ontologies, as a formal representation mechanism, are currently “the best answer to the demand for intelligent systems that operate closer to the human conceptual level” [52], they are appropriate for the definition of workflows and tasks that must be handled by both human and machine workers.

In order to formally define the CompFlow approach, the language and interpretation of an expressive family of DL was established. Through this language, an example domain ontology and the CompFlow ontology have been presented.

The CompFlow ontology is the result of the analysis of state of the art CS and HC approaches, plus workflow and business process representation and execution approaches. It defines the basic concepts and roles required to handle and represent workflow-definitions. Furthermore, it considers scenarios where multiple types of interfaces are used to interact with external human or machine actors. This is useful not only in the context of CS environments, where many Web platforms have emerged, but also in user-centric environments where several physical interfaces are used to interact with multiple actors.

A METHOD FOR THE CONSTRUCTION OF WORKFLOW-DEFINITIONS

Tasks (whether they involve physical actions or not) can be seen as a process that, in a particular context, results in the emergence of new data (i. e. responses) from the presentation of particular pieces of data (i. e. units) to a worker or actor. A workflow of tasks is, thus, the continuous ordered increment of new (different types of) data, in a specific context or domain.

With the assumption that domain ontologies represent the structure and semantics of the data (i. e. the static dimension) that must be presented and retrieved from workers during the execution of a task, a method for the definition and representation of ontology-based workflows of tasks (i. e. the construction of workflow-definition ontologies) is proposed.

The proposed method suggests that the task's input and output data, namely units and responses, correspond to (likely related) individuals of domain ontology concepts. Therefore, and more concretely, the execution of a task can be considered to be an operation on top of individuals and their relationships, according to the domain ontology. The execution of a workflow of tasks is then considered to be the incremental instantiation of the domain ontology according to its structure and semantics.

5.1 THE WORKFLOW-DEFINITION ONTOLOGY

A workflow-definition ontology contains one, and only one, top-level workflow-definition (i. e. a non-sub-workflow-definition), which may contain multiple activity-definitions:

- *Sub-workflow-definitions*, i. e. representations of sub-workflows;
- *Task-definitions*, i. e. representations of tasks;
- *Event-definitions*, i. e. representations of events.

Activity-definitions are connected through *transition-definitions*, which represent transitions between activities.

Task-definitions and event-definitions define the operational **TBox**, which describes and represents the contents inside the operational **ABox**. The operational **TBox**, $OTBox$, of a workflow-definition ontology, OWF , is defined as a partial **TBox** inside the workflow-definition ontology, i. e. $OTBox \subseteq TBox_{OWF}$.

The workflow-definition ontology, defined as $OWF = (DL_{OWF}, TBox_{OWF}, RBox_{OWF}, AnBox_{OWF})$, is a:

- Domain extension (\triangleright_{OD}) of a domain ontology, $OD = (DL_{OD}, TBox_{OD}, RBox_{OD}, AnBox_{OD})$, through Δ_{OD} (see Section 5.2.1 and Section 5.3.1), i. e. $OWF \triangleright_{OD} OD$;
- CompFlow extension (\triangleright_{OCF}) of a CompFlow (or deployment) ontology, $OCF = (DL_{OCF}, TBox_{OCF}, RBox_{OCF}, AnBox_{OCF})$, through Δ_{OCF} (see Section 5.2.1 and Section 5.3.1), i. e. $OWF \triangleright_{OCF} OCF$.

5.2 TASK-DEFINITIONS

Task-definitions are representations of (i) a set of atomic operations, (ii) their input data and (iii) their output data (i. e. of tasks). They are augmented partial replicas of the domain ontology's **TBox**. In fact, a task-definition can be resumed to the representation of a process of obtaining a consistent **ABox** with respect to the task-definition's operational **TBox**, which fully follows role restrictions involving input and output concepts through a closed-world assumption (i. e. all restrictions must be satisfied by the **ABox**).

5.2.1 Definition

Formally, a task-definition is a structure $TDef(OWF) := (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, W, I, p, l, d, au)$, where:

- $AT \in DLAC_{OWF}$ is the atomic task concept;
- $T \subseteq TDT_{OCF}$ is the set of atomic concepts that define the type of the task-definition;
- $AA \in DLAC_{OWF}$ is the atomic assignment concept;
- $IO \subseteq DL_{OWF}$ is a set of input and output concept descriptors;
 - $IOI \subseteq IO \cap DLAC_{OWF}$ is the set of atomic input concepts;

- $IOO \subseteq IO \cap DLAC_{OWF}$ is the set of atomic output concepts;
- $IOI \cup IOO = IO \cap DLAC_{OWF}$;
- $IOBox \subseteq OTBox$ is a partial operational $TBox$ that establishes the role restrictions between all input and output concepts in IO (a refinement of those present in the domain ontology). It does not contain any concept descriptor from DL_{OCF} , DL_{OD} or from any other subset besides IO ;
 - $DTBox \subseteq IOBox$ is the partial $IOBox$ that establishes all datatype role restrictions onto input and output concepts;
- $AR \subseteq DLR_{OWF}$ is a set of custom input and output roles that establish the role restrictions between AA and each atomic input and output concept;
- $ARBox \subseteq OTBox$ is the partial operational $TBox$ that establishes all input and output role restrictions and cardinalities onto AA (using the roles in AR and in DDR_{OCF});
- $\Delta_{OD} : (IOI \cup IOO) \times DLAC_{OD}$ is a type-of relation that associates each atomic input or output concept to a domain concept;
- $\Delta_{OCF} : (IOI \cup IOO) \times DDT_{OCF}$ is a type-of relation that associates each atomic input or output concept to its type in the OCF ontology;
- $W \in DLAC_{OWF}$ is the atomic worker concept;
- $I \in DLAC_{OWF}$ is the atomic task interface concept;
- $p \in PD_{OCF}$ is the individual indicating the priority of the task-definition;
- $l \in DLV_{OWF}$ is the datatype value containing the label of the task-definition;
- $d \in DLV_{OWF}$ is the datatype value containing the description of the task-definition;
- $au \in DLV_{OWF}$ is the datatype value corresponding to the amount of assignments per unit (i. e. assignments with the same input data) of the task-definition.

Accordingly, the functions in [Table 13](#) are considered for this definition.

In the context of the task-definition, the relations Δ_{OD} and Δ_{OCF} can be expressed in pure DL . For two concepts C and D with $(C, D) \in \Delta_{OD}$, then $C \sqsubseteq D$. Similarly, for two concepts, C and D with $(C, D) \in \Delta_{OCF}$, then $C \sqsubseteq D$.

The $IOBox$ represents all the data involved in a single assignment. It must be interpreted using a closed-world assumption (where all unknown knowledge is presumed to be false) during the execution of tasks. On the one hand, input concepts and their

FUNCTION	DESCRIPTION
$isTDef : DLAC_{OWF} \rightarrow \{0,1\}$	indicates if an atomic concept represents a task-definition
$fTDef_{OP} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given AT returns its set of operational atomic concepts, i. e. $fTDef_{OP}(AT) = \{AA\} \cup IOI \cup IOO$
$fTDef_{AA} : DLAC_{OWF} \rightarrow DLAC_{OWF}$	given AT returns its AA , i. e. $fTDef_{AA}(AT) = AA$
$fTDef_{IO} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given AT returns its IO , i. e. $fTDef_{IO}(AT) = IO$
$fTDef_{IOI} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given AT returns its IOI , i. e. $fTDef_{IOI}(AT) = IOI$
$fTDef_{IOO} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given AT returns its IOO , i. e. $fTDef_{IOO}(AT) = IOO$
$fTDef_{IOBox} : DLAC_{OWF} \rightarrow 2^{OTBox}$	given AT returns its $IOBox$, i. e. $fTDef_{IOBox}(AT) = IOBox$
$fTDef_{\Delta_{OD}}$	given AT returns its Δ_{OD} , i. e. $fTDef_{\Delta_{OD}}(AT) = \Delta_{OD}$
$fTDef_{\Delta_{OCF}}$	given AT returns its Δ_{OCF} , i. e. $fTDef_{\Delta_{OCF}}(AT) = \Delta_{OCF}$

Table 13: Additional functions for task-definitions.

role restrictions represent the input data that must be selected from the operational $ABox$. Since a closed-world assumption is followed, all DL restrictions must be satisfied by the selected data. On the other hand, output concepts represent new data that will be added to the operational $ABox$. Similarly, the new data must also satisfy all the restrictions enforced by the $IOBox$. For instance, if a cardinality role restriction of *exactly one* exists between an assignment concept $A1$, and an unit context concept $UC1$, then exactly one individual of type $UC1$ must be explicitly associated with each assignment of type $A1$, in such a way that it satisfies the role restriction.

The structure $TDef$ is a valid task-definition if it satisfies the following conditions:

- All of its concepts are satisfiable with respect to $TBox_{OWF}$;
- $T \neq \emptyset$;
- $\exists C : C \in IOO \wedge (C, Response) \in \Delta_{OCF}$;
- $\forall C \in (IOI \cup IOO) \Rightarrow |\{D | (C, D) \in \Delta_{OCF}\}| = 1$;

- $\forall C, D : C \in IOO \wedge D \in IOI \wedge [C \sqsubseteq D]_{IOBox} \Rightarrow \{X | (C, X) \in \Delta_{OD}\} = \{Y | (D, Y) \in \Delta_{OD}\};$
- $\forall R \in AR \Rightarrow [R \sqsubseteq S]_{RBox_{OWF}} \wedge S \in DDR_{OCF};$
- $\forall C \in (IOI \cup IOO) \Rightarrow \exists R : (AA \xrightarrow{R} C)_{ARBox} \wedge (R \in AR \vee R \in DDR_{OCF});$
- $\forall C, R : C \in (IOI \cup IOO) \wedge (AA \xrightarrow{R} C)_{ARBox} \Rightarrow (R = S \vee [R \sqsubseteq S]_{RBox_{OWF}})$ and:
 - $S = hasUnit \wedge exactC(AA \xrightarrow{R} C) = 1$ if $(C, Unit) \in \Delta_{OCF};$
 - $S = hasUnitContext$ if $(C, UnitContext) \in \Delta_{OCF};$
 - $S = hasResponse$ if $(C, Response) \in \Delta_{OCF};$
 - $S = hasResponseContext$ if $(C, ResponseContext) \in \Delta_{OCF};$
- $\forall C, D, R : C \in (IOI \cup IOO) \wedge D \in (IOI \cup IOO) \wedge (C \xrightarrow{R} D)_{IOBox} \Rightarrow C \neq D;$
- $[W \sqsubseteq Worker]_{TBox_{OWF}};$
- $[I \sqsubseteq TaskInterface]_{TBox_{OWF}}.$

The structure $TDef$ also results in the following additional expressions in the $TBox_{OWF}$ and $AnBox_{OWF}$:

- $\forall C \in T \Rightarrow [AT \sqsubseteq C]_{TBox_{OWF}};$
- $[AA \sqsubseteq Assignment]_{TBox_{OWF}};$
- $[AT \sqsubseteq \forall hasOperationalization.AA]_{TBox_{OWF}};$
- $[AT \sqsubseteq \forall acceptedBy.W]_{TBox_{OWF}};$
- $[AT \sqsubseteq \forall executedThrough.I]_{TBox_{OWF}};$
- $[AT \sqsubseteq hasPriority : p]_{TBox_{OWF}};$
- $[AT \sqsubseteq hasAssignmentsPerUnit : au]_{TBox_{OWF}};$
- $[label(AT, l)]_{AnBox_{OWF}} \wedge [description(AT, d)]_{AnBox_{OWF}}.$

5.2.2 Types

A task-definition is classified according to its operational structure and semantics.

Consequently, three main types of task-definitions are proposed: the *FillTask*, the *SelectionTask*, and the *CreateAndFillTask*. *FillTask* and *SelectionTask* task-definitions are structurally distinguished from *CreateAndFillTask* task-definitions through the existence (or non-existence) of a subsumption between an output concept and an input concept.

More precisely, if there is an output concept C that subsumes an input concept D ($C \sqsubseteq D$), then all output individuals of C must also be input individuals of D (i. e. the inherent task-definition operation does not result in new individuals). In such situations, the task-definition is either a *FillTask* or a *SelectionTask*. Otherwise, if there is an output concept that does not subsume an input concept, then the inherent task-definition operation will result in new individuals. In this case, the task-definition has the type *CreateAndFillTask*.

The structural distinction between *FillTask* and *SelectionTask* task-definitions depends on the cardinality role restrictions of the input and output concepts in the *ARBox*. If there is an output concept C that subsumes an input concept D ($C \sqsubseteq D$) and at most one individual of D can belong to the input of each assignment (i. e. $\forall R : \max C(AA \xrightarrow{R} D)_{ARBox} \leq 1$), then at most one (the same) individual of D can be part of the output of each assignment. In this case, no selection can be performed and the task-definition is of type *FillTask*. Otherwise, it is a *SelectionTask* task-definition, since a selection of the multiple individuals of D must be performed.

Accordingly, a task-definition is:

- A *CreateAndFillTask* if it leads to new individuals of an output concept, i. e. $CreateAndFillTask \in T$ if:

$$\exists C \nexists D : C \in IOO \wedge D \in IOI \wedge [C \sqsubseteq D]_{IOBox}$$

- A *FillTask* is it fills or updates the datatype role values of an already existent individual, i. e. $FillTask \in T$ if:

$$\exists C \exists D \forall R : C \in IOO \wedge D \in IOI \wedge [C \sqsubseteq D]_{IOBox} \wedge (AA \xrightarrow{R} D)_{ARBox} \wedge \max C(AA \xrightarrow{R} D)_{ARBox} \leq 1$$

- A *SelectionTask* if it picks from or filters a set of already existent individuals, i. e. $SelectionTask \in T$ if:

$$\exists C \exists D \forall R : C \in IOO \wedge D \in IOI \wedge [C \sqsubseteq D]_{IOBox} \wedge (AA \xrightarrow{R} D)_{ARBox} \wedge \max C(AA \xrightarrow{R} D)_{ARBox} > 1$$

5.2.3 Cardinalities

Role restrictions in the *ARBox* establish the cardinalities of the input and output data for each assignment. The cardinality of the role restrictions in the *IOBox* must not

contradict those in the *ARBox*. Otherwise, the instantiation of the task-definition and associated creation of assignments will result in an inconsistent operational *ABox*.

In the particular case of subsumptions between concepts in the *IOBox*, $D \sqsubseteq C$, the role cardinality values of D in the *ARBox* must be less or equal than the role cardinality values of C , i.e.:

$$\begin{aligned} \forall C, D, R, S : C \in (IOI \cup IOO) \wedge D \in (IOI \cup IOO) \wedge [D \sqsubseteq^+ C]_{IOBox} \wedge \\ (AA \xrightarrow{R} C)_{ARBox} \wedge (AA \xrightarrow{S} D)_{ARBox} \Rightarrow \\ \max C(AA \xrightarrow{S} D)_{ARBox} \leq \max C(AA \xrightarrow{R} C)_{ARBox} \wedge \\ \text{exact} C(AA \xrightarrow{S} D)_{ARBox} \leq \text{exact} C(AA \xrightarrow{R} C)_{ARBox} \wedge \\ \min C(AA \xrightarrow{S} D)_{ARBox} \leq \min C(AA \xrightarrow{R} C)_{ARBox} \end{aligned}$$

This condition is enforced to allow the execution of a closed-world reasoning process during the instantiation of the task-definition and association of the input data. I.e., if $D \sqsubseteq C$, then all D are C . Thus, following a closed-world assumption, the dataset can never contain more individuals of D than those of C .

On the other hand, if there is a relationship between two concepts in the *IOBox*, $(D \xrightarrow{R} C)_{IOBox}$, the role cardinality values of the relationship in the *IOBox* must be less or equal than the role cardinality values of C in the *ARBox*, i.e.:

$$\begin{aligned} \forall C, D, R, S : C \in (IOI \cup IOO) \wedge D \in (IOI \cup IOO) \wedge (D \xrightarrow{R} C)_{IOBox} \wedge \\ (AA \xrightarrow{S} C)_{ARBox} \Rightarrow \\ \max C(D \xrightarrow{R} C)_{IOBox} \leq \max C(AA \xrightarrow{S} C)_{ARBox} \wedge \\ \text{exact} C(D \xrightarrow{R} C)_{IOBox} \leq \text{exact} C(AA \xrightarrow{S} C)_{ARBox} \wedge \\ \min C(D \xrightarrow{R} C)_{IOBox} \leq \min C(AA \xrightarrow{S} C)_{ARBox} \end{aligned}$$

Not meeting this condition results in an invalid closed-world interpretation of the task-definition. For instance, the role restriction $(AA \xrightarrow{S} C)_{ARBox}$ may demand for at most two individuals of C , and the *IOBox* may contain a role restriction $(D \xrightarrow{R} C)_{ARBox}$ demanding that for each individual of D there are exactly three of C . In this situation, for each assignment and its data, one of these axioms can never be satisfied.

If the role restriction is applied between an output concept and an input concept, an additional optional condition may be enforced regarding role cardinality restrictions

inside the $ARBox$, i. e. the role cardinality of one of the concepts in the $ARBox$ must be less or equal than one:

$$\begin{aligned} \forall C, D, R, S, T : & ((C \in IOI \wedge D \in IOO) \vee (C \in IOO \wedge D \in IOI)) \wedge \\ & (D \xrightarrow{R} C)_{IOBox} \wedge (AA \xrightarrow{T} D)_{ARBox} \wedge (AA \xrightarrow{S} C)_{ARBox} \Rightarrow \\ & \max C(AA \xrightarrow{S} C)_{ARBox} \leq 1 \vee \max C(AA \xrightarrow{T} D)_{ARBox} \leq 1 \end{aligned}$$

If this condition is enforced, the relationship R between C and D can be automatically established during the execution of the task. Otherwise, a combinatorial problem may arise due to the need to relate multiple instances of C with multiple instances of D , through R .

5.2.4 Example No. 1: Creating Individuals

The DL representation of a task-definition that creates new individuals is presented in [Figure 17](#). The depicted task-definition is defined as $TDef1 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 1)$, where:

- $AT = A1T$;
- $T = \{CreateAndFillTask\}$;
- $AA = A1$;
- $IOI = \{A', B'\}$;
- $IOO = \{C'\}$;
- $[A' \sqsubseteq \exists p1.B']_{IOBox}$, $[A' \sqsubseteq = 1p2.C']_{IOBox}$, $[A' \sqsubseteq p5 : "abc"]_{DTBox}$
and $[C' \sqsubseteq p6 : 3]_{DTBox}$;
- $AR = \emptyset$;
- $[A1 \sqsubseteq = 1hasUnit.A']_{ARBox}$, $[A1 \sqsubseteq \exists hasUnitContext.B']_{ARBox}$
and $[A1 \sqsubseteq = 1hasResponse.C']_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (B', B), (C', C)\}$;
- $\Delta_{OCF} = \{(A', Unit), (B', UnitContext), (C', Response)\}$;
- $[A1TW \equiv Worker]_{TBox_{OWF}}$ (all workers);
- $[A1TI \equiv TaskInterface]_{TBox_{OWF}}$ (all task interfaces).

without any related individuals of B' are still eligible as units, and (ii) that the worker may choose not to submit any response for the assignment of $TDef2$.

The task-definition in this example is defined as $TDef2 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 1)$, where:

- $AT = A1T$;
- $T = \{CreateAndFillTask\}$;
- $AA = A1$;
- $IOI = \{A', B'\}$;
- $IOO = \{C'\}$;
- $\lfloor A' \sqsubseteq \geq 0 p1 . B' \rfloor_{IOBox}$ and $\lfloor A' \sqsubseteq \geq 0 p2 . C' \rfloor_{IOBox}$;
- $AR = \emptyset$;
- $\lfloor A1 \sqsubseteq = 1 hasUnit . A' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq \geq 0 hasUnitContext . B' \rfloor_{ARBox}$ and $\lfloor A1 \sqsubseteq \geq 0 hasResponse . C' \rfloor_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (B', B), (C', C)\}$;
- $\Delta_{OCF} = \{(A', Unit), (B', UnitContext), (C', Response)\}$;
- $\lfloor A1TW \equiv Worker \rfloor_{TBox_{OWF}}$ (all workers);
- $\lfloor A1TI \equiv TaskInterface \rfloor_{TBox_{OWF}}$ (all task interfaces).

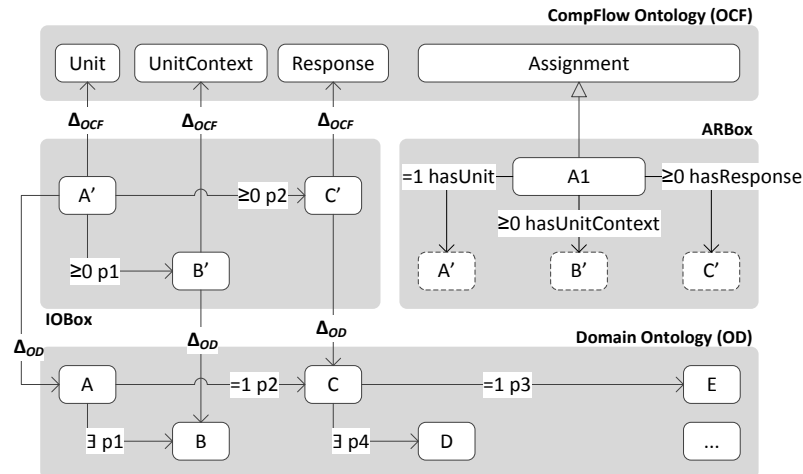


Figure 18: Operational DL representation of a task-definition with optional input and output data.

5.2.6 Example No. 3: Concept Hierarchies

Hierarchies of input and output concepts are formed when a subsumption exists between two or more concepts. Although the semantics remain that of *DL*, the operational interpretation is special regarding output concepts. The task-definition *TDef3*, depicted in Figure 19, exemplifies such a scenario.

In this scenario, the worker must provide an individual of *C'* (the response) for every individual of *A'* (the unit). However, since output sub-concepts of *C'* exist in the representation, the worker must also pick the type of the response individual, which may be either *CI'* or *CII'*. A relationship, *p1*, to *AI'* is established only if the response is of type *CI'*.

The task-definition in this example is defined as $TDef3 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 1)$, where:

- $AT = A1T$;
- $T = \{CreateAndFillTask\}$;
- $AA = A1$;
- $IOI = \{A', AI', AII'\}$;
- $IOO = \{C', CI', CII'\}$;

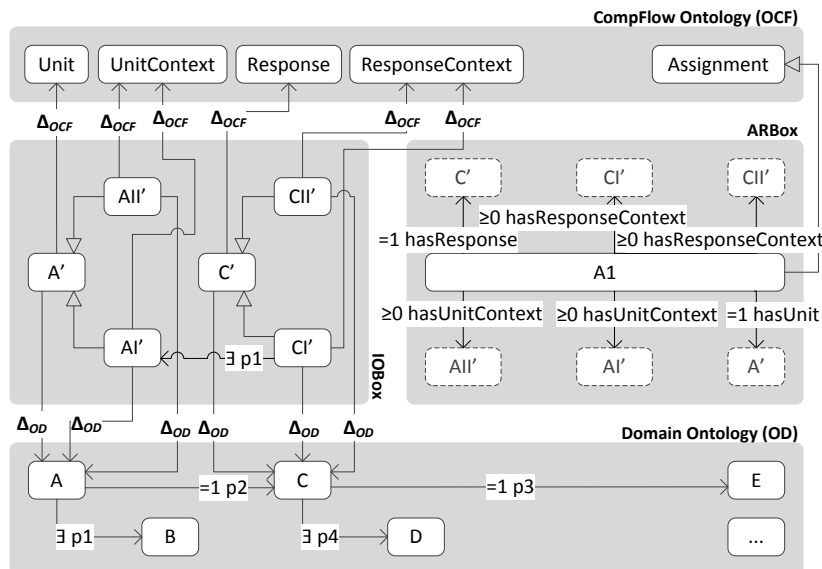


Figure 19: Operational DL representation of a task-definition with hierarchies of input and output concepts.

- $\lfloor AI' \sqsubseteq A' \rfloor_{IOBox}$, $\lfloor AII' \sqsubseteq A' \rfloor_{IOBox}$, $\lfloor CI' \sqsubseteq C' \rfloor_{IOBox}$, $\lfloor CII' \sqsubseteq C' \rfloor_{IOBox}$ and $\lfloor CI' \sqsubseteq \exists p1.AI' \rfloor_{IOBox}$;
- $AR = \emptyset$;
- $\lfloor A1 \sqsubseteq= 1hasUnit.A' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq= 1hasUnitContext.AI' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq= 1hasUnitContext.AII' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq= 1hasResponse.C' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq= 1hasResponseContext.CI' \rfloor_{ARBox}$ and $\lfloor A1 \sqsubseteq= 1hasResponseContext.CII' \rfloor_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (AI', A), (AII', A), (C', C), (CI', C), (CII', C)\}$;
- $\Delta_{OCF} = \{(A', Unit), (AI', UnitContext), (AII', UnitContext), (C', Response), (CI', ResponseContext), (CII', ResponseContext)\}$;
- $\lfloor A1TW \equiv Worker \rfloor_{TBox_{OWF}}$ (all workers);
- $\lfloor A1TI \equiv TaskInterface \rfloor_{TBox_{OWF}}$ (all task interfaces).

5.2.7 Example No. 4: Selecting or Filtering Individuals

The previous examples have focused on operations that lead to the emergence of new individuals. However, other operations (e.g. update or delete) on already existent individuals are also available. In these cases, a subsumption relationship from an output concept to an input concept must be established. Such a situation is depicted in [Figure 20](#), and defined through the task-definition $TDef4 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 1)$, where:

- $AT = A1T$;
- $T = \{MajorityVotingFilterTask\}$;
- $AA = A1$;
- $IOI = \{A', B'\}$;
- $IOO = \{A''\}$;
- $\lfloor A'' \sqsubseteq A' \rfloor_{IOBox}$ and $\lfloor A' \sqsubseteq \exists p1.B' \rfloor_{IOBox}$;
- $AR = \emptyset$;
- $\lfloor A1 \sqsubseteq \exists hasUnitContext.A' \rfloor_{ARBox}$, $\lfloor A1 \sqsubseteq \exists hasUnitContext.B' \rfloor_{ARBox}$ and $\lfloor A1 \sqsubseteq= 1hasResponse.A'' \rfloor_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (A'', A), (B', B)\}$;
- $\Delta_{OCF} = \{(A', UnitContext), (B', UnitContext), (A'', Response)\}$;

- $[A1TW \equiv Person]_{TBox_{OWF}}$ (only human workers);
- $[A1TI \equiv TaskInterface]_{TBox_{OWF}}$ (all task interfaces).

TDef4 depicts an operation where all output individuals must be a subset of (already existent) input individuals. Notice that there are no unit concepts in this task-definition. Since task-definitions of type *FilterTask* have an enforced cardinality restriction of exactly one onto unit concepts (otherwise they are not valid), multiple individuals of A' would not be able to be associated with a single assignment if A' was defined as a unit concept.

When a unit concept exists, the total amount of assignments is defined by the amount of units, times the amount of assignments per unit (*au*). In the case of *TDef4*, however, the amount of units is assumed to be equal to one, as long as unit context data are available. The total amount of assignments is, thus, equal to *au*.

TDef4 is a *SelectionTask*, which means that workers have to select or vote on the best or most appropriate individual of A' . The selected or voted individuals will also become individuals of A'' (i. e. A'' is the class of selected individuals of A'). However, since *TDef4* is also of type *FilterTask*, and in particular of type *MajorityVotingFilterTask*, a majority voting strategy will be employed in obtaining the most selected individual of A' . The remaining individuals of A' are then removed (filtered) from the underlying operational *ABox*.

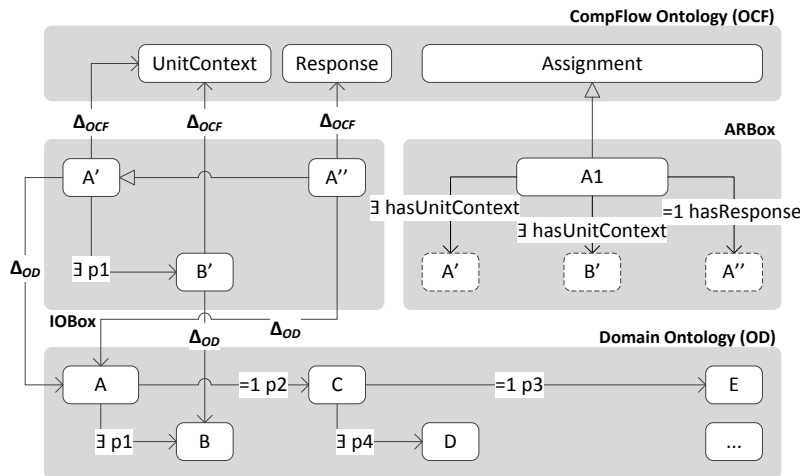


Figure 20: Operational DL representation of a task-definition with a filter operation.

5.2.8 Example No. 5: Updating Individuals

As with selection operations, the definition of an update operation requires a subsumption relationship from an output concept to an input concept. Figure 21 depicts a task-definition with an update operation, $TDef5 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 1)$, where:

- $AT = A1T$;
- $T = \{FillTask\}$;
- $AA = A1$;
- $IOI = \{A', B'\}$;
- $IOO = \{A''\}$;
- $[A'' \sqsubseteq A']_{IOBox}$ and $[A' \sqsubseteq \exists p1.B']_{IOBox}$;
- $AR = \emptyset$;
- $[A1 \sqsubseteq = 1hasUnit.A']_{ARBox}$, $[A1 \sqsubseteq \exists hasUnitContext.B']_{ARBox}$ and $[A1 \sqsubseteq = 1hasResponse.A'']_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (A'', A), (B', B)\}$;
- $\Delta_{OCF} = \{(A', Unit), (B', UnitContext), (A'', Response)\}$;
- $[A1TW \equiv Worker]_{TBox_{OWF}}$ (all workers);
- $[A1TI \equiv TaskInterface]_{TBox_{OWF}}$ (all task interfaces).

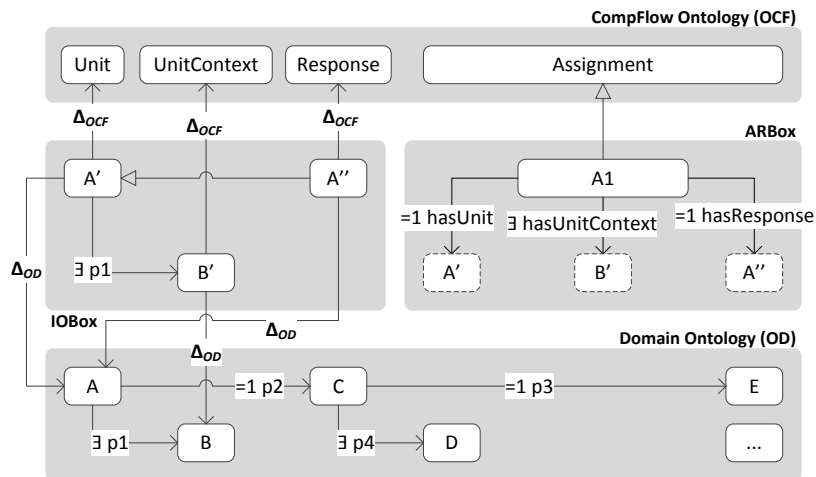


Figure 21: Operational DL representation of a task-definition with an update operation.

In this case, workers have to fill the datatype role values of each individual of A' . After the values (and the assignment) are successfully submitted, the individuals become part of A'' .

5.3 EVENT-DEFINITIONS

An event-definition is a representation of events and constitutes a waiting point inside the workflow-definition. Since event-definitions do not represent operations, the arrival or occurrence of an event (an instance of the event-definition) is the only requirement in order to proceed to the following activities.

5.3.1 Definition

Formally, an event-definition is a structure $EDef(OWF) := (AE, T, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, l, p, l, d)$, where:

- $AE \in DLAC_{OWF}$ is the atomic event concept;
- $T \in EDT_{OCF}$ is the atomic concept that defines the type of event-definition;
- $IO \subseteq DL_{OWF}$ is a set of data concept descriptors;
 - $IOI = IO \cap DLAC_{OWF}$ is the set of atomic data concepts;
- $IOBox \subseteq OTBox$ is a partial operational **TBox** that establishes the role restrictions between all data concepts in IO (a refinement of those present in the domain ontology). It does not contain any concept descriptor from DL_{OCF} , DL_{OD} or from any other subset besides IO ;
 - $DTBox \subseteq IOBox$ is the partial $IOBox$ that establishes all datatype role restrictions onto data concepts;
- $AR \subseteq DLR$ is a set of custom roles that establish the role restrictions between AE and each atomic data concept;
- $ARBox \subseteq OTBox$ is the partial operational **TBox** that establishes all atomic data concept role restrictions onto AE (using the roles in AR and in DDR_{OCF});
- $\Delta_{OD} : IOI \times DLAC_{OD}$ is a type-of relation that associates each atomic data concept to a domain concept;

- $\Delta_{OCF} : IOI \times \{Unit, UnitContext\}$ is a type-of relation that associates each atomic data concept to its type in the *OCF* ontology;
- $I \in DLAC_{OWF}$ is the atomic event interface concept;
- $p \in PD_{OCF}$ is the individual indicating the priority of the event-definition;
- $l \in DLv_{OWF}$ is the datatype value containing the label of the event-definition;
- $d \in DLv_{OWF}$ is the datatype value containing the description of the event-definition.

Accordingly, the functions in [Table 14](#) are considered for this definition.

In the context of the event-definition, the relations Δ_{OD} and Δ_{OCF} can be expressed in pure DL. For two concepts C and D with $(C, D) \in \Delta_{OD}$, then $C \sqsubseteq D$. Similarly, for two concepts, C and D with $(C, D) \in \Delta_{OCF}$, then $C \sqsubseteq D$.

The structure *EDef* is a valid event-definition if it satisfies the following conditions:

- All of its concepts are satisfiable with respect to $TBox_{OWF}$;
- $\forall C \in IOI \Rightarrow |\{D | (C, D) \in \Delta_{OCF}\}| = 1$;
- $\forall C \in IOI \Rightarrow \exists X : (C, X) \in \Delta_{OD}$;

FUNCTION	DESCRIPTION
$isEDef : DLAC_{OWF} \rightarrow \{0, 1\}$	indicates if an atomic concept represents an event-definition
$fEDef_{OP} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given <i>AE</i> returns its set of operational atomic concepts, i. e. $fEDef_{OP}(AE) = \{AE\} \cup IOI$
$fEDef_{IO} : DLAC_{OWF} \rightarrow 2^{DLC_{OWF}}$	given <i>AE</i> returns its <i>IO</i> , i. e. $fEDef_{IO}(AE) = IO$
$fEDef_{IOI} : DLAC_{OWF} \rightarrow 2^{DLAC_{OWF}}$	given <i>AE</i> returns its <i>IOI</i> , i. e. $fEDef_{IOI}(AE) = IOI$
$fEDef_{IOBox} : DLAC_{OWF} \rightarrow 2^{OTBox}$	given <i>AE</i> returns its <i>IOBox</i> , i. e. $fEDef_{IOBox}(AE) = IOBox$
$fEDef_{\Delta_{OD}}$	given <i>AE</i> returns its Δ_{OD} , i. e. $fEDef_{\Delta_{OD}}(AE) = \Delta_{OD}$
$fEDef_{\Delta_{OCF}}$	given <i>AE</i> returns its Δ_{OCF} , i. e. $fEDef_{\Delta_{OCF}}(AE) = \Delta_{OCF}$

Table 14: Additional functions for event-definitions.

- $\forall R \in AR \Rightarrow [R \sqsubseteq S]_{RBox_{OWF}} \wedge S \in \{hasUnit, hasUnitContext\};$
- $\forall C \in IOI \Rightarrow \exists R : (AE \xrightarrow{R} C)_{ARBox} \wedge R \in AR \cup \{hasUnit, hasUnitContext\};$
- $\forall R, C : (AE \xrightarrow{R} C)_{ARBox} \wedge C \in IOI \Rightarrow (R = S \vee [R \sqsubseteq S]_{RBox_{OWF}} \text{ and:}$
 - $S = hasUnit \wedge [AE \sqsubseteq= 1R.C]_{ARBox}$ if $(C, Unit) \in \Delta_{OCF};$
 - $S = hasUnitContext$ if $(C, UnitContext) \in \Delta_{OCF};$
- $\forall C, D, R : C \in IOI \wedge D \in IOI \wedge (C \xrightarrow{R} D)_{IOBox} \Rightarrow C \neq D;$
- $[I \sqsubseteq EventInterface]_{TBox_{OWF}}.$

The structure $EDef$ also results in the following additional expressions in the $TBox_{OWF}$ and $AnBox_{OWF}$:

- $[AE \sqsubseteq T]_{TBox_{OWF}};$
- $[AE \sqsubseteq \forall executedThrough.I]_{TBox_{OWF}};$
- $[AE \sqsubseteq hasPriority : p]_{TBox_{OWF}};$
- $[label(AE, l)]_{AnBox_{OWF}} \wedge [description(AE, d)]_{AnBox_{OWF}}.$

5.3.2 Example

An event-definition representing events with data that trigger the instantiation and execution of a workflow-definition (i. e. an *InstantiationEvent*), can be defined as $EDef1 = (AE, T, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, E1I, medium, l, d)$, where:

- $AE = E1;$
- $T = InstantiationEvent;$
- $IOI = \{A', B'\};$
- $[A' \sqsubseteq \exists p1.B']_{IOBox};$
- $[E1 \sqsubseteq= 1hasUnit.A']_{ARBox}$ and $[E1 \sqsubseteq \exists hasUnitContext.B']_{ARBox};$
- $\Delta_{OD} = \{(A', A), (B', B)\};$
- $\Delta_{OCF} = \{(A', Unit), (B', UnitContext)\};$
- $[E1I \equiv EventInterface]_{TBox_{OWF}}$ (all event interfaces).

This structure translates into the operational DL $TBox$ presented in [Figure 22](#).

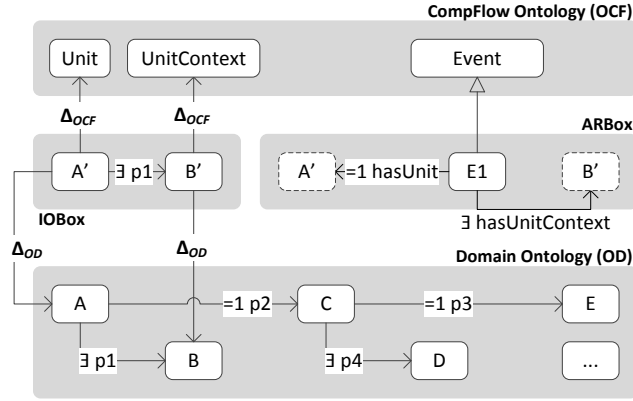


Figure 22: Operational DL representation of an event-definition.

5.4 TRANSITION-DEFINITIONS

A transition-definition (see Section 4.2.5) features the capabilities established by most flow control components in languages such as the XPD. Using DL concept descriptors, transition-definitions are specified either through a *transitionTo* role restriction onto *Activity* concepts or through *Transition* concepts.

5.4.1 Definition

Formally, a transition-definition is a structure $RDef(OWF) := (AR, T, CI, CO, Cond)$, where:

- $AR \in DLAC_{OWF}$ is the atomic transition concept;
- $T \subseteq RDT_{OCF}$ is the set of atomic concepts that define the type of the transition-definition;
- $CI \subseteq DLC_{OWF}$ is the set of concepts representing incoming activity-definitions, according to the syntax $C, D \longrightarrow A \mid C \sqcup D$;
- $CO \subseteq DLC_{OWF}$ is the set of concepts representing the outgoing activity-definitions, according to the syntax $C, D \longrightarrow A \mid C \sqcup D$;
- $Cond \subseteq DLv_{OWF}$ is a set of conditions onto the operational ABBox.

Accordingly, the functions in Table 15 are considered for this definition.

A $RDef$ is a valid transition-definition if it satisfies the following conditions:

- All of its concepts are satisfiable with respect to $TBox_{OWF}$;

FUNCTION	DESCRIPTION
$isRDef : DLAC_{OWF} \rightarrow \{0,1\}$	indicates if an atomic concept represents a transition-definition
$fRDef_T : DLAC_{OWF} \rightarrow 2^{RD_{TOCF}}$	given AR returns its T , i. e. $fRDef_T(AR) = T$
$fRDef_{CI} : DLAC_{OWF} \rightarrow 2^{DL_{COWF}}$	given AR returns its CI , i. e. $fRDef_{CI}(AR) = CI$
$fRDef_{CO} : DLAC_{OWF} \rightarrow 2^{DL_{COWF}}$	given AR returns its CO , i. e. $fRDef_{CO}(AR) = CO$
$fRDef_{Cond} : DLAC_{OWF} \rightarrow 2^{DL_{v_{OWF}}}$	given AR returns its $Cond$, i. e. $fRDef_{Cond}(AR) = Cond$

Table 15: Additional functions for transition-definitions.

- $T \neq \emptyset$;
- $|CI| \geq 1 \vee |CO| \geq 1$;
- $\forall C \in CI \Rightarrow [C \sqsubseteq Activity]_{TBox_{OWF}} \wedge (C \in DLAC_{OWF} \vee (\exists D : D \in DLAC_{OWF} \wedge \wedge D \in CI \wedge [D \sqsubseteq C]_{TBox_{OWF}}))$;
- $\forall C \in CO \Rightarrow [C \sqsubseteq Activity]_{TBox_{OWF}} \wedge (C \in DLAC_{OWF} \vee (\exists D : D \in DLAC_{OWF} \wedge \wedge D \in CO \wedge [D \sqsubseteq C]_{TBox_{OWF}}))$.

The structure $RDef$ also results in the following additional expressions in the $TBox_{OWF}$:

- $\forall C \in T \Rightarrow [AR \sqsubseteq C]_{TBox_{OWF}}$;
- $\forall C \in CI \Rightarrow [AR \sqsubseteq \exists from.C]_{TBox_{OWF}}$;
- $\forall C \in CO \Rightarrow [AR \sqsubseteq \exists to.C]_{TBox_{OWF}}$;
- $\forall c \in Cond \Rightarrow [AR \sqsubseteq hasCondition : c]_{TBox_{OWF}}$.

5.4.2 Types

A transition-definition is classified according to its operational structure and semantics.

A transition-definition is a *BasicTransition* if it has no conditions and exactly one input path, i. e. $BasicTransition \in T$ if:

$$|CI| = 1 \wedge Cond = \emptyset \wedge |CI \cap DLAC_{OWF}| = 1$$

A transition-definition is a *SynchronizationTransition* if it waits for (synchronizes) the flow of multiple input paths, i. e. $SynchronizationTransition \in T$ if:

$$|CI| > 1$$

A transition-definition is a *MergeTransition* if it waits for a single incoming path from a set of possible multiple input paths, i. e. $MergeTransition \in T$ if:

$$\exists C \in CI \wedge C \notin DLAC_{OWF}$$

Notice that, according to the syntax definition of the concept descriptors in CI , *MergeTransition* transition-definitions establish incoming paths through a union. This means that the flow from only one of the incoming paths in the union is required to arrive in order to continue the execution of the transition.

A transition-definition is a *ParallelTransition* if it spawns two or more parallel output paths, i. e. $ParallelTransition \in T$ if:

$$|CO| > 1$$

A transition-definition is a *DisjunctTransition* if it picks a single output path from a set of possible output paths, i. e. $DisjunctTransition \in T$ if:

$$\exists C \in CO \wedge C \notin DLAC_{OWF}$$

Similarly to *MergeTransition* transition-definitions, *DisjunctTransition* transition-definitions establish outgoing paths through a union. This means that the flow must continue through only one of the possible set of output paths in the union.

Finally, a transition-definition is a *ConditionalTransition* if it establishes at least one condition, i. e. $ConditionalTransition \in T$ if:

$$|Cond| \geq 1$$

If the transition-definition is a *BasicTransition*, then it can be reduced to the following representation in the $TBox_{OWF}$, which does not require the existence of an *AR* (but a role restriction instead), i.e.:

$$\forall C, D : C \in CI \wedge D \in CO \Rightarrow [C \sqsubseteq \exists transitionTo.D]_{TBox_{OWF}}$$

5.4.3 Example No. 1: Flow Synchronization

A transition-definition for synchronizing two incoming activity-definitions can be defined as $RDef1 = (AR, T, CI, CO, Cond)$, where:

- $AR = TR1$;
- $T = \{SynchronizationTransition\}$;
- $CI = \{IncomingActivity1, IncomingActivity2\}$;
- $CO = \{OutgoingActivity\}$;
- $Cond = \emptyset$.

This structure would translate to the following $TBox, \rho$:

- $\lfloor TR1 \sqsubseteq SynchronizationTransition \rfloor_{\rho}$;
- $\lfloor TR1 \sqsubseteq \exists from.IncomingActivity1 \rfloor_{\rho}$;
- $\lfloor TR1 \sqsubseteq \exists from.IncomingActivity2 \rfloor_{\rho}$;
- $\lfloor TR1 \sqsubseteq \exists to.OutgoingActivity \rfloor_{\rho}$.

Because $RDef1$ is of type *SynchronizationTransition*, the instantiation and execution of both incoming activity-definitions must be concluded so that the outgoing activity-definitions can be instantiated and executed.

5.4.4 Example No. 2: Flow Merge

A transition-definition for merging two incoming activity-definitions can be defined as $RDef2 = (AR, T, CI, CO, Cond)$ in a $TBox \rho$, where:

- $AR = TR2$;
- $T = \{MergeTransition\}$;
- $CI = \{X\}$ with $\lfloor X \equiv IncomingActivity1 \sqcup IncomingActivity2 \rfloor_{\rho}$;
- $CO = \{OutgoingActivity\}$;
- $Cond = \emptyset$.

This structure would translate to the following $TBox, \rho$:

- $\lfloor TR2 \sqsubseteq MergeTransition \rfloor_{\rho}$;
- $\lfloor TR2 \sqsubseteq \exists from.(IncomingActivity1 \sqcup IncomingActivity2) \rfloor_{\rho}$;

- $\lfloor TR2 \sqsubseteq \exists to.OutgoingActivity \rfloor_{\rho}$.

Because $RDef2$ is of type *MergeTransition*, the instantiation and execution of only one of the incoming activity-definitions must be concluded so that the outgoing activity-definitions can be instantiated and executed.

5.4.5 Example No. 3: Flow Parallelization

A parallel transition-definition for two outgoing activity-definitions can be defined as $RDef3 = (AR, T, CI, CO, Cond)$, where:

- $AR = TR3$;
- $T = \{ParallelTransition, BasicTransition\}$;
- $CI = \{IncomingActivity\}$;
- $CO = \{OutgoingActivity1, OutgoingActivity2\}$;
- $Cond = \emptyset$.

This structure would translate to the following $TBox, \rho$:

- $\lfloor TR3 \sqsubseteq ParallelTransition \rfloor_{\rho}$;
- $\lfloor TR3 \sqsubseteq BasicTransition \rfloor_{\rho}$;
- $\lfloor TR3 \sqsubseteq \exists from.IncomingActivity \rfloor_{\rho}$;
- $\lfloor TR3 \sqsubseteq \exists to.OutgoingActivity1 \rfloor_{\rho}$;
- $\lfloor TR3 \sqsubseteq \exists to.OutgoingActivity2 \rfloor_{\rho}$.

Since this is a *BasicTransition*, alternatively, the following $TBox, \rho$, can be formed:

- $\lfloor IncomingActivity \sqsubseteq \exists transitionTo.OutgoingActivity1 \rfloor_{\rho}$;
- $\lfloor IncomingActivity \sqsubseteq \exists transitionTo.OutgoingActivity2 \rfloor_{\rho}$.

Because $RDef3$ is of type *ParallelTransition*, both outgoing activity-definitions will be instantiated and executed.

5.4.6 Example No. 4: Flow Disjunction

A disjunct transition-definition for two outgoing activity-definitions can be defined as $RDef4 = (AR, T, CI, CO, Cond)$ in a $TBox \rho$, where:

- $AR = TR4$;
- $T = \{DisjunctTransition, BasicTransition\}$;
- $CI = \{IncomingActivity\}$;
- $CO = \{X\}$ with $\lfloor X \equiv OutgoingActivity1 \sqcup OutgoingActivity2 \rfloor_\rho$;
- $Cond = \emptyset$.

This structure would translate to the following $TBox, \rho$:

- $\lfloor TR4 \sqsubseteq DisjunctTransition \rfloor_\rho$;
- $\lfloor TR4 \sqsubseteq BasicTransition \rfloor_\rho$;
- $\lfloor TR4 \sqsubseteq \exists from.IncomingActivity \rfloor_\rho$;
- $\lfloor TR4 \sqsubseteq \exists to.(OutgoingActivity1 \sqcup OutgoingActivity2) \rfloor_\rho$.

Since this is a *BasicTransition*, alternatively, a $TBox, \rho$ with $\lfloor IncomingActivity \sqsubseteq \exists transitionTo.(OutgoingActivity1 \sqcup OutgoingActivity2) \rfloor_\rho$ can be formed.

Because *RDef4* is of type *DisjunctTransition*, only one of the outgoing activity-definitions will be instantiated and executed.

5.4.7 Example No. 5: Flow Conditions

A conditional transition-definition between two activity-definitions can be defined as $RDef5 = (AR, T, CI, CO, Cond)$, where:

- $AR = TR5$;
- $T = \{ConditionalTransition\}$;
- $CI = \{IncomingActivity\}$;
- $CO = \{OutgoingActivity\}$;
- $Cond = \{\text{"}\forall x, y : C(x) \wedge D(y) \Rightarrow R(x, y)\text{"}$.

This structure would translate to the following $TBox, \rho$:

- $\lfloor TR5 \sqsubseteq ConditionalTransition \rfloor_\rho$;
- $\lfloor TR5 \sqsubseteq \exists from.IncomingActivity \rfloor_\rho$;
- $\lfloor TR5 \sqsubseteq \exists to.OutgoingActivity \rfloor_\rho$;
- $\lfloor TR5 \sqsubseteq hasCondition : \text{"}\forall x, y : C(x) \wedge D(y) \Rightarrow R(x, y)\text{"} \rfloor_\rho$.

Because $RDef5$ is of type *ConditionalTransition*, the outgoing activity-definition will be instantiated and executed only if the specified conditions are satisfied. For instance, in $RDef5$, the *OutgoingActivity* activity-definition will only be instantiated if the condition $\forall x, y : C(x) \wedge D(y) \Rightarrow R(x, y)$ applied on top of the operational $ABox$ evaluates to *true* (i. e. all individuals of C and D must be related through R).

5.5 WORKFLOW-DEFINITIONS

A workflow-definition is the representation of a workflow. It is a graph of activity-definitions (i. e. task-definitions, event-definitions and sub-workflow-definitions) linked by transition-definitions.

5.5.1 Definition

Formally, a workflow-definition is a structure $WfDef(OWF) := (AW, T, AD, TRD, IED, \diamond_{WF}, Cond, p, l, d)$, where:

- $AW \in DLAC_{OWF}$ is the atomic workflow-definition concept;
- $T \in \{Workflow, Loop\}$ is the atomic concept that defines the type of the workflow-definition;
- $AD \subseteq DLAC_{OWF}$ is the set of atomic concepts representing the activity-definitions;
 - $SAD \subseteq AD$ is the subset of atomic concepts representing the activity-definitions that start the workflow-definition;
 - $TD \subseteq AD$ is the subset of atomic concepts representing the task-definitions;
 - $ED \subseteq AD$ is the subset of atomic concepts representing the event-definitions;
 - $WD \subseteq AD$ is the subset of atomic concepts representing the sub-workflow-definitions;
- $TRD \subseteq DLAC_{OWF}$ is the set of atomic concepts representing transition-definitions between activity-definitions;

- $IED \subseteq DLAC_{OWF}$ is the set of atomic concepts representing event-definitions of type *InstantiationEvent* that trigger the instantiation and execution of the workflow-definition;
- $\diamond_{WF} : DLAC_{OWF} \times DLAC_{OWF}$ is a transitive dependency relation that associates an atomic input concept of a task-definition to an atomic operational concept (event, assignment, input or output) of an activity-definition;
- $Cond \subseteq DLv_{OWF}$ is a set of exit conditions, required only for loops;
- $p \in PD_{OCF}$ is the individual indicating the priority of the workflow-definition;
- $l \in DLv_{OWF}$ is the individual containing the label of the workflow-definition;
- $d \in DLv_{OWF}$ is the individual containing the description of the workflow-definition.

The function $isWfDef : DLAC_{OWF} \rightarrow \{0,1\}$ indicates if an atomic concept represents a workflow-definition.

The structure $WfDef$ is a valid workflow-definition if the dependencies between activity-definitions are valid and it satisfies the following conditions:

- All of its concepts are satisfiable with respect to $TBox_{OWF}$;
- $\forall C \in AD \Rightarrow [C \sqsubseteq Activity]_{TBox_{OWF}}$;
- $\forall C \in TD \Rightarrow [C \sqsubseteq Task]_{TBox_{OWF}}$;
- $\forall C \in ED \Rightarrow [C \sqsubseteq Event]_{TBox_{OWF}}$;
- $\forall C \in WD \Rightarrow [C \sqsubseteq Workflow]_{TBox_{OWF}}$;
- $\forall C \in TRD \Rightarrow [C \sqsubseteq Transition]_{TBox_{OWF}}$;
- $\forall C \in IED \Rightarrow [C \sqsubseteq InstantiationEvent]_{TBox_{OWF}}$;
- $SAD \neq \emptyset$;
- $\forall C \in AD \Rightarrow C \in SAD^+$, where $SAD^+ = \bigcup_{n>0} SAD^n$, $SAD^1 = SAD$ and $SAD^n = \{D | \exists C, X : C \in SAD^{n-1} \wedge X \in TRD \wedge C \in frDef_{CI}(X) \wedge D \in frDef_{CO}(X)\}$;
- $\forall A1, C : A1 \in TD \wedge C \in fTDef_{IOI}(A1) \Rightarrow \exists X : (C, X) \in fTDef_{\Delta_{OD}}(A1) \vee (C, X) \in \diamond_{WF}$;
- $\forall A1, C : A1 \in TD \wedge C \in fTDef_{IOO}(A1) \Rightarrow \exists X : (C, X) \in fTDef_{\Delta_{OD}}(A1) \vee (\exists D : [C \sqsubseteq D]_{TBox_{OWF}} \wedge D \in fTDef_{IOI}(A1))$;
- $T = Loop$ if $Cond \neq \emptyset$.

The structure $WfDef$ also results in the following additional expressions in the $TBox_{OWF}$ and $AnBox_{OWF}$:

- $\lfloor AW \sqsubseteq T \rfloor_{TBox_{OWF}}$;
- $\forall X \in SAD \Rightarrow \lfloor AW \sqsubseteq \exists hasStartActivity.X \rfloor_{TBox_{OWF}}$;
- $\lfloor AW \sqsubseteq \exists hasInstantiationEvent.D \rfloor_{TBox_{OWF}} \wedge \lfloor D \equiv \bigsqcup_{C \in IED}(C) \rfloor_{TBox_{OWF}}$ if $IED \neq \emptyset$;
- $\lfloor AW \sqsubseteq hasPriority : p \rfloor_{TBox_{OWF}}$;
- $\forall ec \in Cond \Rightarrow \lfloor AW \sqsubseteq hasExitCondition : ec \rfloor_{TBox_{OWF}}$;
- $\lfloor label(AW, l) \rfloor_{AnBox_{OWF}} \wedge \lfloor description(AW, d) \rfloor_{AnBox_{OWF}}$.

5.5.2 Dependencies on Task-Definitions

The input concepts of a task-definition represent all the data associated with each instantiation (and each assignment), prior to any inherent operations. In this sense, any data dependencies to other task/event-definitions must be established onto input concepts. Output concepts can only rely on (and be related to) the input concepts of their task-definition. Thus, they cannot be dependent on the operational concepts of other task/event-definitions.

The relation \diamond_{WF} establishes dependencies between the input concepts of a task-definition, and the operational concepts of task/event-definitions. For instance, if C is an output concept of the task-definition A , D is an input concept of a task-definition B , and $(D, C) \in \diamond_{WF}$, then all individuals of D must also be individuals of C . I.e. the input individuals of D , of a task represented by B , must always be output individuals of C , of a task represented by A . Thus, in the context of the workflow-definition, the relation \diamond_{WF} can be expressed in pure DL through a subsumption. For two concepts C and D with $(C, D) \in \diamond_{WF}$, then $C \sqsubseteq D$.

Given the set of atomic concepts representing task-definitions and event-definitions in the workflow-definition, dependencies in \diamond_{WF} can only be established between an atomic input concept of a task-definition and an atomic operational concept of a task-definition or event-definition, i.e.:

$$\begin{aligned} \forall C, D : (C, D) \in \diamond_{WF} \Rightarrow \\ \exists A1, A2 : A1 \in TD \wedge A2 \in (TD \cup ED) \wedge C \in fTDef_{IOI}(A1) \wedge \\ ((isTDef(A2) \wedge D \in fTDef_{OP}(A2)) \vee (isEDef(A2) \wedge D \in fEDef_{OP}(A2))) \end{aligned}$$

For such a dependency, the domain mappings, Δ_{OD} , between dependent operational concepts must be the same, i.e.:

$$\begin{aligned} \forall C, D, A1, A2 : (C, D) \in \diamond_{WF} \wedge A1 \in TD \wedge A2 \in (TD \cup ED) \wedge C \in fTDef_{IOI}(A1) \wedge \\ ((isTDef(A2) \wedge D \in fTDef_{OP}(A2)) \vee (isEDef(A2) \wedge D \in fEDef_{OP}(A2))) \Rightarrow \\ \{X | (C, X) \in fTDef_{\Delta_{OD}}(A1)\} = \{Y | (isTDef(A2) \wedge (D, Y) \in fTDef_{\Delta_{OD}}(A2)) \vee \\ (isEDef(A2) \wedge (D, Y) \in fEDef_{\Delta_{OD}}(A2))\} \end{aligned}$$

Consequently, two dependent operational concepts must have the same domain type in Δ_{OD} . This ensures the satisfiability of all operational concepts when the relations Δ_{OD} and \diamond_{WF} are interpreted as DL subsumptions.

If the previous conditions are satisfied, the dependencies in \diamond_{WF} are valid and a special dependency is also established between the atomic concepts that represent both activity-definitions, $A1$ and $A2$. This special dependency is denoted by $A2 \prec A1$ ($A1$ depends on $A2$), where \prec is a transitive relation:

$$\begin{aligned} \forall A1, A2 \exists C, D : (C, D) \in \diamond_{WF} \wedge A1 \in TD \wedge A2 \in (TD \cup ED) \wedge C \in fTDef_{IOI}(A1) \wedge \\ ((isTDef(A2) \wedge D \in fTDef_{OP}(A2)) \vee (isEDef(A2) \wedge D \in fEDef_{OP}(A2))) \Rightarrow \\ A2 \prec A1 \end{aligned}$$

A task-definition may depend on itself, i. e. $(A1, A1) \in \prec^+$, where \prec^+ is the transitive closure of the \prec relation. In such situations, a flow loop must be established in order to provide an exit condition. This can be achieved either through a *Loop* sub-workflow-definition or through a *ConditionalTransition* transition-definition.

5.5.3 Inferring Transition-Definitions from Dependencies

Basic transition-definitions in *TRD* can be inferred from dependencies between task/event-definitions as follows:

$$\begin{aligned} \forall C, D : (C, D) \in \prec^- \wedge C \neq D \Rightarrow \exists X : X \in DLAC_{OWF} \wedge X \in TRD \wedge isRDef(X) \wedge \\ BasicTransition \in fRDef_T(X) \wedge fRDef_{CI}(X) = \{C\} \wedge fRDef_{CO}(X) = \{D\} \wedge \\ fRDef_{Cond} = \emptyset \end{aligned}$$

Where \prec^- is the transitive reduction on the \prec relation.

5.5.4 Aggregation of Redundant Results

The instantiation and execution of a task-definition may result in new output data to the operational **ABox**. Thus, when $au > 1$ (i. e. when there is more than one assignment per unit), the instantiation and execution of a task-definition may result in redundant and possibly inconsistent data in the operational **ABox**. In order to merge and aggregate these data and remove inconsistencies and redundancy from the operational **ABox**, a *FilterTask* task-definition must be specified.

A *FilterTask* task-definition (represented by the atomic concept $AT\alpha$) for the aggregation of assignments from a target task-definition (represented by the atomic concept AT) always satisfies the special dependency $AT \prec AT\alpha$. Given a target task-definition, $TDef$, its assignment aggregation *FilterTask* task-definition, $T\alpha Def$, can be automatically derived through an augmented replica of the operational $TBox$ of $TDef$. The required steps are presented in [Algorithm 1](#), where:

- $TDef = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, W, I, p, l, d, au)$;
- $T\alpha Def = (AT\alpha, T\alpha, AA\alpha, IO\alpha, IOBox\alpha, AR\alpha, ARBox\alpha, \Delta_{\alpha OD}, \Delta_{\alpha OCF}, W\alpha, I\alpha, p\alpha, l\alpha, d\alpha, au\alpha)$.

Algorithm 1: The assignment aggregation task-definition automatic construction process.

```

New Task-Definition  $T\alpha Def$ 
Set  $T\alpha = \{MajorityVotingFilterTask\}$ 
Set  $AR\alpha = AR$ 
Set  $p\alpha = p$ 

// create the unit context concept of the previous assignment in  $T\alpha Def$ 
New UnitContext concept  $AAE \in IOI\alpha$  such that  $(AAE, UnitContext) \in \Delta_{\alpha OCF}$ 
 $AAE$  depends on  $AA$ :  $(AAE, AA) \in \diamond_{WF}$ 
Rel.  $AA\alpha$  to  $AAE$ :  $[AA\alpha \sqsubseteq \exists hasUnitContext.AAE]_{ARBox\alpha}$ 

// create unit concepts in  $T\alpha Def$ 
 $\forall C, R: C \in IOI \wedge (C, Unit) \in \Delta_{OCF} \wedge (AA \xrightarrow{R} C)_{ARBox}$ 
New Unit concept  $D \in IOI\alpha$  such that  $(D, Unit) \in \Delta_{\alpha OCF}$ 
 $D$  depends on  $C$ :  $(D, C) \in \diamond_{WF}$ 
Rel.  $AA\alpha$  to  $D$ :  $exactC(AA\alpha \xrightarrow{R} D)_{ARBox\alpha} = 1$ 
Rel.  $AAE$  to  $D$ :  $exactC(AAE \xrightarrow{R} D)_{IOBox\alpha} = 1$ 

```



```

// create remaining unit context concepts in  $T\alpha Def$ 
 $\forall C, R : C \in (IOI \cup IOO) \wedge (C, Unit) \notin \Delta_{OCF} \wedge (AA \xrightarrow{R} C)_{ARBox}$ 
  New UnitContext concept  $D \in IOI\alpha$  such that  $(D, UnitContext) \in \Delta_{OCF}$ 
   $D$  depends on  $C : (D, C) \in \diamond_{WF}$ 
  Rel.  $AA\alpha$  to  $D : (AA\alpha \xrightarrow{R} D)_{ARBox\alpha} \wedge$ 
     $minC(AA\alpha \xrightarrow{R} D)_{ARBox\alpha} = minC(AA \xrightarrow{R} C)_{ARBox}$ 
  Rel.  $AAE$  to  $D : sameR(AA \xrightarrow{R} C, AAE \xrightarrow{R} D)_{(ARBox, IOBox\alpha)}$ 

// establish remaining relationships in the  $IOBox\alpha$  present in the  $IOBox$ 
 $\forall C1, D1 : C1 \in (IOI \cup IOO) \wedge D1 \in (IOI\alpha \cup IOO\alpha) \wedge (D1, C1) \in \diamond_{WF}$ 
 $\forall C2, D2 : C2 \in (IOI \cup IOO) \wedge D2 \in (IOI\alpha \cup IOO\alpha) \wedge (D2, C2) \in \diamond_{WF}$ 
 $\forall R : sameR(C1 \xrightarrow{R} C2, D1 \xrightarrow{R} D2)_{(IOBox, IOBox\alpha)}$ 

// create the response concept in  $T\alpha Def$ 
New Response concept  $AAO \in IOO\alpha$  such that  $(AAO, Response) \in \Delta_{OCF}$ 
 $AAO$  depends on  $AAE : [AAO \sqsubseteq AAE]_{IOBox\alpha}$ 
Rel.  $AA\alpha$  to  $AAO : [AA\alpha \sqsubseteq = hasResponse.AAO]_{ARBox\alpha}$ 

```

The remaining attributes of the $T\alpha Def$ task-definition must be manually established. The au value, in particular, is important, since it establishes the amount of different votes that must be given to a set of redundant assignments.

5.5.5 Example No. 1: Aggregation of Task-Definition Results

Operational concepts from other task-definitions may be used as part of the input specification of a new task-definition, which leads to a dependency. These dependencies can be used to specify a filter task-definition that aggregates the redundant assignments (those with the same units and unit context) of another task-definition. The workflow-definition, partially depicted in [Figure 23](#), contains a sequence of two task-definitions that illustrate this scenario.

The workflow-definition is defined as $WfDef1 = (AW, T, AD, TRD, IED, \diamond_{WF}, Cond, medium, l, d)$, where:

- $AW = WF1$;
- $T = Workflow$;
- $AD = \{A1T, A2T\}$;
- $SAD = \{A1T\}$;

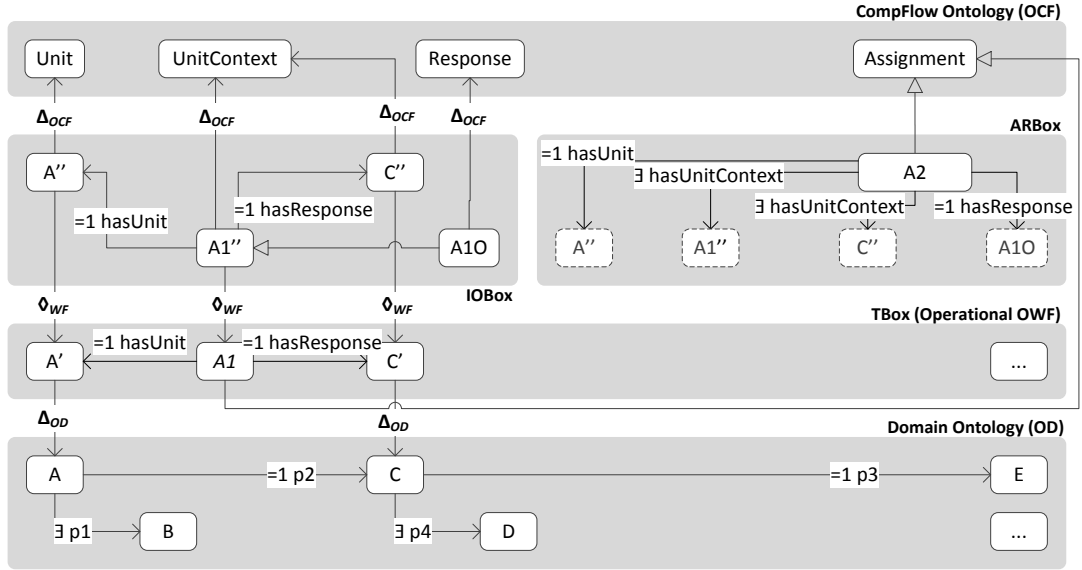


Figure 23: Operational DL representation of a dependent task-definition that filters the assignments of a previous task-definition.

- $TRD = \{T1T2\}$ where $RDef_{T1T2} = (T1T2, \{BasicTransition\}, \{A1T\}, \{A2T\}, \emptyset)$;
- $IED = \emptyset$;
- $\diamond_{WF} = \{(A1'', A1), (A'', A'), (C'', C')\}$;
- $Cond = \emptyset$.

The two task-definitions represented by $A1T$ and $A2T$ are:

- $T1Def1$ (with the assignment concept $A1$), which demands eight redundant assignments per unit;
- $T2Def1$ (with the assignment concept $A2$), which merges and aggregates the redundant assignments of $T1Def1$ through the application of a majority voting strategy that demands six votes for each redundant set of $T1Def1$ assignments.

The task-definition $T1Def1$ is defined as $T1Def1 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A1TW, A1TI, medium, l, d, 8)$, where:

- $AT = A1T$;
- $T = \{CreateAndFillTask\}$;
- $AA = A1$;
- $IOI = \{A'\}$;
- $IOO = \{C'\}$;
- $IOBox = \emptyset$;

- $AR = \emptyset$;
- $\lfloor A1 \sqsubseteq = 1hasUnit.A' \rfloor_{ARBox}$ and $\lfloor A1 \sqsubseteq = 1hasResponse.C' \rfloor_{ARBox}$;
- $\Delta_{OD} = \{(A', A), (C', C)\}$;
- $\Delta_{OCF} = \{(A', Unit), (C', Response)\}$;
- $\lfloor A1TW \equiv Worker \rfloor_{TBox_{OWF}}$ (all workers);
- $\lfloor A1TI \equiv TaskInterface \rfloor_{TBox_{OWF}}$ (all task interfaces).

The task-definition $T2Def1$ is defined as $T2Def1 = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, A2TW, A2TI, medium, l, d, 6)$, where:

- $AT = A2T$;
- $T = \{MajorityVotingFilterTask\}$;
- $AA = A2$;
- $IOI = \{A1'', A'', C''\}$;
- $IOO = \{A1O\}$;
- $\lfloor A1'' \sqsubseteq = 1hasResponse.C'' \rfloor_{IOBox}$, $\lfloor A1'' \sqsubseteq = 1hasUnit.A'' \rfloor_{IOBox}$ and $\lfloor A1O \sqsubseteq A1'' \rfloor_{IOBox}$;
- $AR = \emptyset$;
- $\lfloor A2 \sqsubseteq = 1hasUnit.A'' \rfloor_{ARBox}$, $\lfloor A2 \sqsubseteq \exists hasUnitContext.A1'' \rfloor_{ARBox}$, $\lfloor A2 \sqsubseteq \exists hasUnitContext.C'' \rfloor_{ARBox}$ and $\lfloor A2 \sqsubseteq = 1hasResponse.A1O \rfloor_{ARBox}$;
- $\Delta_{OD} = \{(A'', A), (C'', C)\}$;
- $\Delta_{OCF} = \{(A'', Unit), (A1'', UnitContext), (C'', UnitContext), (A1O, Response)\}$;
- $\lfloor A2TW \equiv Worker \rfloor_{TBox_{OWF}}$ (all workers);
- $\lfloor A2TI \equiv TaskInterface \rfloor_{TBox_{OWF}}$ (all task interfaces).

The $T2Def1$ task-definition defines a task where, given a set of redundant assignments (with the same unit) from $T1Def1$, the worker will be asked to select the assignment with the best or most appropriate response.

An instance of $T2Def1$ must have six assignments per unit (six votes for each unit). However, since $T2Def1$ is a *FilterTask* task-definition, its assignments will be the subject to the application of a filter or voting strategy in order to remove the least selected or unselected assignments of $T1Def1$ from the operational [ABox](#).

5.5.6 Example No. 2: Text Partition and Translation

The workflow-definition depicted in [Figure 24](#) represents a more concrete example, where a workflow-definition for partitioning and translating (sections of) a document is defined. Such a workflow-definition, according to the document ontology in [Figure 9](#), requires at least three task-definitions:

1. Partitions sections into paragraphs;
2. Translates paragraphs;
3. Assembles translated paragraphs into translated sections.

The text partition and translation workflow-definition is defined as $WfDef2 = (AW, T, AD, TRD, IED, \diamond_{WF}, Cond, medium, l, d)$, where:

- $AW = TranslationWorkflow$;
- $T = Workflow$;
- $AD = \{T1, T2, T3\}$;
- $TRD = \{TR12, TR23\}$;
- $IED = \emptyset$;
- $T1Def = (T1, \{CreateAndFillTask\}, T1Assignment, IO, IOBox, \emptyset, ARBox, \Delta_{OD}, \Delta_{OCF}, W, l, p, l, d, 1)$;
 - $IO = \{SectionUnit_T1, ParagraphResponse_T1\}$;
 - $[SectionUnit_T1 \sqsubseteq \exists contains.ParagraphResponse_T1]_{IOBox}$;
 - $[T1Assignment \sqsubseteq= 1hasUnit.SectionUnit_T1]_{ARBox}$ and $[T1Assignment \sqsubseteq \exists hasResponse.ParagraphResponse_T1]_{ARBox}$;
 - $\Delta_{OD} = \{(SectionUnit_T1, Section), (ParagraphResponse_T1, Paragraph)\}$;
 - $\Delta_{OCF} = \{(SectionUnit_T1, Unit), (ParagraphResponse_T1, Response)\}$;
- $T2Def = (T2, \{CreateAndFillTask\}, T2Assignment, IO, \emptyset, \emptyset, ARBox, \Delta_{OD}, \Delta_{OCF}, W, l, p, l, d, 1)$;
 - $IO = \{ParagraphUnit_T2, ParagraphResponse_T2\}$;
 - $[T2Assignment \sqsubseteq= 1hasUnit.ParagraphUnit_T2]_{ARBox}$ and $[T2Assignment \sqsubseteq= 1hasResponse.ParagraphResponse_T2]_{ARBox}$;
 - $\Delta_{OD} = \{(ParagraphUnit_T2, Paragraph), (ParagraphResponse_T2, Paragraph)\}$;

- $\Delta_{OCF} = \{(ParagraphUnit_T2, Unit), (ParagraphResponse_T2, Response)\};$
- $T3Def = (T3, \{CreateAndFillTask\}, T3Assignment, IO, IOBox, \emptyset, ARBox, \Delta_{OD}, \Delta_{OCF}, W, I, p, l, d, 1);$
 - $IO = \{SectionUnit_T3, SectionResponse_T3, T2AssignmentUC_T3, OrigParagUC_T3, TransParagUC_T3\};$
 - $[T2AssignmentUC_T3 \sqsubseteq = 1hasUnit.OrigParagUC_T3]_{IOBox},$
 $[T2AssignmentUC_T3 \sqsubseteq = 1hasResponse.TransParagUC_T3]_{IOBox},$
 $[SectionUnit_T3 \sqsubseteq \exists contains.OrigParagUC_T3]_{IOBox}$ and
 $[SectionResponse_T3 \sqsubseteq \exists contains.TransParagUC_T3]_{IOBox};$
 - $[T3Assignment \sqsubseteq = 1hasUnit.SectionUnit_T3]_{ARBox},$
 $[T3Assignment \sqsubseteq = 1hasResponse.SectionResponse_T3]_{ARBox},$
 $[T3Assignment \sqsubseteq \exists hasUnitContext.T2AssignmentUC_T3]_{ARBox},$
 $[T3Assignment \sqsubseteq \exists hasUnitContext.OrigParag_T3]_{ARBox}$ and
 $[T3Assignment \sqsubseteq \exists hasUnitContext.TransParagUC_T3]_{ARBox};$
 - $\Delta_{OD} = \{(SectionUnit_T3, Section), (SectionResponse_T3, Section),$
 $(OrigParagUC_T3, Paragraph), (TransParagUC_T3, Paragraph)\};$
 - $\Delta_{OCF} = \{(SectionUnit_T3, Unit), (SectionResponse_T3, Response),$
 $(OrigParagUC_T3, UnitContext), (TransParagUC_T3, UnitContext),$
 $(T2AssignmentUC_T3, UnitContext)\};$
- $TR12Def = (TR12, \{BasicTransition\}, \{T1\}, \{T2\}, \emptyset);$
- $TR23Def = (TR23, \{BasicTransition\}, \{T2\}, \{T3\}, \emptyset);$
- $\diamond_{WF} = \{(ParagraphUnit_T2, ParagraphResponse_T1),$
 $(T2AssignmentUC_T3, T2Assignment)\};$
- $W = WPart$ and $[WPart \equiv Worker]_{TBox_{OWF}}$ (all workers);
- $I = IPart$ and $[IPart \equiv TaskInterface]_{TBox_{OWF}}$ (all task interfaces).

Notice that in $T3$, the assignments of $T2$ are included as contextual information. This allows workers to translate sections by assembling previously translated paragraphs.

The transitions between tasks are inferred from the dependencies between task-definitions.

Activity-definitions and transition-definitions are specified in an ontology that represents a top-level workflow-definition. These definitions (in the [TBox](#), [RBox](#) and [AnBox](#)) fully represent workflows and can be instantiated multiple times (in an [ABox](#)). Workflow-definitions import and extend an already existent domain ontology, which partially describes the input and output data flowing throughout the workflow. This allows the re-usability of already existent domain ontologies.

The full semantics of the operations that must be performed for each task are captured by task-definitions through augmented extensions of the domain ontology. The type of each task-definition depends on how these extensions are modelled. This allows the automatic instantiation and execution of task-definitions without the need for a specific implementation for each task-definition.

INSTANTIATION AND EXECUTION OF WORKFLOW-DEFINITIONS

A top-level workflow-definition (found in a workflow-definition ontology) can be instantiated and executed multiple times inside a job. The instantiation of a workflow-definition follows a closed-world assumption and creates workflows (instances of the workflow-definition) that fully satisfy the restrictions present in the workflow-definition ontology.

The instantiation and execution of a workflow-definition leads to the incremental instantiation and execution of (as the workflow execution progresses):

- Sub-workflow-definition instances, called sub-workflows;
- Task-definition instances, called tasks;
- Event-definition instances, called events;
- Transition-definition instances, called transitions.

These instances are created in the **ABox** of the job's knowledge base, defined as $KBJ = (ADL_{KBJ}, ABox_{KBJ}, O_{KBJ})$, where:

- OWF is the workflow-definition ontology imported by O_{KBJ} , i. e. $O_{KBJ} \triangleright OWF$;
- $OABox \subseteq ABox_{KBJ}$ is the operational **ABox** of the job.

The job's ontology is the union of all the workflow-definition ontologies that must be instantiated and executed in the job. The operational **ABox** contains the input and output data, shared and accessible by all the workflows in the job.

6.1 TASK-DEFINITION INSTANTIATION AND EXECUTION

The instantiation and execution of a task-definition (or the creation and execution of a task) imply several steps. [Figure 25](#) depicts the flow diagram of the proposed process.

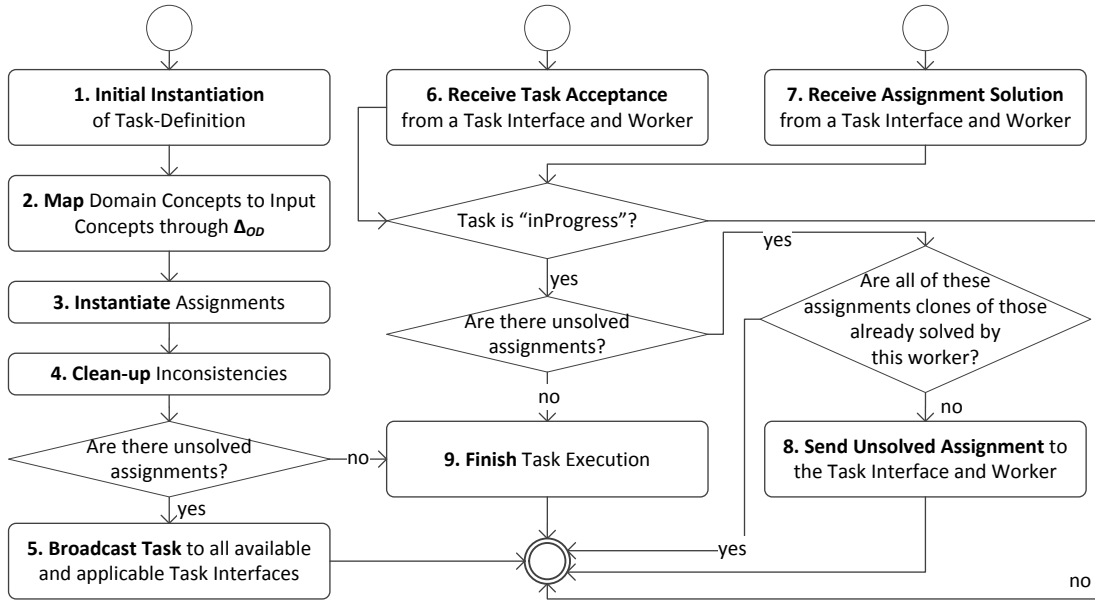


Figure 25: Overview of the task-definition instantiation and execution process.

6.1.1 Definition of Task

Formally, a task is a structure $TDInst(KBJ) := (t, s, p, l, d, au, AI, DI, DIBox, RIBox, TI, AW, PW, TDef)$, where:

- $t \in ADLn_{KBJ}$ is the task individual;
- $s \in SD_{OCF}$ is the individual in OCF indicating the current state of the task;
- $p \in PD_{OCF}$ is the individual in OCF indicating the priority of the task;
- $l \in ADLv_{KBJ}$ is the datatype value containing the label of the task;
- $d \in ADLv_{KBJ}$ is the datatype value containing the description of the task;
- $au \in ADLv_{KBJ}$ is the datatype value containing the amount of assignments per unit of the task;
- $AI \subseteq ADLn_{KBJ}$ is the set of assignment individuals;
- $DI \subseteq ADLn_{KBJ}$ is the set of input and output individuals;
 - $DII \subseteq DI$ is the subset of input individuals;
 - $DIO \subseteq DI$ is the set of output individuals;
- $DIBox \subseteq OABox$ is the set of assertions that relate and establish each input and output individual;

- $DIIBox \subseteq DIBox$ is the set of assertions that relate and establish input individuals;
- $DIOBox \subseteq DIBox$ is the set of assertions that relate and establish output individuals;
- $RIBox \subseteq OABox$ is the set of assertions that relate each assignment to each input and output individual;
- $TI \subseteq ADLn_{KBJ}$ is the set of task interfaces that received the task;
- $AW \subseteq ADLn_{KBJ}$ is the set of workers that accepted the task;
- $PW \subseteq ADLn_{KBJ}$ is the set of workers that solved at least one assignment;
- $TDef = (AT, T, AA, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, W, I, p_{TDef}, l_{TDef}, d_{TDef}, au_{TDef})$ is the task-definition.

$TDInst$ is an instance of $TDef$ if it establishes a consistent $ABox$ with respect to the $TBox$ established by $TDef$, plus:

- $\lfloor AT(t) \rfloor_{ABox_{KBJ}}$;
- $\lfloor hasState(t, s) \rfloor_{ABox_{KBJ}}$;
- $\lfloor hasPriority(t, p) \rfloor_{ABox_{KBJ}}$;
- $\lfloor label(t, l) \rfloor_{ABox_{KBJ}}$;
- $\lfloor description(t, d) \rfloor_{ABox_{KBJ}}$;
- $\lfloor hasAssignmentsPerUnit(t, au) \rfloor_{ABox_{KBJ}}$;
- $\forall a \in AI \Rightarrow \lfloor AA(a) \rfloor_{OABox} \wedge \lfloor hasOperationalization(t, a) \rfloor_{ABox_{KBJ}}$;
- $\forall x \in DI \Rightarrow \exists X : X \in (IOI \cup IOO) \wedge \lfloor X(x) \rfloor_{OABox}$;
- The $DIBox$ must be consistent with respect to the $IOBox$;
- $\forall x, X, R : x \in DI \wedge X \in (IOI \cup IOO) \wedge \lfloor X(x) \rfloor_{OABox} \wedge (AA \xrightarrow{R} X)_{ARBox} \Rightarrow \exists a \in AI : \lfloor R(a, x) \rfloor_{OABox}$;
- The $RIBox$ must be consistent with respect to the $ARBox$;
- $\forall i \in TI \Rightarrow I(i) \wedge \lfloor executedThrough(t, i) \rfloor_{ABox_{KBJ}} \wedge \exists a \in AI : \lfloor executedThrough(a, i) \rfloor_{OABox}$;
- $\forall aw \in AW \Rightarrow \lfloor W(aw) \rfloor_{ABox_{KBJ}} \wedge \lfloor acceptedBy(t, aw) \rfloor_{ABox_{KBJ}}$;
- $\forall pw \in PW \Rightarrow \lfloor W(pw) \rfloor_{ABox_{KBJ}} \wedge \lfloor performedBy(t, pw) \rfloor_{ABox_{KBJ}} \wedge \exists a \in AI : \lfloor executedThrough(a, pw) \rfloor_{OABox}$.

6.1.2 Instantiation

The task-definition instantiation process is divided into four steps: (1) initial instantiation, (2) domain concept to input concept mapping, (3) assignment instantiation, and (4) inconsistency clean-up.

INITIAL INSTANTIATION (1): Given a task-definition, $TDef$, an initial instantiation $TDInst = (t, s, p, l, d, au, AI, DI, DIBox, RIBox, TI, AW, PW, TDef)$ can be performed automatically as follows:

- Create an individual $t \in ABox_{KBJ}$;
- $s = notStarted$;
- $p = p_{TDef}$;
- $l = l_{TDef}$;
- $d = d_{TDef}$;
- $au = au_{TDef}$.

DOMAIN CONCEPT TO INPUT CONCEPT MAPPING (2): Afterwards, a mapping between domain concepts and input concepts of $TDef$, onto the operational $ABox$ is also performed automatically through Δ_{OD} .

The mapping process classifies domain individuals that fulfil the conditions specified by the input concepts of the task-definition and the role restrictions between them. This process, presented in [Algorithm 2](#), is performed onto the operational $ABox$.

Algorithm 2: The domain concept to input concept mapping process in the task-definition instantiation.

```

// it. all individuals, x, in the operational ABox, and all input concepts, C
∀x, C : C ∈ IOI ∧ x ∈ OABox
  // check if the individual has the domain type of C
  // and if it has as type all the dependencies of C
  If (∀X : (C, X) ∈ ΔOD ⇒ [X(x)]OABox) and (∀X : (C, X) ∈ ◇WF ⇒ [X(x)]OABox) and
  // also check if all the role restrictions of C are satisfied by x
  (∀X : [C ⊆ X]IOBox ∧ X ∉ IOO ∧ (∄R, D : D ∈ IOO ∧ (C  $\xrightarrow[R]{X}$  D)IOBox) ⇒ [X(x)]OABox)
  then
    // if all these conditions are satisfied, then x is of type C
    [C(x)]DIBox and x ∈ DI

```

ASSIGNMENT INSTANTIATION (3): From the previous mapping and classification, the assignment instantiation of $TDInst$ is performed in three sub-steps, (a) unit association, (b) context association, (c) assignment cloning.

The unit association process (a) creates one assignment per unit as presented in [Algorithm 3](#).

Algorithm 3: The unit association process in the task-definition instantiation.

```

// CList is initialized with all unit concepts
Init. the list CList = ( $\leq, X$ ),  $X = \{C \mid C \in IOI \wedge (C, Unit) \in \Delta_{OCF} \wedge (AA \xrightarrow{R} C)_{ARBox}\}$ 
Init. the list IList = ( $\leq, \emptyset$ )
Invoke CreateAssignments (CList, 0, IList)

/* this routine creates assignments all possible combinations of individuals of
the unit concepts in CList */
Start of routine CreateAssignments (CList, Lvl, IList)
If Lvl  $\geq$  size of CList then
  Init.  $i = 0$ 
  While  $i < Lvl$ 
    Create new assignment  $a \in AI$ 
     $\forall R : (AA \xrightarrow{R} CList[i])_{ARBox} \Rightarrow [R(a, IList[i])]_{RIBox}$ 
  Exit CreateAssignments

For each individual  $x$  of concept CList[Lvl]
  Set IList[Lvl] =  $x$ 
  Invoke CreateAssignments (CList, Lvl + 1, IList)
End of routine CreateAssignments

```

The context association process (b), presented in [Algorithm 4](#), defines the input context of each assignment.

Algorithm 4: The context association process in the task-definition instantiation.

```

 $\forall a \in AI$ 
/* TODO is initialized with all input concepts (the order prioritizes top-
level concepts) */
Init. the list TODO = ( $\leq, X$ ),  $X = \{C \mid C \in IOI\}$  ordered ascendantly by
 $am = \left| \{D \mid \exists R : (D \xrightarrow{R} C)_{IOBox}\} \right|$ 
Init. DONE =  $\emptyset$ 

// associate all individuals related to the assignment's unit individuals
 $\forall C : C \in IOI \wedge (C, Unit) \in \Delta_{CFO}$ 

```

```

    Invoke FillAssignment (a, C, TODO ∪ {C}, DONE)
  /* if TODO is not empty, then there are some unit context individuals,
     unrelated to unit individuals, that must be associated */
  While size of TODO > 0
    Init. C to the first concept in TODO
     $\forall c : (AA \xrightarrow{R_C} C)_{ARBox} \wedge [C(c)]_{DIBox} \Rightarrow [RC(a,c)]_{RIBox}$ 
    Invoke FillAssignment (a, C, TODO, DONE)

  /* this routine associates individuals related to the already associated input
     individuals (of C) in a */
  Start of routine FillAssignment (a, C, TODO, DONE)
  If C not in TODO then Exit FillAssignment
  Remove C from TODO
  Set DONE = DONE ∪ {C}
  If size of TODO is 0 then Exit FillAssignment

   $\forall R_C, c : (AA \xrightarrow{R_C} C)_{ARBox} \wedge [C(c)]_{OABox} \wedge [R_C(a,c)]_{RIBox}$ 
   $\forall D : D \in TODO$ 
   $\forall R, R_D, d : (C \xrightarrow{R} D)_{IOBox} \wedge (AA \xrightarrow{R_D} D)_{ARBox} \wedge$ 
   $\wedge [D(d)]_{DIBox} \wedge [R(c,d)]_{OABox} \Rightarrow [R_D(a,d)]_{RIBox}$ 
   $\forall R, R_D, d : (D \xrightarrow{R} C)_{IOBox} \wedge (AA \xrightarrow{R_D} D)_{ARBox} \wedge$ 
   $\wedge [D(d)]_{DIBox} \wedge [R(d,c)]_{OABox} \Rightarrow [R_D(a,d)]_{RIBox}$ 
  Invoke FillAssignment (a, D, TODO, DONE)
  End of routine FillAssignment

```

Finally, as presented in [Algorithm 5](#), each assignment is cloned (*c*) to satisfy the amount of assignments per unit in *au*.

Algorithm 5: The assignment cloning process in the task-definition instantiation.

```

  Init.  $AI' = \emptyset$ 
   $\forall a \in AI$ 
  Init.  $i = au - 1$ 
  While  $i > 0$ 
    Clone a into  $a' \in AI'$ 
    Set  $i = i - 1$ 
  Set  $AI = AI \cup AI'$ 

```

INCONSISTENCY CLEAN-UP (4): Assignments that result in an inconsistent operational *ABox* must be removed and all domain individuals not related to at least one assignment must be unmapped (reverse of step 2). Also, if any of the new assignments

is equivalent to an assignment of a previous task (i. e. its task-definition is the same and it has exactly the same input data), then it must be removed. This allows new tasks in loops to consider only new assignments, instead of also considering those that have already been solved.

6.1.3 Execution

The task-definition execution process starts with the broadcast of the task through all available and applicable task interfaces (step 5). Afterwards, all allowed workers will be able to accept the task, which leads to step 6. When a worker accepts a task, an unsolved assignment (if available) will be sent to the worker (step 8). When the worker submits a solved assignment (leading to step 7), another unsolved assignment is sent to the worker until no further assignments are available. An unsolved assignment is only available to a worker if the worker has not previously solved a clone of the assignment (containing the same input data).

6.2 EVENT-DEFINITION INSTANTIATION AND EXECUTION

The instantiation and execution of an event-definition (or the creation and execution of an event) is performed differently according to its type (*RunningEvent* or *Instantiation-Event*).

The instantiation and execution of *InstantiationEvent* event-definitions is integrated into the instantiation and execution of workflow-definitions, which is described in [Section 6.4](#).

RunningEvent event-definitions, however, are components in the flow of the workflow-definition. The steps involved in the instantiation and execution of a *RunningEvent* event-definition are presented in [Figure 26](#).

The execution consists in waiting for an event occurrence (or instance of the event-definition) to arrive through an event interface. After the arrival of an event, the event is stored in the operational [ABox](#) and the workflow execution continues.

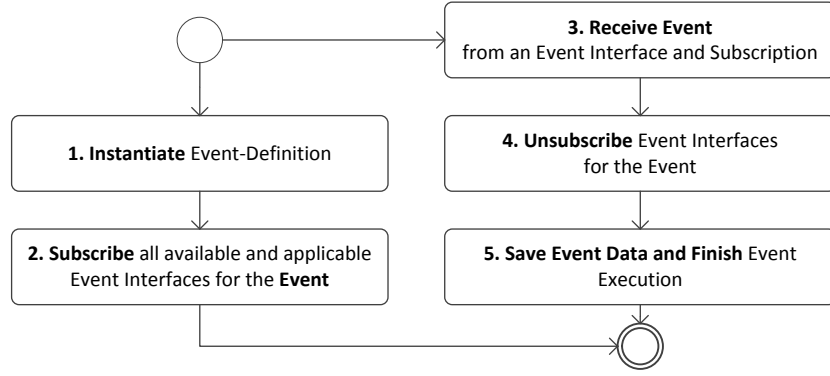


Figure 26: Overview of the *RunningEvent* event-definition instantiation and execution process.

6.2.1 Definition of Event

Formally, an event is a structure $EDInst(KBJ) := (e, s, p, l, d, i, DI, DIBox, RIBox, EDef)$, where:

- $e \in ADLn_{KBJ}$ is the event individual;
- $s \in SD_{CF}$ is the individual in OCF indicating the current state of the event;
- $p \in PD_{CF}$ is the individual in OCF indicating the priority of the event;
- $l \in ADLv_{KBJ}$ is the datatype value containing the label of the event;
- $d \in ADLv_{KBJ}$ is the datatype value containing the description of the event;
- $i \in ADLn_{KBJ}$ is event interface individual that triggered the event;
- $DI \subseteq ADLn_{KBJ}$ is the set of data individuals;
- $DIBox \subseteq OABox$ is the set of assertions that relate each data individuals;
- $RIBox \subseteq OABox$ is the set of assertions that relate e to each input and output individuals;
- $EDef = (AE, T, IO, IOBox, AR, ARBox, \Delta_{OD}, \Delta_{OCF}, I, p_{EDef}, l_{EDef}, d_{EDef})$ is the event-definition.

$EDInst$ is an instance of $EDef$ if it establishes a consistent $ABox$ with respect to the $TBox$ established by $EDef$, plus:

- $[AE(e)]_{OABox}$;
- $[hasState(e, s)]_{OABox} \wedge [hasPriority(e, p)]_{OABox}$;
- $[label(e, l)]_{OABox} \wedge [description(e, d)]_{OABox}$;
- $[I(i)]_{OABox} \wedge [executedThrough(e, i)]_{OABox}$;

- $\forall x \in DI \Rightarrow \exists X \in IOI : \lfloor X(x) \rfloor_{OABox}$;
- The *DIBox* must be consistent with respect to the *IOBox*;
- $\forall x, X, R : x \in DI \wedge X \in IOI \wedge \lfloor X(x) \rfloor_{OABox} \wedge (AE \xrightarrow{R} X)_{ARBox} \Rightarrow \lfloor R(e, x) \rfloor_{OABox}$;
- The *RIBox* must be consistent with respect to the *ARBox*.

6.2.2 Instantiation

Given an event-definition *EDef*, a partial instantiation (1) $EDInst = (e, s, p, l, d, i, DI, DIBox, RIBox, EDef)$ can be performed automatically as follows:

- Create an individual $e \in OABox$;
- $s = notStarted$;
- $p = p_{EDef}$;
- $l = l_{EDef}$;
- $d = d_{EDef}$.

6.2.3 Execution

The execution of an event consists in five main steps. First, a subscription to all applicable event interfaces is performed (2). Afterwards, the execution process must wait for an occurrence of the event-definition to arrive (3). Since only one occurrence of the event-definition is desired for a *RunningEvent*, the previously performed subscriptions must be cancelled (4). Finally, the event occurrence and its data must be added to the operational *ABox* (5), completing the initial instantiation of the event.

6.3 TRANSITION-DEFINITION INSTANTIATION AND EXECUTION

The instantiation and execution of a transition-definition (or the creation and execution of a transition) depends on how it is classified. Accordingly, the steps involved in the instantiation and execution of a transition-definition are presented in [Figure 27](#).

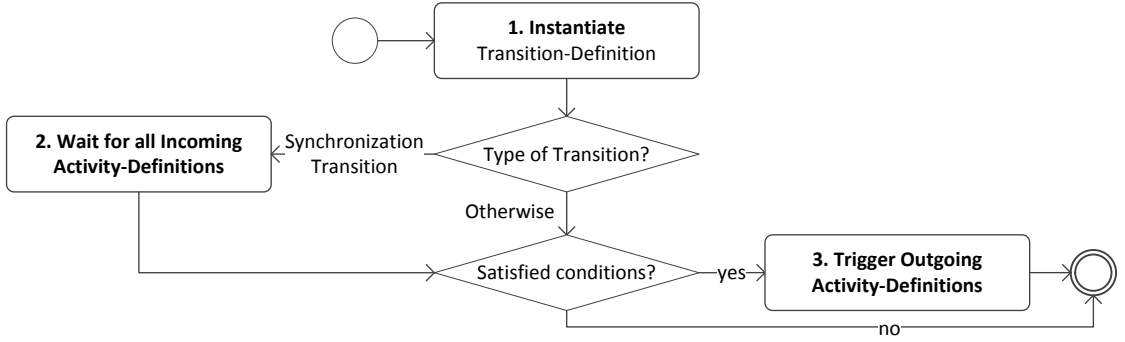


Figure 27: Overview of the transition-definition instantiation and execution process.

6.3.1 Definition of Transition

Formally, a transition is a structure $RDInst(KBJ) := (r, CI, CO, Cond, RDef)$, where:

- $r \in ADLn_{OWF}$ is the transition individual;
- $CI \subseteq ADLn_{OWF}$ is the set of incoming activity individuals;
- $CO \subseteq ADLn_{OWF}$ is the set of outgoing activity individuals;
- $Cond \subseteq ADLv_{OWF}$ is the set of conditions that must be satisfied for this transition to be performed;
- $RDef = (AR, T, CI_{RDef}, CO_{RDef}, Cond_{RDef})$ is the transition-definition.

Accordingly, the functions in Table 16 are considered for this definition.

$RDInst$ is an instance of $RDef$ if it establishes a consistent $ABox$ with respect to the $TBox$ established by $RDef$, plus:

- $\lfloor AR(r) \rfloor_{ABox_{KBJ}}$;
- $\forall c \in CI \Rightarrow \exists C \in CI_{RDef} : \lfloor C(c) \rfloor_{ABox_{KBJ}} \wedge \lfloor from(r, c) \rfloor_{ABox_{KBJ}}$;
- $\forall c \in CO \Rightarrow \exists C \in CO_{RDef} : \lfloor C(c) \rfloor_{ABox_{KBJ}} \wedge \lfloor to(r, c) \rfloor_{ABox_{KBJ}}$;
- $\forall c \in Cond \Rightarrow \lfloor hasCondition(r, c) \rfloor_{ABox_{KBJ}}$;

FUNCTION	DESCRIPTION
$fRDInst_{CI} : ADLn_{OWF} \rightarrow 2^{ADLn_{OWF}}$	given r returns its CI , i. e. $fRDInst_{CI}(r) = CI$
$fRDInst_{CO} : ADLn_{OWF} \rightarrow 2^{ADLn_{OWF}}$	given r returns its CO , i. e. $fRDInst_{CO}(r) = CO$

Table 16: Additional functions for transitions.

- $\forall D : (AR \xrightarrow{from} D)_{TBox_{OWF}} \wedge D \in CI \Rightarrow \exists d : [D(d)]_{ABox_{KBJ}} \wedge [from(r, d)]_{ABox_{KBJ}}$;
- $\forall D : (AR \xrightarrow{to} D)_{TBox_{OWF}} \wedge D \in CO \Rightarrow \exists d : [D(d)]_{ABox_{KBJ}} \wedge [to(r, d)]_{ABox_{KBJ}}$.

6.3.2 Instantiation

Given a transition-definition $RDef$, a partial instantiation (\mathfrak{t}) $RInst = (r, CI, CO, Cond, RDef)$ can be performed automatically as follows:

- Create an individual $r \in ABox_{KBJ}$;
- $Cond = Cond_{RDef}$.

6.3.3 Execution

During the execution of a transition, the execution process must wait for an incoming activity for each of the activity-definitions defined in CI_{RDef} . Since at least one of these activities is already finished when the transition-definition instantiation and execution process is triggered, the transition execution process must only wait if the transition is a *SynchronizationTransition* (step 2). A similar process happens for outgoing activity-definitions in CO_{RDef} (step 3). Each activity-definition defined in CO_{RDef} will be instantiated and executed only once. If a union concept is present in CO_{RDef} , one of the activity-definitions represented in the union concept will be randomly selected for instantiation and execution.

The instantiation and execution of outgoing activity-definitions is only triggered if all the conditions of the transition are satisfied.

6.4 WORKFLOW-DEFINITION INSTANTIATION AND EXECUTION

The instantiation of a workflow-definition is performed incrementally, i.e. activity-definitions are instantiated only when they are reached and their execution is imminent. In this sense, a partial initial instantiation of the workflow-definition is performed when its execution starts (see [Figure 28](#)).

The instantiation and execution of workflow-definitions can be triggered by a special kind of events called instantiation events.

The partial initial instantiation consists in (i) creating an individual of the workflow-definition’s atomic concept and (ii) setting all role values for this individual, according to the role restrictions and annotations in the workflow-definition ontology, i. e. state, priority, label, description and the exit condition (if applicable).

6.4.1 Definition of Workflow

Formally, a workflow is a structure $WfInst(KBJ) := (wf, s, p, l, d, ec, ie, AC, RAC, WfDef)$, where:

- $wf \in ADLn_{OWF}$ is the workflow individual;
- $s \in SD_{OCF}$ is the individual in OCF indicating the current state of the workflow;
- $p \in PD_{OCF}$ is the individual in OCF indicating the priority of the workflow;
- $l \in ADLv_{OWF}$ is the datatype value containing the label of the workflow;
- $d \in ADLv_{OWF}$ is the datatype value containing the description of the workflow;
- $ec \in ADLv_{OWF}$ is the datatype value exit condition of the loop (if applicable);

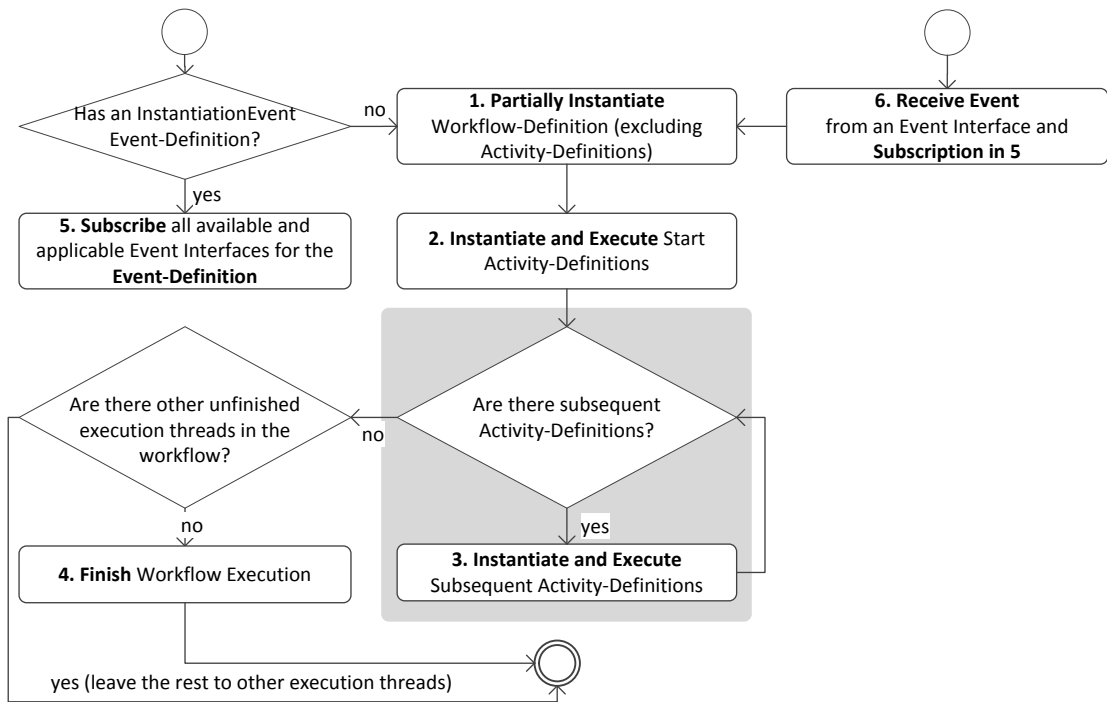


Figure 28: Overview of the workflow-definition instantiation and execution process.

- $ie \in ADLn_{OWF}$ is the event individual that triggered the creation and execution of the workflow (if applicable);
- $AC \subseteq ADLn_{OWF}$ is the set of activity individuals in the workflow;
 - $SAC \subseteq ABox_{KBJ}$ is the set of activity individuals that start the workflow;
 - $CAC \subseteq ABox_{KBJ}$ is the set of activity individuals with state *inProgress*;
- $RAC \subseteq ABox_{KBJ}$ is the set of transition individuals;
- $WfDef = (AW, T, AD, TRD, IED, \diamond_{WF}, Cond_{WfDef}, p_{WfDef}, l_{WfDef}, d_{WfDef})$ is the workflow-definition.

$WfInst$ is an instance of $WfDef$ if it establishes a consistent $ABox$ with respect to the $TBox$ established by $WfDef$, plus:

- $[AW(wf)]_{ABox_{KBJ}}$;
- $[hasState(wf, s)]_{ABox_{KBJ}} \wedge [hasPriority(wf, p)]_{ABox_{KBJ}}$;
- $[label(wf, l)]_{ABox_{KBJ}} \wedge [description(wf, d)]_{ABox_{KBJ}}$;
- $[hasExitCondition(wf, ec)]_{ABox_{KBJ}}$ if there is an ec ;
- $[hasInstantiationEvent(wf, ie)]_{ABox_{KBJ}}$ if $IED \neq \emptyset$;
- $\forall a \in AC \Rightarrow \exists A : A \in AD \wedge [A(a)]_{ABox_{KBJ}} \wedge [hasActivity(wf, a)]_{ABox_{KBJ}}$;
- $\forall a, b, r : r \in RAC \wedge a \in fRDInst_{CI}(r) \wedge b \in fRDInst_{CO}(r) \Rightarrow \exists A, B, R : A \in AD \wedge B \in AD \wedge R \in TRD \wedge A \in fRDef_{CI}(R) \wedge B \in fRDef_{CO}(R)$;
- $\forall a \in SAC \Rightarrow \exists A : A \in SAD \wedge [A(a)]_{ABox_{KBJ}} \wedge [hasStartActivity(wf, a)]_{ABox_{KBJ}}$;
- $\forall a \in CAC \Rightarrow \exists A : A \in AD \wedge [A(a)]_{ABox_{KBJ}} \wedge [hasCurrentActivity(wf, a)]_{ABox_{KBJ}} \wedge [hasState(a, inProgress)]_{ABox_{KBJ}}$.

6.4.2 Instantiation

Given a workflow-definition $WfDef$, an initial partial instantiation $WfInst = (wf, s, p, l, d, ec, ie, AC, RAC, WfDef)$ can be performed automatically as follows:

- Create an individual $wf \in ABox_{KBJ}$;
- $s = notStarted$;
- $p = p_{WfDef}$;
- $l = l_{WfDef}$;

- $d = d_{WfDef}$;
- $ec = ec_{WfDef}$.

6.4.3 Event-Driven Instantiation

If the workflow-definition has an associated *InstantiationEvent*, the instantiation and execution process will be triggered for each occurrence of the *InstantiationEvent*. Consequently, all applicable event interfaces must be subscribed (step 5) and the execution engine must wait for the occurrence of events, which will lead to step 6.

6.4.4 Execution

The execution of a workflow starts with the instantiation and execution of all activity-definitions that start its workflow-definition (step 2). This step is repeated for all following activity-definitions once the execution of the previous activity-definition instance is finished (step 3). The workflow is only finished if there are no running parallel execution threads (4). This condition ensures that no activities are running and that there are no more activity-definitions left to instantiate and execute.

6.4.5 Example

Consider a job's knowledge base, $KB1 = (ADL_{KB1}, ABox_{KB1}, O_{KB1})$, where:

- *OWF* is the workflow-definition ontology presented in [Section 5.5.6](#) and imported by O_{KB1} (i. e. $O_{KB1} \triangleright OWF$);
- $OABox_{KB1} \subseteq ABox_{KB1}$ is the operational *ABox* of the job.

The input dataset inside the $OABox_{KB1}$ contains:

- $[Section(s1)]_{OABox_{KB1}}$;
- $[Section(s2)]_{OABox_{KB1}}$;
- $[Section(s3)]_{OABox_{KB1}}$;
- $[hasText(s1, "...")]_{OABox_{KB1}}$;
- $[hasText(s2, "...")]_{OABox_{KB1}}$;

- $[hasText(s3, "...")]_{OABox_{KB1}}$;

Also, there is a set of available worker individuals $w1$, $w2$ and $w3$ of $WPart$, and a set of available task interfaces $ti1$ and $ti2$, where only $ti1$ is an individual of $IPart$. Both $w1$ and $w2$ worker individuals are *accessibleThrough* the task interface individual $ti1$.

The instantiation and execution of the top-level workflow-definition, represented by the concept *TranslationWorkflow*, starts with its initial instantiation $WfInst1 = (wf1, s, p, l, d, ec, ie, AC, RAC, WfDef2)$:

- $s = notStarted$;
- $p = medium$;
- $l = "Translation Workflow"$;
- $d = "Please translate the given text from English to Portuguese"$;
- $ec = \emptyset$;
- No ie ;
- $AC = \emptyset$ and $RAC = \emptyset$;

Afterwards, the execution of $wf1$ starts, and the first activity-definition (T1) must be instantiated and executed.

As activity-definitions are instantiated, the activity individuals will be added to AC .

The workflow-definition $WfDef2$ contains only basic transition-definitions. Thus the instantiation of each transition is represented through the *transitionTo* relationship between activities.

INSTANTIATION AND EXECUTION OF T1: The evolution of the $OABox$ during the instantiation and execution of T1 is depicted in [Figure 29](#).

The **initial instantiation** of T1 is given by $TInst1 = (t1, s, p, l, d, au, AI, DI, DIBox, RIBox, TI, AW, PW, T1Def)$ in the $ABox_{KB1}$, where:

- $t1 \in SAC$;
- $s = notStarted$;
- $p = medium$;
- $l = "Section Partition"$;
- $d = "Please partition the given text into paragraphs"$;
- $au = 1$;

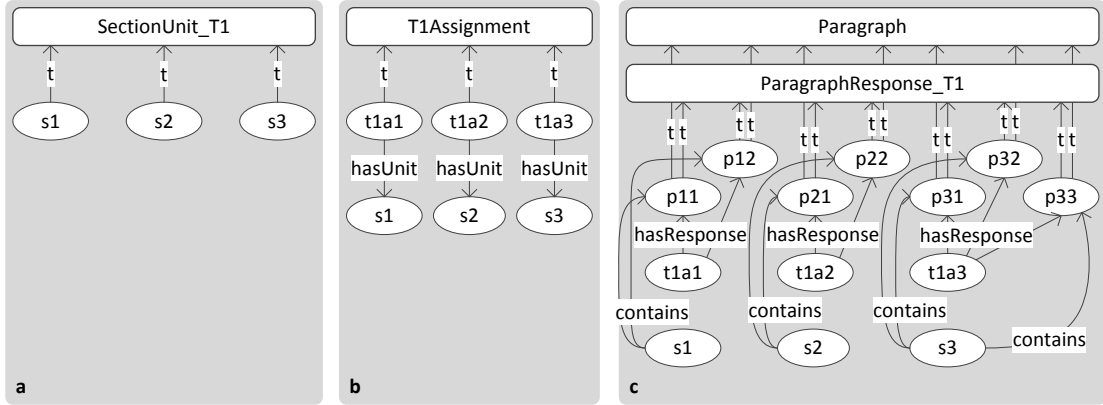


Figure 29: Evolution of the operational ABox during the instantiation and execution of T_1 (only new data is represented at each stage a, b and c; t is an abbreviation for *type*).

- $AI = \emptyset$;
- $DI = \emptyset$, $DIBox = \emptyset$ and $RIBox = \emptyset$;
- $TI = \emptyset$, $AW = \emptyset$ and $PW = \emptyset$.

The initial instantiation of t_1 is followed by the **domain concept to input concept mapping** step. Thus, the [Algorithm 2](#) is employed, leading to $DI = \{s_1, s_2, s_3\}$ and all of its individuals classified as $SectionUnit_T_1$ (a).

The **assignment instantiation** step begins with the unit association process, which creates one ($au = 1$) assignment for each individual of $SectionUnit_T_1$, i. e. $AI = \{t1a_1, t1a_2, t1a_3\}$ and the relationship $hasUnit$ is established between each assignment and its unit in the $RIBox$ (b). Due to the lack of $UnitContext$ concepts the context association process is skipped. Also, the assignments are not cloned since only one assignment per unit is required.

Finally, since all assignments are consistent with respect to the operational $TBox$, t_1 can be executed by **broadcasting the task** through all applicable interfaces, i. e. ti_1 . Similarly, only workers that are individuals of $WPart$ may participate in this task, i. e. w_1 and w_2 . The worker individual w_3 cannot participate since it is not accessible through an applicable task interface.

As specified in the task-definition T_1Def , for each assignment in AI , workers must provide one or more individuals of $ParagraphResponse_T_1$ (i. e. $Paragraph$) along with their datatype role values (i. e. $hasText$ and $hasOrdinal$). These assignments are submitted through ti_1 and the relationship $contains$ is established automatically.

If, meanwhile, the task t_1 is **accepted by the workers** w_1 and w_2 , then $AW = \{w_1, w_2\}$. Consequently, an **unsolved assignment will be sent** to both w_1 and w_2 .

Assuming that the **assignments are submitted** as follows:

- $w1$ solves $t1a1$ and submits the paragraphs $p11$ and $p12$ through $ti1$;
- $w1$ solves $t1a2$ and submits the paragraphs $p21$ and $p22$ through $ti1$;
- $w2$ solves $t1a3$ and submits the paragraphs $p31$, $p32$ and $p33$ through $ti1$.

Then, $DI = \{s1, s2, s3, p11, p12, p21, p22, p31, p32, p33\}$, $TI = \{ti1\}$ and $PW = \{w1, w2\}$. Furthermore, all the submitted paragraphs become individuals of both *ParagraphResponse_T1* and *Paragraph* in the *DIBox*. The relationships *hasResponse* (with the corresponding assignment) and *contains* (with the corresponding section) are automatically established for each paragraph (c).

After all assignments are solved, $t1$ is *finished* and the workflow continues to the instantiation and execution of $T2$.

INSTANTIATION AND EXECUTION OF T2: The evolution of the *OABox* during the instantiation and execution of $T2$ is depicted in Figure 30.

The **initial instantiation** of $T2$ is given by $TInst2 = (t2, s, p, l, d, au, AI, DI, DIBox, RIBox, TI, AW, PW, T2Def)$ in the $ABox_{KB1}$, where:

- $t2 \in AC$
- $s = notStarted$;

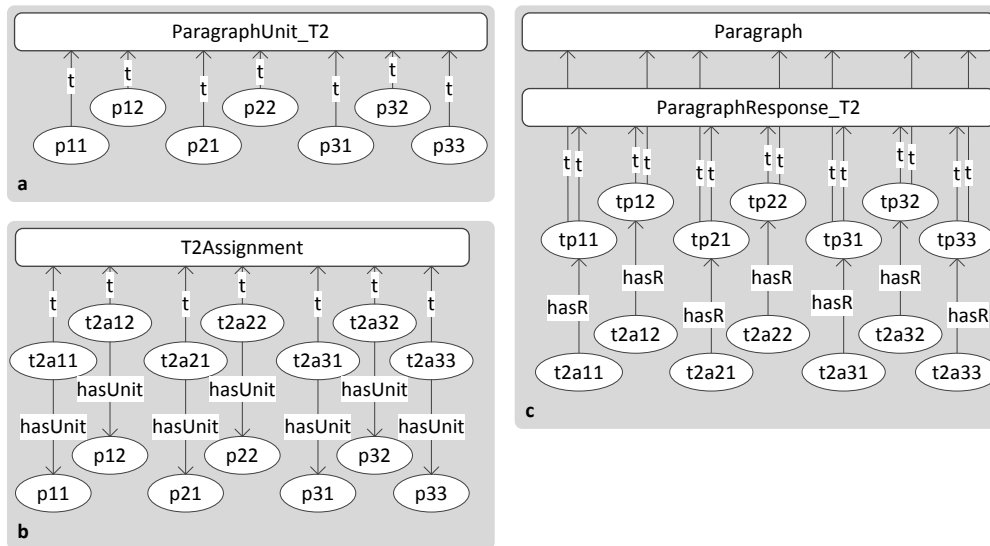


Figure 30: Evolution of the operational ABox during the instantiation and execution of $T2$ (only new data is represented at each stage a, b and c; t is an abbreviation for *type* and $hasR$ is an abbreviation for *hasResponse*).

- $p = \textit{medium}$;
- $l = \textit{“Paragraph Translation”}$;
- $d = \textit{“Please translate the given paragraph from English to Portuguese”}$;
- $au = 1$;
- $AI = \emptyset$;
- $DI = \emptyset, DIBox = \emptyset$ and $RIBox = \emptyset$;
- $TI = \emptyset, AW = \emptyset$ and $PW = \emptyset$.

The initial instantiation of t_2 is followed by the **domain concept to input concept mapping** step. This leads to $DI = \{p_{11}, p_{12}, p_{21}, p_{22}, p_{31}, p_{32}, p_{33}\}$, with all the paragraphs in DI becoming individuals of $ParagraphUnit_T_2$ in the $DIBox$ (a). Notice that only individuals of $ParagraphResponse_T_1$ are electable as the units of t_2 (as established by the dependency in \diamond_{WF}).

Afterwards, the **assignment instantiation** step begins with the unit association process, which creates one ($au = 1$) assignment for each individual of $ParagraphUnit_T_2$, i. e. $AI = \{t_{2a11}, t_{2a12}, t_{2a21}, t_{2a22}, t_{2a31}, t_{2a32}, t_{2a33}\}$ and the relationship $hasUnit$ is established between each assignment and its unit in the $RIBox$ (b).

The task t_2 is then executed following the same execution process described for t_1 . Thus, the completion of t_2 leads to new translated paragraphs $tp_{11}, tp_{12}, tp_{21}, tp_{22}, tp_{31}, tp_{32}$ and tp_{33} in DI and classified as $ParagraphResponse_T_2$ inside the $DIBox$ (c).

INSTANTIATION AND EXECUTION OF T_3 : The **initial instantiation** of T_3 is given by $TInst_3 = (t_3, s, p, l, d, au, AI, DI, DIBox, RIBox, TI, AW, PW, T_3Def)$ in the $ABox_{KB1}$, where:

- $t_3 \in AC$;
- $s = \textit{notStarted}$;
- $p = \textit{medium}$;
- $l = \textit{“Section Translation”}$;
- $d = \textit{“Please assemble the translated paragraphs into a revised translated section”}$;
- $au = 1$;
- $AI = \emptyset$;
- $DI = \emptyset, DIBox = \emptyset$ and $RIBox = \emptyset$;

- $TI = \emptyset$, $AW = \emptyset$ and $PW = \emptyset$.

The initial instantiation of $t3$ is followed by the **domain concept to input concept mapping** step. This, as depicted in Figure 31, leads to $DI = \{s1, s2, s3, p11, p12, p21, p22, p31, p32, p33, t2a11, t2a12, t2a21, t2a22, t2a31, t2a32, t2a33, tp11, tp12, tp21, tp22, tp31, tp32, tp33\}$, with all:

- Sections becoming individuals of $SectionUnit_T3$ in the $DIBox$;
- Original paragraphs becoming individuals of $OriginalParagUC_T3$ in the $DIBox$;
- Translated paragraphs becoming individuals of $TransParagUC_T3$ in the $DIBox$;
- Assignments of T2 becoming individuals of $T2AssignmentUC_T3$ in the $DIBox$.

At this stage, and since all input individuals are mapped, the **assignment instantiation** step begins with the unit association process, which creates one ($au = 1$) assignment for each individual of $SectionUnit_T3$, i.e. $AI = \{t3a1, t3a2, t3a3\}$ and the relationship $hasUnit$ is established between each assignment and its unit in the $RIBox$ (see Figure 32).

Since $t3$ contains $UnitContext$ concepts, the context association process is also required. Thus, the assignment $t3a1$, with the unit $s1$, will have the contextual individuals (related through $hasUnitContext$) $p11, p12, t2a11, t2a12, tp11$ and $tp12$. Similarly, the assignment $t3a2$ will have the contextual individuals $p21, p22, t2a21, t2a22, tp21$ and $tp22$. Finally, the assignment $t3a3$ will have the contextual individuals $p31, p32, p33, t2a31, t2a32, t2a33, tp31, tp32$ and $tp33$ (see Figure 32).

The task $t3$ is then executed following the same execution process described for $t1$ and $t2$. This execution results in three translated sections $ts1, ts2$ and $ts3$ in DI , which are classified as $SectionResponse_T3$ inside the $DIBox$ (see Figure 33).

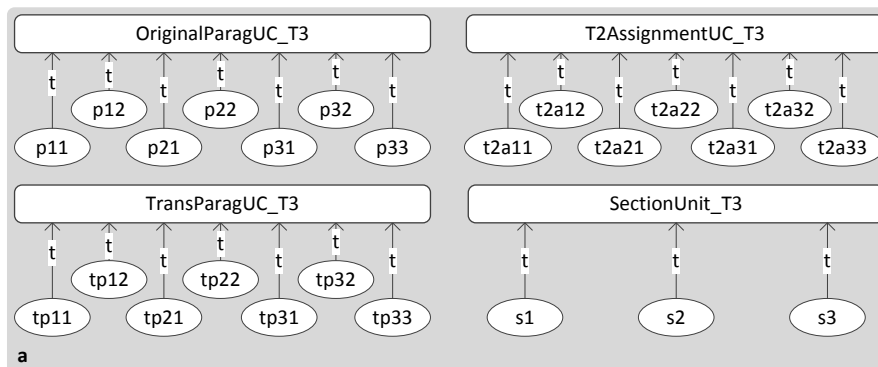


Figure 31: New data in the operational ABox after the domain concept to input concept mapping step of $T3$ (t is an abbreviation for *type*).

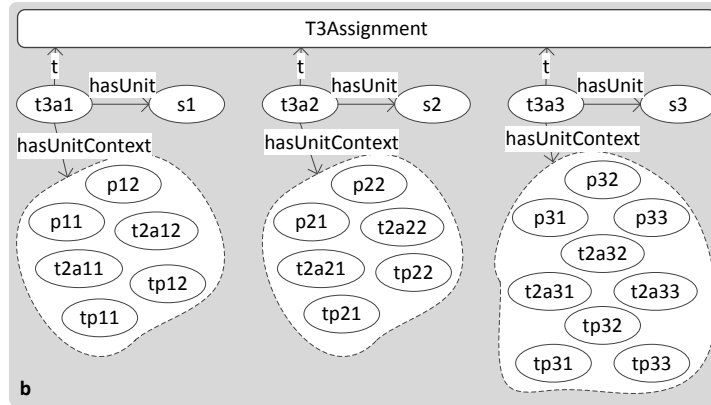


Figure 32: New data in the operational ABox after the unit and context association process of T3 (*t* is an abbreviation for *type*; relationships to dashed groups represent relationships to all its members).

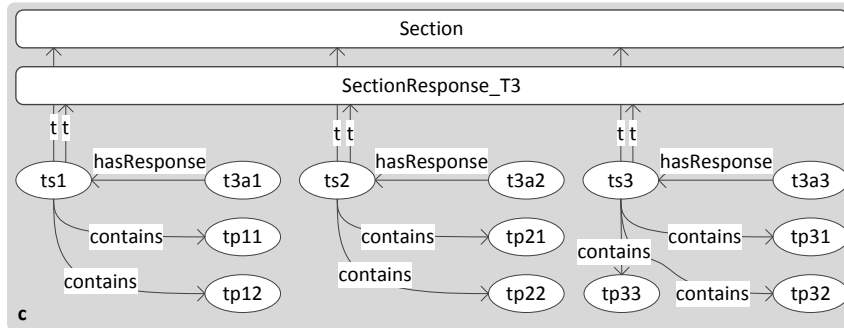


Figure 33: New data in the operational ABox after the execution of T3 (*t* is an abbreviation for *type*).

6.5 SUMMARY

Workflow-definition ontologies (representing a top-level workflow-definition) can be instantiated and executed multiple times inside a job. The job is a container of workflows, and their shared input and output data. Analogously, all workflows are executed in the scope of a job.

Through the application of a closed-world assumption, workflow-definitions can be automatically instantiated and executed without the need of specific implementations for each activity-definition. Furthermore, since the instantiation and execution of activity-definitions is incremental and performed only as the workflow execution progresses, the real-time modification of the workflow-definition with immediate effect in the instantiation and execution of the remaining part of the workflow is possible.

A distinction between the input and output data (inside the operational ABox), and workflow-related individuals is made to allow the domain data to be extracted without

any assertions dependent on the workflow-definition. This is important for environments where the output of the workflow needs to be pipelined into another entity that is only familiar with the domain ontology.

ASSISTED CONSTRUCTION OF WORKFLOW-DEFINITIONS

The CompFlow workflow-definition method establishes the ground rules required to build workflow-definition ontologies. Following these ground rules, a workflow-definition can be built using a construction tool that allows **CRUD** operations on top of the different types of activity-definitions and transition-definitions found in a workflow-definition. Simply allowing direct atomic **CRUD** operations (such as those supported by the Protégé ontology editor¹), however, would leave the process of building and defining activity-definitions a highly specialized process, requiring an expert in ontology representations. While the structure and semantics of domain ontologies are the main reason behind this drawback, they also offer several solutions.

Using the relations between domain concepts, established by role restrictions, it is possible to create workflow-definitions through a series of iterations that follow these relations and establish several atomic operations (task-definitions) on top of domain concepts. As illustrated in [Figure 34](#), and in order to reduce the degree of expertise required to create workflow-definitions, a semi-automatic construction environment for micro-task workflow-definitions from domain ontologies is proposed.

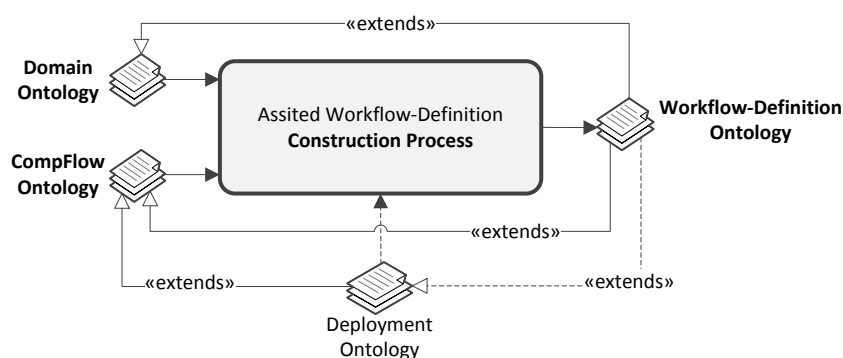


Figure 34: Overview of the CompFlow assisted construction process (dashed connections are optional).

¹<http://protege.stanford.edu/>

7.1 THE LAYERED ARCHITECTURE

The CompFlow construction process is driven by the domain expertise of the creator (the actor building the workflow-definition) to supervise the automatic interpretation of the domain ontology. This is achieved through a layered architecture that defines, for a workflow-definition and its current state, the set of operations that can be performed to further refine the workflow-definition (see [Figure 35](#) for an illustration).

7.1.1 Command Layers

As presented in [Figure 35](#), the assisted workflow-definition construction process relies on three command layers: (1) the atomic command layer, (2) the pattern command layer, and (3) the strategic command layer.

The atomic command layer defines the set of possible atomic operations that can be performed to create and define activity-definitions and transition-definitions. In this layer, a low-level structural analysis of the domain and workflow-definition ontologies is performed.

A pattern command, in the pattern command layer, is a set of atomic commands associated with a particular ontological pattern [16, 20]. These patterns may depend on the employed ontology construction methodology [4] and on the domain of the ontology. In the pattern command layer, a high-level structural and low-level semantic analysis of the domain and workflow-definition ontologies is performed.

The strategic command layer is an abstraction over the previous layers. Each strategy represents a set of atomic and pattern commands. They automate the construction process by restricting the set of possible choices presented to the creator during the workflow-definition construction process. For instance, a strategy for building workflow-definitions that result in recommendations (recommendation workflow-def-

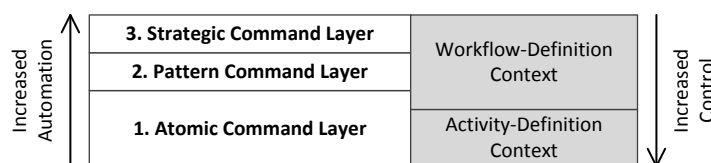


Figure 35: Layered architecture of the operations in the assisted workflow-definition construction process.

initions) could restrict the possible choices of the creator through its set of possible patterns and atomic operations. Through the strategic command layer, a high-level semantic analysis of the domain and workflow-definition ontologies is performed.

7.1.2 Command Contexts

The workflow-definition construction process moves between different types of contexts inside the workflow-definition. For each type of context, there is an associated set of possible atomic and pattern commands. The different types of contexts are directly related to transition-definitions and to each type of activity-definition. Therefore, the types of contexts are:

- The *workflow-definition context*;
- The *task-definition context*;
- The *event-definition context*;
- The *transition-definition context*.

Workflow-definition contexts may contain sub-workflow-definition contexts, forming a structure resembling a tree. Other types of contexts can only become leaves in such a tree. Thus, a task-definition context is always defined inside a workflow-definition context, which may actually be the sub-workflow-definition context of another workflow-definition context (see [Figure 36](#) for an example). The construction process is in one of these contexts at any given time.

The creation of an activity-definition or transition-definition typically starts with a create or update atomic command in a workflow-definition context. From this point onward, commands of the types present in the new child context will be available to the creator. Suggested commands only include those that add or increment data to the workflow-definition.

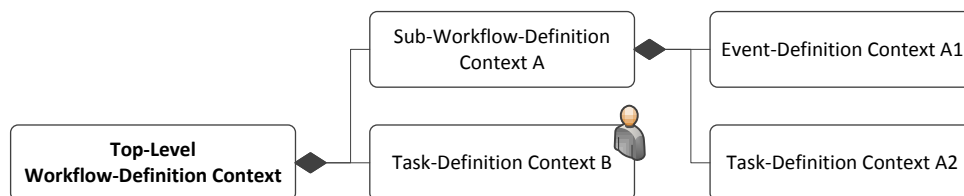


Figure 36: Example of a workflow-definition context hierarchy. The creator is currently found in the *Task-Definition Context B*.

An illustration of the transitions between contexts and their available commands is presented in [Figure 37](#).

7.2 ATOMIC COMMANDS

An activity-definition or transition-definition results from the execution of a set of atomic commands suggested to, or manually defined by, the creator. Multiple types of atomic commands can be performed in the context of a workflow-definition, task-definition, event-definition or transition-definition.

Each type of atomic command becomes applicable only if a set of conditions are satisfied. These conditions typically involve assessing if the structure is complete and valid. Thus, the selection and further triggering of a particular atomic command depends on the current context and in the state of the workflow-definition.

In a workflow-definition context, atomic commands allow creating, removing and updating task-definitions, event-definitions, sub-workflow-definitions and transition-definitions.

Once an atomic command to create or update a task-definition is performed, the process switches to the task-definition context. In this context, there are atomic commands for creating, removing and updating input and output concepts, role restrictions between input and output concepts, and dependencies.

Similarly, if an atomic command to create or update an event-definition is performed, the process switches to an event-definition context. In this context, atomic commands to create, delete and update data concepts and their role restrictions are also included.

An atomic command is only suggested if it adds new ontological data (e. g. an inexistent input or output concept in a task-definition) to the workflow-definition. Otherwise, its suggestion becomes redundant and unnecessary.

Workflow-definitions are always built in the scope of a top-level workflow-definition context. Thus, each context has two associated definitions: (i) the activity-definition or transition-definition of the context and (ii) the parent workflow-definition (non-existent in the top-level workflow-definition context). The parent workflow-definition (ii) of a context is defined as $WD(OWF) = (AW_{WD}, T_{WD}, AD_{WD}, TRD_{WD}, IED_{WD}, \diamond WD_{WF}, Cond_{WD}, p_{WD}, l_{WD}, d_{WD})$. Similarly, the top-level workflow-definition is defined as $TWD(OWF) = (AW_{TWD}, T_{TWD}, AD_{TWD}, TRD_{TWD}, IED_{TWD}, \diamond TWD_{WF}, Cond_{TWD}, p_{TWD}, l_{TWD}, d_{TWD})$.

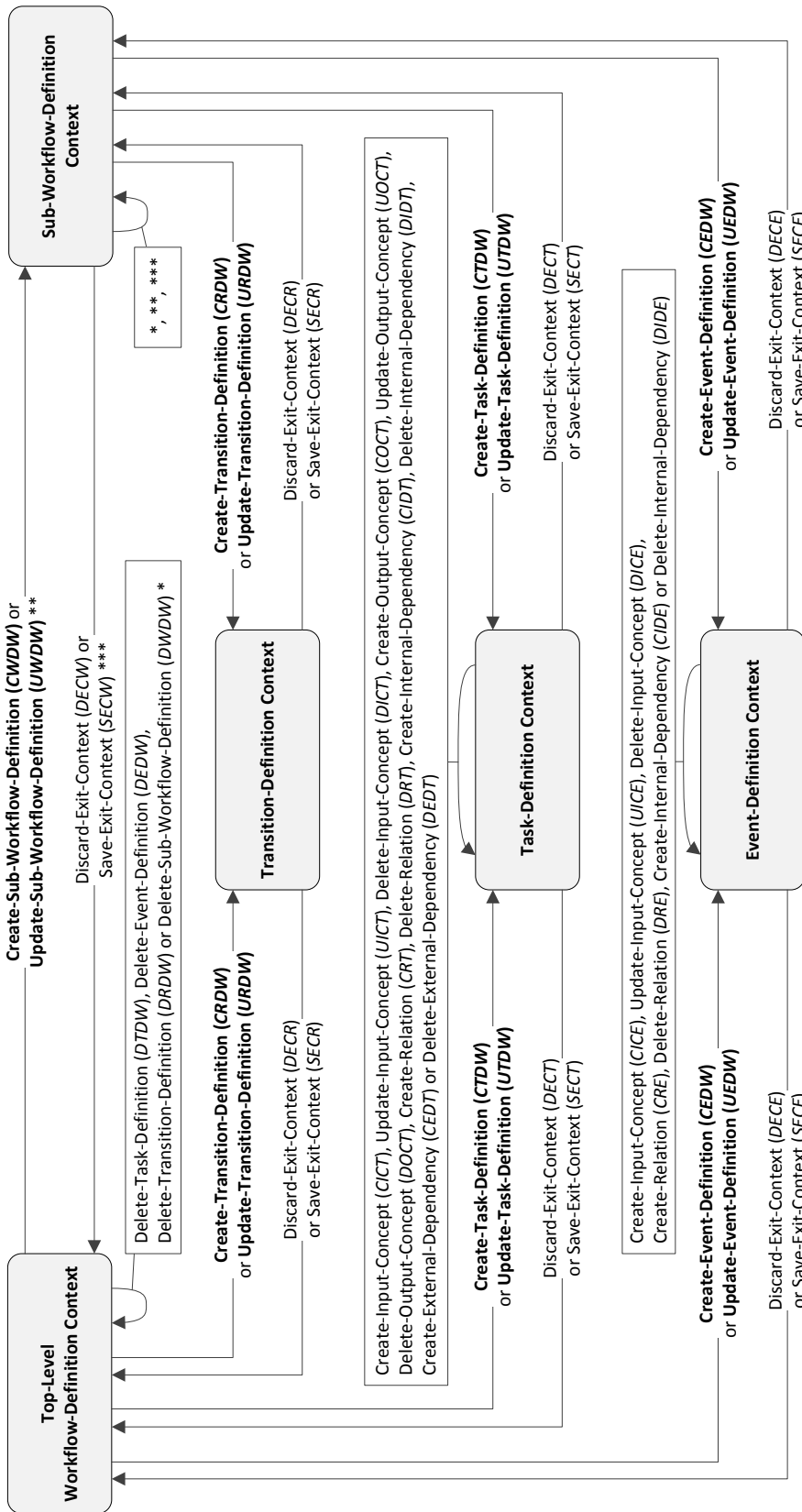


Figure 37: Transitions between contexts in the assisted workflow-definition construction process.

7.2.1 The Task-Definition Context

The construction process enters a task-definition context after the execution of a create or update task-definition command. A task-definition context always contains an associated task-definition $TD(OWF) = (AT_{TD}, T_{TD}, AA_{TD}, IO_{TD}, IOBox_{TD}, AR_{TD}, ARBox_{TD}, \Delta TD_{OD}, \Delta TD_{OCF}, W_{TD}, I_{TD}, p_{TD}, l_{TD}, d_{TD}, au_{TD})$.

7.2.1.1 Create-Input-Concept Commands

A Create-Input-Concept command in a task-definition context is a structure $CICT(TD) := (I, T, RT, RTBox, IBox, DC, WC, SC, p)$, where:

- $I \in DLAC_{OWF}$ is the atomic input concept;
- $T \in \{Unit, UnitContext\}$ is the type of the input concept;
- $RT \in DLAR_{OWF}$ is the atomic role that relates AA_{TD} to I ;
- $RTBox \subseteq TBox_{OWF}$ is the set of role restrictions relating AA_{TD} to I ;
- $IBox \subseteq TBox_{OWF}$ is the set of role restrictions involving I ;
- $DC \subseteq DLAC_{OD}$ is a set of mapped atomic domain concepts;
- $WC \subseteq DLAC_{OWF}$ is a set of external operational concepts in which I depends;
- $SC \subseteq IOI_{TD}$ is a set of internal operational concepts in which I depends;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $CICT$ results in:

- $I \in IOI_{TD}$;
- $RT \in AR_{TD} \cup \{hasUnit, hasUnitContext\}$;
- $RTBox \subseteq ARBox_{TD}$;
- $IBox \subseteq IOBox_{TD}$;
- $\forall D \in DC \Rightarrow (I, D) \in \Delta TD_{OD}$;
- $(I, T) \in \Delta TD_{CFO}$;
- $\forall D \in WC \Rightarrow (I, D) \in \diamond WD_{WF}$;
- $\forall D \in SC \Rightarrow [I \sqsubseteq D]_{TBox_{OWF}}$.

A set of Create-Input-Concept commands, Φ_{CICT} , is suggested through the execution of several algorithms. These algorithms exploit the ΔTD_{OD} relation to suggest the

addition of new concepts and role restrictions to the $IOBox_{TD}$, based on the concepts and role restrictions established in the domain ontology.

[Algorithm 6](#) suggests new input concepts related to other, already existent, input or output concepts in TD . For an already existent concept inside the $IOBox_{TD}$, the role restrictions associated its representation in the domain ontology are followed in order to suggest new domain input concepts.

Algorithm 6: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 1.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all already existent input/output concepts,  $C$ , with the domain type  $X$ 
  // and it. all domain concepts  $Y$ 
   $\forall C, X, Y : C \in (IOI_{TD} \cup IOO_{TD}) \wedge (C, X) \in \Delta TD_{OD} \wedge Y \in DLAC_{OD}$ 
  // it. rels. from  $X$  to  $Y$  and suggest new input concept dependent on  $Y$ 
   $\forall R : (X \xrightarrow{R} Y)_{TBox_{OD}}$ 
  Create new  $(I, T, RT, RTBox, IBox, \{Y\}, \emptyset, \emptyset, 4)$  in  $\Phi_{CICT}$ 
  // include the relationship
  If  $T = Unit$  or  $(C \in IOO_{TD}$  and
     $\exists S : (AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} \wedge \max C(AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} > 1)$  then
    exactC( $C \xrightarrow{R} I$ ) $_{IBox} = 1$ 
  Else
    sameR( $X \xrightarrow{R} Y, C \xrightarrow{R} I$ ) $_{(TBox_{OD}, IBox)}$ 
    sameR( $C \xrightarrow{R} I, AA_{TD} \xrightarrow{RT} I$ ) $_{(IBox, RTBox)}$ 

  // it. rels. from  $Y$  to  $X$  and suggest new input concept dependent on  $Y$ 
   $\forall R : (Y \xrightarrow{R} X)_{TBox_{OD}}$ 
  Create new  $(I, T, RT, RTBox, IBox, \{Y\}, \emptyset, \emptyset, 4)$  in  $\Phi_{CICT}$ 
  // include the relationship
  sameR( $Y \xrightarrow{R} X, I \xrightarrow{R} C$ ) $_{(TBox_{OD}, IBox)}$ 
  Adjust card. of  $I \xrightarrow{R} C$  from  $AA_{TD} \xrightarrow{S} C$  (see Section 5.2.3)
  If  $T = Unit$  or  $(C \in IOO_{TD}$  and
     $\exists S : (AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} \wedge \max C(AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} > 1)$  then
    exactC( $AA_{TD} \xrightarrow{RT} I$ ) $_{RTBox} = 1$ 
  Else
    sameR( $Y \xrightarrow{R} X, AA_{TD} \xrightarrow{RT} I$ ) $_{(TBox_{OD}, RTBox)}$ 

```

[Algorithm 7](#) suggests new input concepts from the set of domain concepts present in the domain ontology that represent a sub-concept of another, already existent, input concept in TD . Although the process is similar to that of [Algorithm 6](#), instead of follow-

ing role restrictions between concepts in the domain ontology, it follows subsumption relations between concepts.

Algorithm 7: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 2.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all already existent input concepts,  $C$ , with the domain type  $X$ 
  // and it. all sub-concepts,  $D$ , of  $X$ 
   $\forall C, X, D : C \in IOI_{TD} \wedge (C, X) \in \Delta TD_{OD} \wedge D \in DLAC_{OD} \wedge [D \sqsubseteq X]_{TBox_{OD}}$ 
  // suggest a new input concept from  $D$  and include the subsumption
  Create new  $(I, T, RT, RTBox, IBox, \{D\}, \emptyset, \{C\}, 3)$  in  $\Phi_{CICT}$ 
  If  $T = Unit$  then  $exactC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 1$ 
  Else  $(AA_{TD} \xrightarrow{RT} I)_{RTBox} \wedge minC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 0$ 

```

Algorithm 8 suggests new input concepts from the set of domain concepts present in the domain ontology, unrelated to other already existent input or output concepts in TD . This algorithm simply suggests new input concepts for each available concept in the domain ontology.

Algorithm 8: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 3.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all domain concepts  $D$  and suggest a new input concept from  $D$ 
   $\forall D \in DLAC_{OD}$ 
  Create new  $(I, T, RT, RTBox, \emptyset, \{D\}, \emptyset, \emptyset, 2)$  in  $\Phi_{CICT}$ 
  If  $T = Unit$  then  $exactC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 1$ 
  Else  $minC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 1$ 

```

Algorithm 9 suggests new input concepts related to other, already existent, input concepts in TD that are dependent on an event or assignment concept. The process is similar to that of Algorithm 6. However, the already existent concept does not contain a representation in the domain ontology. Thus, instead of performing an analysis of the domain ontology, this algorithm follows the concepts and role restrictions established by the depended upon task-definition.

Algorithm 9: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 4.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all already existent input concepts,  $C$ 
   $\forall C : C \in IOI_{TD}$ 

```

```

// it. all task/event-definitions  $X$ , as long as  $C$  is dependent on their
// assignment/event concept
 $\forall X : X \in AD_{WD} \wedge ((C, fTDef_{AA}(X)) \in \diamond WD_{WF} \vee (isEDef(X) \wedge (C, X) \in \diamond WD_{WF}))$ 
// it. and suggest input/output concepts of  $X$  as new input concepts
 $\forall D : D \in (fTDef_{IOI}(X) \cup fTDef_{IOO}(X)) \vee D \in fEDef_{IOI}(X)$ 
  Init.  $DEP = X$ 
  If  $isTDef(X)$  then Set  $DEP = fTDef_{AA}(X)$ 
  Create new  $(I, T, RT, RTBox, IBox, DC, \{D\}, \emptyset, 3)$  in  $\Phi_{CICT}$ 
  sameR( $DEP \xrightarrow{R} D, C \xrightarrow{R} I$ )(TBoxOWF, IBox)
  If  $T = Unit$  then exactC( $AA_{TD} \xrightarrow{RT} I$ )RTBox = 1
  Else sameR( $DEP \xrightarrow{R} D, AA_{TD} \xrightarrow{RT} I$ )(TBoxOWF, RTBox)

```

Algorithm 10 suggests new input concepts dependent on an assignment or event concept and related to other, already existent, input concepts in TD . This is a complement to **Algorithm 9**. For each of the already existent task/event-definition in the workflow-definition (excluding the current one), the algorithm suggests input concepts dependent on their operational concepts that do not possess a representation in the domain ontology (i.e. event and assignment concepts). The suggestion is only made if the new dependent input concept is also related to an already existent input concept in the $IOBox_{TD}$.

Algorithm 10: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 5.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
// it. all already existent input concepts,  $C$ , dependent on  $D$ 
 $\forall C, D : C \in IOI_{TD} \wedge (C, D) \in \diamond WD_{WF}$ 
// get the task/event-definition  $X$  containing the input/output concept  $D$ 
 $\forall X \in AD_{WD} \wedge (D \in (fTDef_{IOI}(X) \cup fTDef_{IOO}(X)) \vee D \in fEDef_{IOI}(X))$ 
// suggest new input concept dependent on the event concept  $X$ ...
  Init.  $DEP = X$ 
  // ...or in the assignment concept of  $X$ 
  If  $isTDef(X)$  then Set  $DEP = fTDef_{AA}(X)$ 
  Create new  $(I, T, RT, RTBox, IBox, \emptyset, \{DEP\}, \emptyset, 3)$  in  $\Phi_{CICT}$ 
  sameR( $DEP \xrightarrow{R} D, I \xrightarrow{R} C$ )(TBoxOWF, IBox)
  If  $T = Unit$  then exactC( $AA_{TD} \xrightarrow{RT} I$ )RTBox = 1
  Else minC( $AA_{TD} \xrightarrow{RT} I$ )RTBox = 1

```

Algorithm 11 suggests new input concepts dependent on an assignment or event concept, unrelated to other already existent input or output concepts in TD .

Algorithm 11: Suggestion of Create-Input-Concept commands in a task-definition context: subset no. 6.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all task/event-definitions,  $X$  (excluding the current)
   $\forall X : X \in AD_{WD} \wedge X \neq AT_{TD} \wedge (isTDef(X) \vee isEDef(X))$ 
  // suggest new input concept dependent on the event concept  $X...$ 
  Init.  $DEP = X$ 
  // ...or in the assignment concept of  $X$ 
  If  $isTDef(X)$  then Set  $DEP = fTDef_{AA}(X)$ 
  Create new  $(I, T, RT, RTBox, \emptyset, \emptyset, \{DEP\}, \emptyset, 2)$  in  $\Phi_{CICT}$ 
  If  $T = Unit$  then  $exactC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 1$ 
  Else  $minC(AA_{TD} \xrightarrow{RT} I)_{RTBox} = 1$ 

```

7.2.1.2 Create-Output-Concept Commands

A Create-Output-Concept command in a task-definition context is a structure $COCT(TD) := (O, T, RT, RTBox, OBox, DC, SC, p)$, where:

- $O \in DLAC_{OWF}$ is the atomic output concept;
- $T \in \{Response, ResponseContext\}$ is the type of the output concept;
- $RT \in DLAR_{OWF}$ is the atomic role that relates AA_{TD} to O ;
- $RTBox \subseteq TBox_{OWF}$ is the set of role restrictions relating AA_{TD} to O ;
- $OBox \subseteq TBox_{OWF}$ is the set of role restrictions involving O ;
- $DC \subseteq DLAC_{OD}$ is a set of mapped atomic domain concepts;
- $SC \subseteq IOI_{TD} \cup IOO_{TD}$ is a set of internal operational concepts in which O depends;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $COCT$ results in:

- $O \in IOO_{TD}$;
- $RT \in AR_{TD} \cup \{hasResponse, hasResponseContext\}$;
- $RTBox \subseteq ARBox_{TD}$;
- $OBox \subseteq IOBox_{TD}$;
- $\forall D \in DC \Rightarrow (O, D) \in \Delta TD_{OD}$;
- $(O, T) \in \Delta TD_{CFO}$;

- $\forall D \in SC \Rightarrow [O \sqsubseteq D]_{TBox_{OWF}}$.

A set of Create-Output-Concept commands, Φ_{COCT} , is suggested through the execution of several algorithms. These algorithms exploit the ΔTD_{OD} relation to suggest the addition of new concepts and role restrictions to the $IOBox_{TD}$, based on the concepts and role restrictions established in the domain ontology.

[Algorithm 12](#) suggests new output concepts related to other, already existent, input or output concepts in TD . The process is similar to the one outlined in [Algorithm 6](#).

Algorithm 12: Suggestion of Create-Output-Concept commands in a task-definition context:
sub-set no. 1.

```

Do for  $T = Response$  and  $T = ResponseContext$ 
// it. all already existent input/output concepts,  $C$ , with the domain type  $X$ 
// and it. all domain concepts  $Y$ 
 $\forall C, X, Y : C \in (IOI_{TD} \cup IOO_{TD}) \wedge (C, X) \in \Delta TD_{OD} \wedge Y \in DLAC_{OD}$ 
// it. rels. from  $X$  to  $Y$  and suggest new output concept dependent on  $Y$ 
 $\forall R : (X \xrightarrow{R} Y)_{TBox_{OD}}$ 
  Create new  $(O, T, RT, RTBox, OBox, \{Y\}, \emptyset, 4)$  in  $\Phi_{COCT}$ 
  // include the relationship
  If  $C \in IOI_{TD} \wedge \exists S : (AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} \wedge \max C(AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} > 1$  then
    exactC( $C \xrightarrow{R} O$ ) $_{OBox} = 1$ 
  Else
    sameR( $X \xrightarrow{R} Y, C \xrightarrow{R} O$ ) $_{(TBox_{OD}, OBox)}$ 
    sameR( $C \xrightarrow{R} O, AA_{TD} \xrightarrow{RT} I$ ) $_{(OBox, RTBox)}$ 

// it. rels. from  $Y$  to  $X$  and suggest new output concept dependent on  $Y$ 
 $\forall R : (Y \xrightarrow{R} X)_{TBox_{OD}}$ 
  Create new  $(O, T, RT, RTBox, OBox, \{Y\}, \emptyset, 4)$  in  $\Phi_{COCT}$ 
  // include the relationship
  sameR( $Y \xrightarrow{R} X, O \xrightarrow{R} C$ ) $_{(TBox_{OD}, OBox)}$ 
  Adjust card. of  $O \xrightarrow{R} C$  from  $AA_{TD} \xrightarrow{S} C$  (see Section 5.2.3)
  If  $C \in IOI_{TD} \wedge \exists S : (AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} \wedge \max C(AA_{TD} \xrightarrow{S} C)_{ARBox_{TD}} > 1$  then
    exactC( $AA_{TD} \xrightarrow{RT} O$ ) $_{RTBox} = 1$ 
  Else
    sameR( $Y \xrightarrow{R} X, AA_{TD} \xrightarrow{RT} O$ ) $_{(TBox_{OD}, RTBox)}$ 

```

[Algorithm 13](#) suggests new output concepts from the set of domain concepts present in the domain ontology that represent a sub-concept of another, already existent, input or output concept in TD . The process is similar to the one outlined in [Algorithm 7](#).

Algorithm 13: Suggestion of Create-Output-Concept commands in a task-definition context:
sub-set no. 2.

```

Do for  $T = \text{Response}$  and  $T = \text{ResponseContext}$ 
  // it. all already existent input/output concepts,  $C$ , with the domain type  $X$ 
  // and it. all sub-concepts,  $D$ , of  $X$ 
   $\forall C, X, D : C \in (IOI_{TD} \cup IOO_{TD}) \wedge (C, X) \in \Delta TD_{OD} \wedge D \in DLAC_{OD}$ 
  If  $[D \sqsubseteq X]_{TBox_{OD}}$  then
    // suggest a new output concept from  $D$  and include the subsumption
    Create new  $(O, T, RT, RTBox, OBox, \{D\}, \{C\}, 3)$  in  $\Phi_{COCT}$ 
     $(AA_{TD} \xrightarrow{RT} O)_{RTBox} \wedge \text{minC}(AA_{TD} \xrightarrow{RT} O)_{RTBox} = 0$ 
     $\text{exactC}(AA_{TD} \xrightarrow{RT} O)_{RTBox} = 0$ 

```

Algorithm 14 suggests new output concepts from the set of domain concepts present in the domain ontology, unrelated to other already existent input or output concepts in TD . The process is similar to the one outlined in Algorithm 8.

Algorithm 14: Suggestion of Create-Output-Concept commands in a task-definition context:
sub-set no. 3.

```

Do for  $T = \text{Response}$  and  $T = \text{ResponseContext}$ 
  // it. all domain concepts  $D$  and suggest a new output concept from  $D$ 
   $\forall D \in DLAC_{OD}$ 
  Create new  $(O, T, RT, RTBox, \emptyset, \{D\}, \emptyset, 2)$  in  $\Phi_{COCT}$ 
   $\text{minC}(AA_{TD} \xrightarrow{RT} O)_{RTBox} = 1$ 

```

7.2.1.3 Create-Relation Commands

A Create-Relation command in a task-definition context is a structure $CRT(TD) := (RRBox, p)$, where:

- $RRBox \subseteq TBox_{OWF}$ is the set of role restrictions between two input or output concepts;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a CRT results in $RRBox \subseteq IOBox_{TD}$.

A set of Create-Relation commands, Φ_{CRT} , is suggested through the execution of Algorithm 15 and Algorithm 16. These algorithms exploit the ΔTD_{OD} relation to suggest the addition of new role restrictions to the $IOBox_{TD}$, based on the role restrictions established in the domain ontology.

Algorithm 15 suggests role restrictions between two already existent input concepts, or between two already existent output concepts.

Algorithm 15: Suggestion of Create-Relation commands in a task-definition context: sub-set no.

1.

```
// it. all pairs of input/output concepts, C and D, with domain types X and Y
∀C,D,X,Y : ((C ∈ IOITD ∧ D ∈ IOITD) ∨ (C ∈ IOOTD ∧ D ∈ IOOTD))
  If (C,X) ∈ ΔTDOD ∧ (D,Y) ∈ ΔTDOD then
    // it. rels., R, from X to Y, in the domain ontology
    ∀R : (X  $\xrightarrow{R}$  Y)TBoxOD
      // suggest new role restrictions from R
      Create new (RRBox,5) in ΦCRT
      sameR(X  $\xrightarrow{R}$  Y, C  $\xrightarrow{R}$  D)(TBoxOD,RRBox)
      Adjust card. of C  $\xrightarrow{R}$  D from AATD  $\xrightarrow{S}$  D (see Section 5.2.3)
```

Algorithm 16 suggests role restrictions between an input concept and an output concept, and vice-versa.

Algorithm 16: Suggestion of Create-Relation commands in a task-definition context: sub-set no.

2.

```
// it. all output concepts, C, and all input concepts, D...
∀C,D,X,Y : ((C ∈ IOOTD ∧ D ∈ IOITD) ∨ (C ∈ IOITD ∧ D ∈ IOOTD))
  // ...with domain types X and Y
  If (C,X) ∈ ΔTDOD ∧ (D,Y) ∈ ΔTDOD then
    // ...where either C or D has a maximum assignment cardinality of 1
    If ∀S : maxC(AATD  $\xrightarrow{S}$  C)ARBoxTD ≤ 1 or ∀S : maxC(AATD  $\xrightarrow{S}$  D)ARBoxTD ≤ 1 then
      // it. rels., R, from X to Y, in the domain ontology
      ∀R : (X  $\xrightarrow{R}$  Y)TBoxOD
        // suggest new role restrictions from R
        Create new (RRBox,5) in ΦCRT
        sameR(X  $\xrightarrow{R}$  Y, C  $\xrightarrow{R}$  D)(TBoxOD,RRBox)
        Adjust card. of C  $\xrightarrow{R}$  D from AATD  $\xrightarrow{S}$  D (see Section 5.2.3)
```

7.2.1.4 Create-Internal-Dependency Commands

A Create-Internal-Dependency command establishes a dependency between input or output concepts inside the task-definition.

A Create-Internal-Dependency command in a task-definition context is a structure $CIDT(TD) := (OC, DC, p)$, where:

- $OC \in (IOI_{TD} \cup IOO_{TD})$ is the atomic origin (dependent) concept;
- $DC \in (IOI_{TD} \cup IOO_{TD})$ is the atomic destination (depended upon) concept;
- $p \in [1,5]$ is the priority value of the command.

The execution of a $CIDT$ results in $[OC \sqsubseteq DC]_{IOBox_{TD}}$.

A set of Create-Internal-Dependency commands, Φ_{CIDT} , is suggested through the execution of [Algorithm 17](#) and [Algorithm 18](#).

[Algorithm 17](#) suggests internal dependencies between two input concepts and between two output concepts. The algorithm exploits the ΔTD_{OD} relation to suggest the addition of new internal dependencies, based on the subsumptions found in the domain ontology.

Algorithm 17: Suggestion of Create-Internal-Dependency commands in a task-definition context: sub-set no. 1.

```
// it. all pairs of input/output concepts, C and D...
∀C,D : ((C ∈ IOOTD ∧ D ∈ IOOTD) ∨ (C ∈ IOITD ∧ D ∈ IOITD))
// ...with domain types X and Y (where X subsumes Y)
If ∀X,Y : (C,X) ∈ ΔTDOD ∧ (D,Y) ∈ ΔTDOD ∧ [X ⊆ Y]TBoxOD then
/* ...where the interval of values denoting the possible amount of C per
assignment is within the interval of values denoting the possible amount
of D per assignment */
If card. of ∀R : AATD  $\xrightarrow{R}$  C are contained in card. ∀S : AATD  $\xrightarrow{S}$  D then
Create new (C,D,2) in ΦCIDT
```

[Algorithm 18](#) suggests internal dependencies between an output concept and an input concept. In general, these suggestions are only made if the concepts have the same domain type in ΔTD_{OD} .

Algorithm 18: Suggestion of Create-Internal-Dependency commands in a task-definition context: sub-set no. 2.

```
// it. all output concepts, C, and all input concepts, D...
∀C,D : C ∈ IOOTD ∧ D ∈ IOITD
// ...with the same domain type
If ∀X,Y : (C,X) ∈ ΔTDOD ∧ (D,Y) ∈ ΔTDOD ∧ X = Y then
/* ...where the interval of values denoting the possible amount of C per
assignment is within the interval of values denoting the possible amount
of D per assignment */
If card. of ∀R : AATD  $\xrightarrow{R}$  C are contained in card. ∀S : AATD  $\xrightarrow{S}$  D then
Create new (C,D,3) in ΦCIDT
```

7.2.1.5 Create-External-Dependency Commands

A Create-External-Dependency command establishes a dependency between an input concept inside the task-definition and an operational concept of another task-definition or event-definition.

A Create-External-Dependency command in a task-definition context is a structure $CEDT(TD) := (OC, DC, p)$, where:

- $OC \in (IOI_{TD} \cup IOO_{TD})$ is the atomic origin (dependent) input concept in TD ;
- $DC \in DLAC_{OWF}$ is the atomic destination (depended upon) operational concept;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $CEDT$ results in $(OC, DC) \in \diamond WD_{WF}$.

A set of Create-External-Dependency commands, Φ_{CEDT} , is suggested through the execution of [Algorithm 19](#), which suggests external dependencies for concepts with the same Δ_{OD} .

Algorithm 19: Suggestion of Create-External-Dependency commands in a task-definition context.

```
// it. all input concepts, C, of the current task-definition and
// it. all operational concepts, D, of all other task/event-definitions, X...
∀C,D,X : C ∈ IOITD ∧ X ∈ ADWD ∧ (D ∈ fTDefOP(X) ∨ D ∈ fEDefOP(X))
  If X ≠ ATTD then
    // ...with the same domain type
    If {E|(C,E) ∈ ΔTDOD} = {E|(D,E) ∈ fTDefΔOD(X) ∨ (D,E) ∈ fEDefΔOD(X)} then
      Create new (C,D,2) in ΦCEDT
```

7.2.2 The Event-Definition Context

The construction process enters an event-definition context after a create or update event-definition command. An event-definition context always contains an associated event-definition $ED(OWF) = (AE_{ED}, T_{ED}, IO_{ED}, IOBox_{ED}, AR_{ED}, ARBox_{ED}, \Delta ED_{OD}, \Delta ED_{OCF}, l_{ED}, p_{ED}, l_{ED}, d_{ED})$.

7.2.2.1 Create-Input-Concept Commands

An Create-Input-Concept command in an event-definition context is a structure $CICE(ED) := (I, T, RT, RTBox, IBox, DC, SC, p)$, where:

- $I \in DLAC_{OWF}$ is the atomic input concept;
- $T \in \{Unit, UnitContext\}$ is the type of the input concept;
- $RT \in DLAR_{OWF}$ is the atomic role that relates AE_{ED} to I ;
- $RTBox \subseteq TBox_{OWF}$ is the set of role restrictions relating AE_{ED} to I ;
- $IBox \subseteq TBox_{OWF}$ is the set of role restrictions involving I ;
- $DC \subseteq DLAC_{OD}$ is a set of mapped atomic domain concepts;
- $SC \subseteq IOI_{ED}$ is a set of internal operational concepts in which I depends;
- $p \in [1,5]$ is the priority value of the command.

The execution of a $CICE$ results in:

- $I \in IOI_{ED}$;
- $RT \in AR_{ED} \cup \{hasUnit, hasUnitContext\}$;
- $RTBox \subseteq ARBox_{ED}$;
- $IBox \subseteq IOBox_{ED}$;
- $\forall D \in DC \Rightarrow (I, D) \in \Delta ED_{OD}$;
- $(I, T) \in \Delta ED_{CFO}$;
- $\forall D \in SC \Rightarrow [I \sqsubseteq D]_{TBox_{OWF}}$.

A set of Create-Input-Concept commands, Φ_{CICE} , is suggested through the execution of several algorithms. These algorithms are an adaptation of those found in the task-definition context. However, since event-definitions do not represent operations on top of data, a distinction between input and output concepts is not made.

[Algorithm 20](#) suggests new data concepts related to other, already existent, data concepts in ED .

Algorithm 20: Suggestion of Create-Input-Concept commands in an event-definition context:
sub-set no. 1.

```

Do for  $T = Unit$  and  $T = UnitContext$ 
  // it. all already existent data concepts,  $C$ , with the domain type  $X$ 
  // and it. all domain concepts  $Y$ 

```

```

 $\forall C, X, Y : C \in IOI_{ED} \wedge (C, X) \in \Delta ED_{OD} \wedge Y \in DLAC_{OD}$ 
  // it. rels. from X to Y and suggest new input concept dependent on Y
   $\forall R : (X \xrightarrow{R} Y)_{TBox_{OD}}$ 
    Create new (I, T, RT, RTBox, IBox, {Y},  $\emptyset$ ,  $\emptyset$ , 4) in  $\Phi_{CICE}$ 
    // include the relationship
    If T = Unit then exactC( $C \xrightarrow{R} I$ )IBox = 1
    Else sameR( $X \xrightarrow{R} Y, C \xrightarrow{R} I$ )(TBoxOD, IBox)
      sameR( $C \xrightarrow{R} I, AE_{ED} \xrightarrow{RT} I$ )(IBox, RTBox)

  // it. rels. from Y to X and suggest new input concept dependent on Y
   $\forall R : (Y \xrightarrow{R} X)_{TBox_{OD}}$ 
    Create new (I, T, RT, RTBox, IBox, {Y},  $\emptyset$ ,  $\emptyset$ , 4) in  $\Phi_{CICE}$ 
    // include the relationship
    sameR( $Y \xrightarrow{R} X, I \xrightarrow{R} C$ )(TBoxOD, IBox)
    Adjust card. of  $I \xrightarrow{R} C$  from  $AE_{ED} \xrightarrow{S} C$  (see Section 5.2.3)
    If T = Unit then exactC( $AE_{ED} \xrightarrow{RT} I$ )RTBox = 1
    Else sameR( $Y \xrightarrow{R} X, AE_{ED} \xrightarrow{RT} I$ )(TBoxOD, RTBox)

```

Algorithm 21 suggests new data concepts from the set of domain concepts present in the domain ontology that represent a sub-concept of another, already existent, data concept in *ED*.

Algorithm 21: Suggestion of Create-Input-Concept commands in an event-definition context: sub-set no. 2.

```

Do for T = Unit and T = UnitContext
  // it. all already existent data concepts, C, with the domain type X
  // and it. all sub-concepts, D, of X
   $\forall C, X, D : C \in IOI_{ED} \wedge (C, X) \in \Delta ED_{OD} \wedge D \in DLAC_{OD} \wedge [D \sqsubseteq X]_{TBox_{OD}}$ 
    // suggest a new input concept from D and include the subsumption
    Create new (I, T, RT, RTBox, IBox, {D},  $\emptyset$ , {C}, 3) in  $\Phi_{CICE}$ 
    If T = Unit then exactC( $AE_{ED} \xrightarrow{RT} I$ )RTBox = 1
    Else ( $AE_{ED} \xrightarrow{RT} I$ )RTBox  $\wedge$  minC( $AE_{ED} \xrightarrow{RT} I$ )RTBox = 0

```

Algorithm 22 suggests new data concepts from the set of domain concepts present in the domain ontology, unrelated to other already data concepts in *ED*.

Algorithm 22: Suggestion of Create-Input-Concept commands in an event-definition context: sub-set no. 3.

```

Do for T = Unit and T = UnitContext
  // it. all domain concepts D and suggest a new input concept from D
   $\forall D \in DLAC_{OD}$ 

```

```

Create new  $(I, T, RT, RTBox, \emptyset, \{D\}, \emptyset, \emptyset, 2)$  in  $\Phi_{CICE}$ 
If  $T = \text{Unit}$  then  $\text{exactC}(AE_{ED} \xrightarrow{RT} I)_{RTBox} = 1$ 
Else  $\text{minC}(AE_{ED} \xrightarrow{RT} I)_{RTBox} = 1$ 

```

7.2.2.2 Create-Relation Commands

A Create-Relation command in an event-definition context is a structure $CRE(ED) := (RRBox, p)$, where:

- $RRBox \subseteq TBox_{OWF}$ is the set of role restrictions between two data concepts;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a CRE results in $RRBox \subseteq IOBox_{ED}$.

A set of Create-Relation commands, Φ_{CRE} , is suggested through the execution of [Algorithm 23](#), which suggests role restrictions between two data concepts.

Algorithm 23: Suggestion of Create-Relation commands in an event-definition context.

```

// it. all pairs of data concepts, C and D, with domain types X and Y
 $\forall C, D, X, Y : C \in IOI_{ED} \wedge D \in IOI_{ED} \wedge (C, X) \in \Delta ED_{OD} \wedge (D, Y) \in \Delta ED_{OD}$ 
// it. rels., R, from X to Y, in the domain ontology
 $\forall R : (X \xrightarrow{R} Y)_{TBox_{OD}}$ 
// suggest new role restrictions from R
Create new  $(RRBox, 5)$  in  $\Phi_{CRE}$ 
 $\text{sameR}(X \xrightarrow{R} Y, C \xrightarrow{R} D)_{(TBox_{OD}, RRBox)}$ 
Adjust card. of  $C \xrightarrow{R} D$  from  $AE_{ED} \xrightarrow{S} D$  (see Section 5.2.3)

```

7.2.2.3 Create-Internal-Dependency Commands

A Create-Internal-Dependency command in an event-definition context is a structure $CIDE(ED) := (OC, DC, p)$, where:

- $OC \in (IOI_{TD} \cup IOO_{TD})$ is the atomic origin (dependent) concept;
- $DC \in (IOI_{TD} \cup IOO_{TD})$ is the atomic destination (depended upon) concept;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $CIDE$ results in $[OC \sqsubseteq DC]_{IOBox_{ED}}$.

A set of Create-Internal-Dependency commands, Φ_{CIDE} , is suggested through the execution of [Algorithm 24](#), which suggests internal dependencies between two data concepts.

Algorithm 24: Suggestion of Create-Internal-Dependency commands in an event-definition context.

```

// it. all pairs of data concepts, C and D...
∀C,D : C ∈ IOIED ∧ D ∈ IOIED
// ...with domain types X and Y (where X subsumes Y)
If ∀X,Y : (C,X) ∈ ΔEDOD ∧ (D,Y) ∈ ΔEDOD ∧ [X ⊆ Y]TBoxOD then
/* ...where the interval of values denoting the possible amount of C per
assignment is within the interval of values denoting the possible amount
of D per assignment */
If card. of ∀R : AEED  $\xrightarrow{R}$  C are contained in card. ∀S : AEED  $\xrightarrow{S}$  D then
Create new (C,D,2) in ΦCIDE

```

7.2.3 The Transition-Definition Context

The construction process enters a transition-definition context after a create or update transition-definition command. A transition-definition context always contains an associated transition-definition $RD(OWF) = (AR_{RD}, T_{RD}, CI_{RD}, CO_{RD}, Cond_{RD})$.

Operations on top of transition-definitions are, currently, either manual or fully automated through some of the commands in the workflow-definition context. A transition-definition may be automatically established between two activity-definitions if they are dependent.

7.2.4 The Workflow-Definition Context

The construction process enters a workflow-definition context after the creation or update of a top-level workflow-definition, or after a create or update sub-workflow-definition command. A workflow-definition context always contains an associated workflow-definition $WD(OWF) = (AW_{WD}, T_{WD}, AD_{WD}, TRD_{WD}, IED_{WD}, \diamond_{WF}, Cond_{WD}, p_{WD}, l_{WD}, d_{WD})$.

7.2.4.1 Create-Task-Definition Commands

A Create-Task-Definition command in a workflow-definition context is a structure $CTDW(WD) := (AT, AA, W, I, tp, l, d, au, p)$, where:

- $AT \in DLAC_{OWF}$ is the new task-definition atomic concept (name derived from AA);
- $AA \in DLAC_{OWF}$ is the new task-definition atomic assignment concept;
- $W \in DLAC_{OWF}$ is the new task-definition atomic worker concept;
- $I \in DLAC_{OWF}$ is the new task-definition atomic task interface concept;
- $tp \in PD_{OCF}$ is the new task-definition priority (defaults to *medium*);
- $l \in DLV_{OWF}$ is the new task-definition label;
- $d \in DLV_{OWF}$ is the new task-definition description;
- $au \in DLV_{OWF}$ is the new task-definition assignments per unit value (defaults to 1);
- $p \in [1,5]$ is the priority value of the command.

The execution of a $CTDW$ results in a new task-definition $TDef = (AT, T_{TDef}, AA, IO_{TDef}, IOBox_{TDef}, AR_{TDef}, ARBox_{TDef}, \Delta_{OD}, \Delta_{OCF}, W, I, tp, l, d, au)$ in OWF . With the creation of a new task-definition, the context of the construction process changes to the new task-definition context.

A Create-Task-Definition command is always suggested in each step of the construction process in the workflow-definition context (with $p = 4$). The creator must manually supply AA, W, I, l and d .

7.2.4.2 Create-Event-Definition Commands

A Create-Event-Definition command in a workflow-definition context is a structure $CEDW(WD) := (AE, T, I, ep, l, d, p)$, where:

- $AE \in DLAC_{OWF}$ is the new event-definition atomic concept;
- $T \in EDT_{OCF}$ is the new event-definition type (defaults to *RunningEvent*);
- $I \in DLAC_{OWF}$ is the new event-definition atomic event interface concept;
- $ep \in PD_{OCF}$ is the new event-definition priority (defaults to *medium*);
- $l \in DLV_{OWF}$ is the new event-definition label;
- $d \in DLV_{OWF}$ is the new event-definition description;
- $p \in [1,5]$ is the priority value of the command.

The execution of a $CEDW$ results in a new event-definition $EDef = (AE, T, IO_{EDef}, IOBox_{EDef}, AR_{EDef}, ARBox_{EDef}, \Delta_{OD}, \Delta_{OCF}, I, ep, l, d)$ in OWF , where $AE \in IED_{WD}$ if

$T = \text{InstantiationEvent}$. Consequently, the construction process context changes to the new event-definition context.

A Create-Event-Definition command is always suggested in each step of the construction process in the workflow-definition context (with $p = 4$). The creator must manually supply AE, I, l and d .

7.2.4.3 Create-Transition-Definition Commands

A Create-Transition-Definition command in a workflow-definition context is a structure $CRDW(WD) := (AR, CI, CO, p)$, where:

- $AR \in DLAC_{OWF}$ is the new transition-definition atomic concept;
- $CI \subseteq DLC_{OWF}$ is the new transition-definition set of incoming activity-definition concepts;
- $CO \subseteq DLC_{OWF}$ is the new transition-definition set of outgoing activity-definition concepts;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $CRDW$ results in a new transition-definition $RDef = (AR, T_{RDef}, CI, CO, Cond_{RDef})$ in OWF , where $AR \in TRD_{WD}$. Consequently, the construction process context changes to the new transition-definition context.

Create-Transition-Definition commands are suggested for each activity-definition in OWF and not in AD_{WD} , i. e. for each activity-definition that is not linked to the workflow-definition through a transition-definition or as a start activity-definition. The set of suggested Create-Transition-Definition commands, Φ_{CRDW} , is built through the execution of [Algorithm 25](#) and [Algorithm 26](#). The creator must manually supply AR . The type, T_{RDef} , is inferred (see [Section 5.4](#)).

[Algorithm 25](#) suggests transition-definitions for all existing activity-definitions that are not yet linked to a workflow-definition.

Algorithm 25: Suggestion of Create-Transition-Definition commands in a workflow-definition context: sub-set 1.

```
// it. all activity-definitions C and D...
∀C, D : C ∈ ADWD ∧ D ∈ DLACOWF ∧ D ∉ ADWD
// ...where D is not linked to the wf-def. and is not an InstantiationEvent
If [D ⊆ Activity]TBoxOWF ∧ [D ⊈ InstantiationEvent]TBoxOWF then
  Create new (AR, {C}, {D}, 3) in ΦCRDW // suggest transition-def. from C to D
```

Algorithm 26 suggests transition-definitions according to the operational dependencies in \diamond_{WF} , i. e. between dependent activity-definitions.

Algorithm 26: Suggestion of Create-Transition-Definition commands in a workflow-definition context: sub-set 2.

```
// it. all distinct and dependent activity-definitions C and D
// the transitive reduction ensures that all dependencies are satisfied
 $\forall C, D : (C, D) \in \prec^- \wedge C \neq D \wedge C \notin IED_{WD}$ 
// suggest transition-definition from C to D, since D depends on C
Create new (AR, {C}, {D}, 3) in  $\Phi_{CRDW}$ 
```

7.2.4.4 Create-Sub-Workflow-Definition Commands

A Create-Sub-Workflow-Definition command in a workflow-definition context is a structure $CWDW(WD) := (AW, ec, wp, l, d, p)$, where:

- $AW \in DLAC_{OWF}$ is the new sub-workflow-definition atomic concept;
- $ec \in DLn_{OWF}$ is the new sub-workflow-definition loop exit condition (optional);
- $wp \in PD_{OCF}$ is the new sub-workflow-definition priority (defaults to *medium*);
- $l \in DLv_{OWF}$ is the new sub-workflow-definition label;
- $d \in DLv_{OWF}$ is the new sub-workflow-definition description;
- $p \in [1, 5]$ is the priority value of the command.

The execution of a $CWDW$ results in a new sub-workflow-definition $WDef = (AW, T_{WDef}, AD_{WDef}, TRD_{WDef}, IED_{WDef}, \diamond_{WF}, Cond, wp, l, d)$ in OWF . Consequently, the construction process context changes to the new sub-workflow-definition context.

A Create-Sub-Workflow-Definition command is always suggested in each step of the construction process in the workflow-definition context (with $p = 4$). The creator must manually supply AW , l and d . Additionally, the creator may choose to supply ec and p .

7.3 PATTERN COMMANDS

Pattern commands are sets of atomic commands triggered by the existence of a particular pattern in the domain and workflow-definition ontologies. As with atomic com-

mands, pattern commands are suggested in the context of an activity-definition or transition-definition.

Currently, three pattern commands are proposed in the workflow-definition context: the *Follow-Role-CreateAndFill*, the *Partition*, and the *Assembler* pattern commands.

7.3.1 *Follow-Role-CreateAndFill* Pattern Commands

The iterative acceptance of *Follow-Role-CreateAndFill* pattern commands allows the creator to build a workflow-definition that incrementally instantiates the domain ontology.

The *Follow-Role-CreateAndFill* pattern command exists in the workflow-definition context. It creates a new task-definition by following the role restrictions from or to a domain concept in the output of another task-definition.

A situation where this pattern command would be useful is found in the document partition and translation workflow-definition presented in [Figure 24](#). For instance, after the construction of T_1 , and according to the domain ontology, a *Follow-Role-CreateAndFill* command is suggested. This command builds a task-definition that follows the *contains* role restriction onto *Paragraph*, and further partitions the paragraphs (those that belong to the output of T_1) into sentences.

A *Follow-Role-CreateAndFill* pattern command in a workflow-definition context is a structure $FRCF(WD) := (OA, CD, SBox, DC, AT, AA, W, TI, l, d, au, I, O, p)$, where:

- $OA \in DLAC_{OWF}$ is the dependent upon activity-definition atomic concept;
- $CD \in DLAC_{OWF}$ is an input or output concept in OA ;
- $SBox \subseteq TBox_{OD}$ is a set of role restrictions that associate the domain concepts of CD with the domain concept DC ;
- $DC \subseteq DLAC_{OD}$ is the domain atomic concept, linked to the response concept of the new task-definition;
- $AT \in DLAC_{OWF}$ is the new task-definition atomic concept (name derived from AA);
- $AA \in DLAC_{OWF}$ is the new task-definition atomic assignment concept;
- $W \in DLAC_{OWF}$ is the new task-definition atomic worker concept;
- $TI \in DLAC_{OWF}$ is the new task-definition atomic task interface concept;

- $tp \in PD_{OCF}$ is the new task-definition priority (defaults to *medium*);
- $l \in DLv_{OWF}$ is the new task-definition label;
- $d \in DLv_{OWF}$ is the new task-definition description;
- $au \in DLv_{OWF}$ is the new task-definition assignments per unit value (defaults to 1);
- $I \in DLAC_{OWF}$ is the new task-definition unit concept dependent on CD ;
- $O \in DLAC_{OWF}$ is the new task-definition response concept linked to DC ;
- $p \in [1,5]$ is the priority value of the pattern command.

The structure $FRCF$ is a valid Follow-Role-CreateAndFill pattern command if it satisfies the following conditions:

- $[OA \sqsubseteq Activity]_{TBox_{OWF}}$;
- $[OA \not\sqsubseteq InstantiationEvent]_{TBox_{OWF}}$;
- $|SBox| \geq 1$;
- Regarding CD :
 - $CD \in (fTDef_{IOI}(OA) \cup fTDef_{IOO}(OA))$ if $isTDef(OA)$;
 - $CD \in fEDef_{IOI}(OA)$ if $isEDef(OA)$;
- $\forall C, D, R : (C \xrightarrow{R} D)_{SBox} \Rightarrow \exists X : ((C = DC \wedge (CD, D) \in X) \vee (D = DC \wedge (CD, C) \in X)) \wedge ((isTDef(OA) \wedge X = fTDef_{\Delta OD}(OA)) \vee (isEDef(OA) \wedge X = fEDef_{\Delta OD}(OA)))$.

7.3.1.1 Execution

As presented in [Algorithm 27](#), the execution of a $FRCF$ results in the execution of a sequence of atomic commands.

Algorithm 27: Execution of a Follow-Role-CreateAndFill pattern command in a workflow-definition context.

```

// create a new (partial) task-def. and move onto the task-def. context
Init. Create-Task-Definition  $\delta_{CTDW} = (AT, AA, W, TI, tp, l, d, au, p)$ 
Exec.  $\delta_{CTDW}$ 

// get the domain type of the input/output concept,  $CD$ ,
// in the task/event-definition,  $OA$ 
Init.  $IDC = \emptyset$ 

```

```

If isTDef(OA) then IDC = {Y | (CD, Y) ∈ fTDefΔOD(OA)}
If isEDef(OA) then IDC = {Y | (CD, Y) ∈ fEDefΔOD(OA)}
// create the new unit concept dependent on CD, with the same domain type
Init. Create-Input-Concept  $\delta_{CICT} = (I, \{Unit\}, \emptyset, RTBox, \emptyset, IDC, \{CD\}, \emptyset, p)$ 
Exec.  $\delta_{CICT}$ 

Init. S = hasResponse
// create OBox, with the role restrictions that will be added to the IOBox
Init. OBox such that  $\forall C, R : C \in IDC \wedge R \in DLR_{OWF}$  and:
  sameR( $C \xrightarrow{R} DC, I \xrightarrow{R} O$ )(SBox, OBox)
  sameR( $DC \xrightarrow{R} C, O \xrightarrow{R} I$ )(SBox, OBox)
// create RTBox, with the role restrictions that will be added to the ARBox
Init. RTBox such that  $(AA \xrightarrow{S} O)_{RTBox}$  and:
  minC( $AA \xrightarrow{S} O$ )RTBox = 1 ∧ exactC( $AA \xrightarrow{S} O$ )RTBox = 0
  maxC( $AA \xrightarrow{S} O$ )RTBox = ∞
  If  $\exists R : (I \xrightarrow{R} O)_{OBox}$  then minC( $AA \xrightarrow{S} O$ )RTBox = max∀R(minC( $I \xrightarrow{R} O$ )OBox)
  If  $\exists R : (I \xrightarrow{R} O)_{OBox}$  then exactC( $AA \xrightarrow{S} O$ )RTBox = exactC( $I \xrightarrow{R} O$ )OBox
  If  $\exists R : (I \xrightarrow{R} O)_{OBox}$  then maxC( $AA \xrightarrow{S} O$ )RTBox = min∀R(maxC( $I \xrightarrow{R} O$ )OBox)
// create the new response concept dependent on DC, with the OBox and RTBox
Init. Create-Output-Concept  $\delta_{COCT} = (O, \{Response\}, \emptyset, RTBox, OBox, \{DC\}, \emptyset, p)$ 
Exec.  $\delta_{COCT}$ 

Init. and Exec. Save-Exit-Context Command

```

7.3.1.2 Suggestion

A Follow-Role-CreateAndFill pattern command can only be suggested or executed if at least one task-definition exists in the current workflow-definition context. A set of Follow-Role-CreateAndFill pattern commands, Φ_{FRCF} , is suggested through the execution of [Algorithm 28](#), which, according to the domain ontology, suggests new task-definitions dependent on the currently existent task-definitions.

Algorithm 28: Suggestion of Follow-Role-CreateAndFill pattern commands in a workflow-definition context.

```

// it. all task/event-defs., X, with output/data concepts, C, of domain type D
// also it. all domain concepts, E
 $\forall X, C, D, E : X \in AD_{WD} \wedge E \in DLAC_{OD} \wedge$ 
   $\wedge ((isTDef(X) \wedge C \in fTDef_{IOO}(X) \wedge (C, D) \in fTDef_{\Delta_{OD}}(X)) \vee$ 
   $\vee (isEDef(X) \wedge C \in fEDef_{IOI}(X) \wedge (C, D) \in fEDef_{\Delta_{OD}}(X)))$ 
  // it. rels., R, from D to E in the domain ontology

```

```

 $\forall R : (D \xrightarrow{R} E)_{TBox_{OWF}}$ 
  // suggest command with unit concept dependent on C
  Init.  $SBox : sameR(D \xrightarrow{R} E, D \xrightarrow{R} E)_{(TBox_{OWF}, SBox)}$ 
  Create new  $(X, C, SBox, E, AT, AA, W, I, l, d, au, I, O, 5)$  in  $\Phi_{FRCF}$ 
  // it. rels.,  $R$ , from  $E$  to  $R$  in the domain ontology
 $\forall R : (E \xrightarrow{R} D)_{TBox_{OWF}}$ 
  // suggest command with unit concept dependent on C
  Init.  $SBox : sameR(E \xrightarrow{R} D, E \xrightarrow{R} D)_{(TBox_{OWF}, SBox)}$ 
  Create new  $(X, C, SBox, E, AT, AA, W, I, l, d, au, I, O, 5)$  in  $\Phi_{FRCF}$ 

```

7.3.2 Partition Pattern Commands

The partition pattern command creates several new task-definitions, incrementally asking for the parts of given domain objects by following meronymic role restrictions from and to domain concepts (see Figure 38 for an illustration). For instance, consider the document domain ontology in Figure 9. A *Section* is formed by one or more *Paragraph*, which are formed by one or more *Sentence*. These are related through the meronymic role *contains*. For this domain ontology, a partition pattern command that creates two task-definitions, of type *CreateAndFillTask*, would be suggested. The first task-definition would ask for the paragraphs of a given section. The second task-definition would ask for the sentences of the previously submitted paragraphs.

A partition pattern command in a workflow-definition context is a structure $PW := (r, SBox, W, TI, tp, au, p)$, where:

- $r \in DLR_{OD}$ is the meronymic role;
- $SBox \subseteq TBox_{OD}$ is a set of restrictions on the role r that associate two domain concepts;
- $W \in DLAC_{OWF}$ is the atomic worker concept for new task-definitions;
- $TI \in DLAC_{OWF}$ is the atomic task interface concept for new task-definitions;
- $tp \in PD_{OCF}$ is the priority for new task-definitions (defaults to *medium*);
- $au \in DLv_{OWF}$ is the assignments per unit value for new task-definitions (defaults to 1);
- $p \in [1, 5]$ is the priority value of the pattern command.

The structure PW is a valid partition pattern command if $|SBox| \geq 1$.

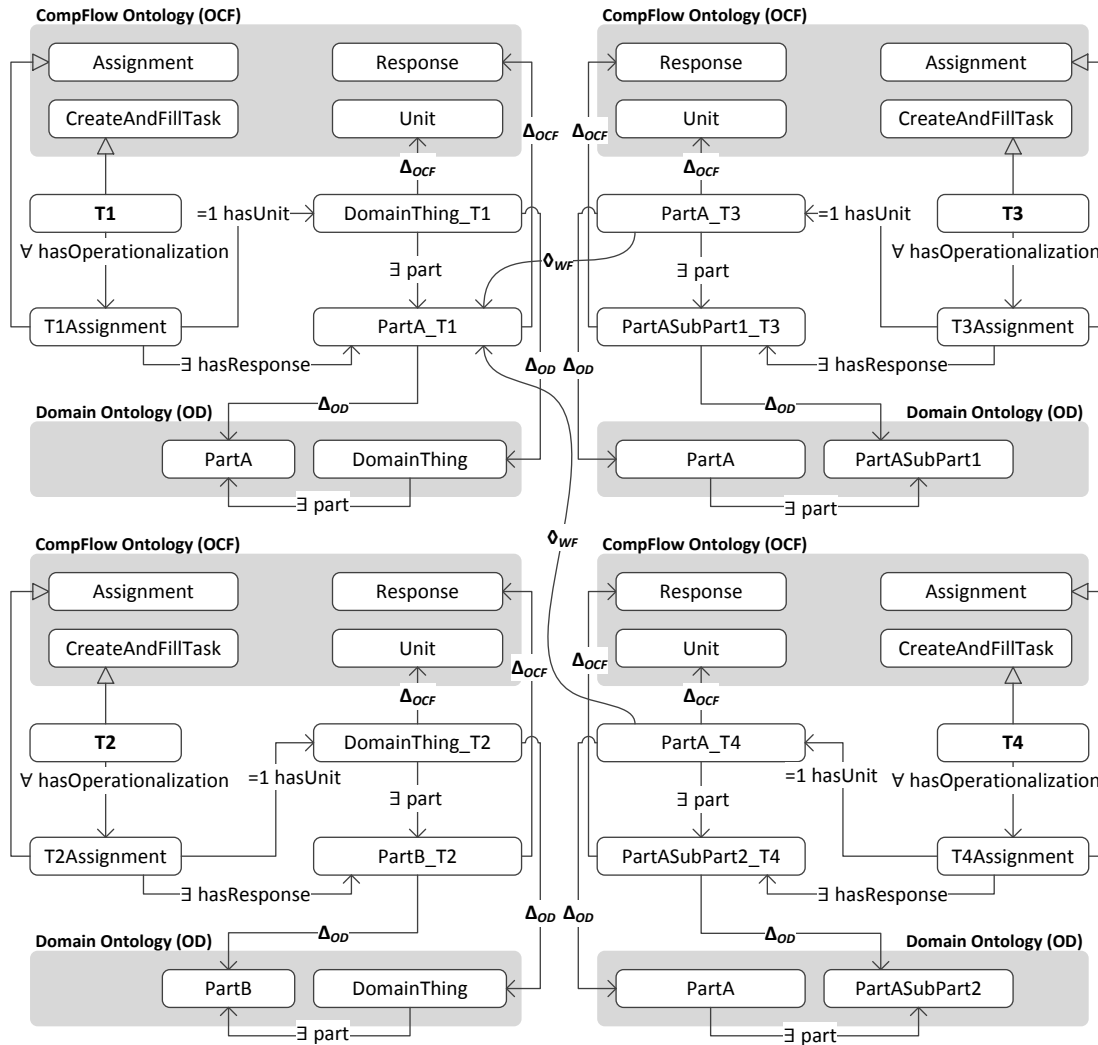


Figure 38: Result of the execution of a partition pattern command with four partition steps. Notice that T_1 and T_2 can start concurrently. T_3 and T_4 can also start concurrently, but only after T_1 is finished.

7.3.2.1 Execution

As presented in [Algorithm 29](#), the execution of a PW results in the execution of a sequence of atomic commands.

Algorithm 29: Execution of a partition pattern command in a workflow-definition context.

```

// it. all meronymic rels. between two concepts, C and D and
// build a new partition task-definition (C must be at the top of the tree)
 $\forall C, D : (C \xrightarrow{r} D)_{SBox} \wedge \nexists E : (E \xrightarrow{R} C)_{SBox}$ 
  Invoke BuildPartition ( $\emptyset, C, D$ )

// this routine builds a new task-definition that partitions C into D

```

```

Start of routine BuildPartition ( $WC, C, D$ )
  Init.  $AT$  and  $AA$  with unique names derived from  $C$  and  $D$ 
  Init.  $l$  and  $d$  with the unique name of  $AT$ 

  // create the new partial task-definition
  Init. Create-Task-Definition  $\delta_{CTDW} = (AT, AA, W, TI, tp, l, d, au, p)$ 
  Exec.  $\delta_{CTDW}$ 

  // create the unit concept with domain type  $C$ 
  Init.  $I$  with unique name derived from  $C$ 
  Init. Create-Input-Concept  $\delta_{CIC} = (I, \{Unit\}, \emptyset, RTBox, \emptyset, \{C\}, WC, \emptyset, p)$ 
  Exec.  $\delta_{CIC}$ 

  // create the response concept with domain type  $D$ 
  Init.  $O$  with unique name derived from  $D$ 
  // include the meronymic relationship from  $C$ 
  Init.  $OBox$  such that  $sameR(C \xrightarrow{r} D, I \xrightarrow{r} O)_{(SBox, OBox)}$ 
  Init.  $RTBox$  such that  $(AA \xrightarrow{hasResponse} O)_{RTBox}$  and:
     $minC(AA \xrightarrow{hasResponse} O)_{RTBox} = minC(C \xrightarrow{r} D)_{SBox}$ 
     $exactC(AA \xrightarrow{hasResponse} O)_{RTBox} = exactC(C \xrightarrow{r} D)_{SBox}$ 
     $maxC(AA \xrightarrow{hasResponse} O)_{RTBox} = maxC(C \xrightarrow{r} D)_{SBox}$ 
  Init. Create-Output-Concept  $\delta_{COCT} = (O, \{Response\}, \emptyset, RTBox, OBox, \{D\}, \emptyset, p)$ 
  Exec.  $\delta_{COCT}$ 

  Init. and Exec. Save-Exit-Context Command

  // continue following the meronymic relationships to other concepts
  // and build new partition task-definitions for them
   $\forall E : (D \xrightarrow{r} E)_{SBox}$ 
    Invoke BuildPartition ( $\{O\}, D, E$ )
End of routine BuildPartition

```

7.3.2.2 Suggestion

Partition pattern commands are suggested to the creator in a workflow-definition context if a meronymic role structure is detected in the domain ontology. A set of partition pattern commands, Φ_{PW} , is suggested through two different strategies: (i) lexical verification and (ii) lexical and structural verification. The first (i) performs a purely lexical verification of the roles present in the domain ontology (see [Algorithm 30](#)), while the latter (ii) performs both a lexical and a structural verification (see [Algorithm 31](#)).

Algorithm 30: Suggestion of partition pattern commands in a workflow-definition context: lexical verification.

```
// it. all meronymic roles, R, in the domain ontology
∀R : R ∈ DLROD ∧ isMeronymicRole(R)
  // create the partial SBox with all the R role restrictions
  Init. SBox = ∅
  ∀C, D : C ∈ DLACOD ∧ D ∈ DLACOD ∧ (C  $\xrightarrow{R}$  D)TBoxOD
    sameR(C  $\xrightarrow{R}$  D, C  $\xrightarrow{R}$  D)(TBoxOD, SBox)
  // if the partial SBox is not empty, suggest the command
  If SBox ≠ ∅ then
    Create new (R, SBox, W, TI, tp, au, 3) in ΦPW
```

The structural verification performed by [Algorithm 31](#) restricts the meronym relationship to concepts with a common hypernym (super-concept).

Algorithm 31: Suggestion of partition pattern commands in a workflow-definition context: lexical and structural verification.

```
// it. all meronymic roles, R, in the domain ontology, which
// relate individuals of E to smaller and contained individuals (still) of E
∀R : R ∈ DLROD ∧ isMeronymicRole(R) ∧ ∃E : (E  $\xrightarrow{R}$  E)TBoxOD
  // create the partial SBox with all the R role restrictions
  Init SBox = ∅
  ∀C, D : C ∈ DLACOD ∧ D ∈ DLACOD ∧ [C ⊆ E]TBoxOD ∧
    ∧ [D ⊆ E]TBoxOD ∧ (C  $\xrightarrow{R}$  D)TBoxOD
    sameR(C  $\xrightarrow{R}$  D, C  $\xrightarrow{R}$  D)(TBoxOD, SBox)
  // if the partial SBox is not empty, suggest the command
  If SBox ≠ ∅ then
    Create new (R, SBox, W, TI, tp, au, 4) in ΦPW
```

This type of partition pattern command does not include higher-level parts (e.g. sections) as contextual information of task-definitions that partition smaller parts (e.g. sentences). For instance, as illustrated in [Figure 38](#), a *DomainThing* is not present as the *UnitContext* of task-definitions that partition *PartA* into *PartASubPart1* and *PartASubPart2*. However, a type of partition pattern command that considers such a scenario may be formulated as an extension of this partition pattern command.

7.3.3 Assembler Pattern Commands

The assembler pattern command creates several new task-definitions, which incrementally assemble the given parts into new domain objects according to meronymic role restrictions from and to domain concepts (see [Figure 39](#) for an illustration). It provides the inverse set of operations of the partition pattern command. For instance, once again considering the document domain ontology in [Figure 9](#), an assembler pattern command that creates two task-definitions of type *CreateAndFillTask* are suggested. The first task-definition asks for paragraphs that assemble sets of given sentences. The second task-definition asks for sections that assemble the previously submitted paragraphs.

An assembler pattern command in a workflow-definition context is a structure $AW := (r, SBox, W, TI, tp, au, p)$, where:

- $r \in DLR_{OD}$ is the meronymic role;
- $SBox \subseteq TBox_{OD}$ is a set of restrictions on the role r that associate two domain concepts;
- $W \in DLAC_{OWF}$ is the atomic worker concept for new task-definitions;
- $TI \in DLAC_{OWF}$ is the atomic task interface concept for new task-definitions;
- $tp \in PD_{OCF}$ is the priority for new task-definitions (defaults to *medium*);
- $au \in DLv_{OWF}$ is the assignments per unit value for new task-definitions (defaults to 1);
- $p \in [1,5]$ is the priority value of the pattern command.

The structure AW is a valid assembler pattern command if $|SBox| \geq 1$.

7.3.3.1 Execution

As presented in [Algorithm 32](#), the execution of an AW results in the execution of a sequence of atomic commands.

Algorithm 32: Execution of an assembler pattern command in a workflow-definition context.

```

// do a bottom-up recursive construction of assembly task-definitions for all
// meronymic trees of concepts with the top-level concept C
 $\forall C \nexists E : (E \xrightarrow{r} C)_{SBox}$ 
  Invoke BuildAssembler ( $\emptyset, C, \{D | (C \xrightarrow{r} D)_{SBox}\}$ )
```

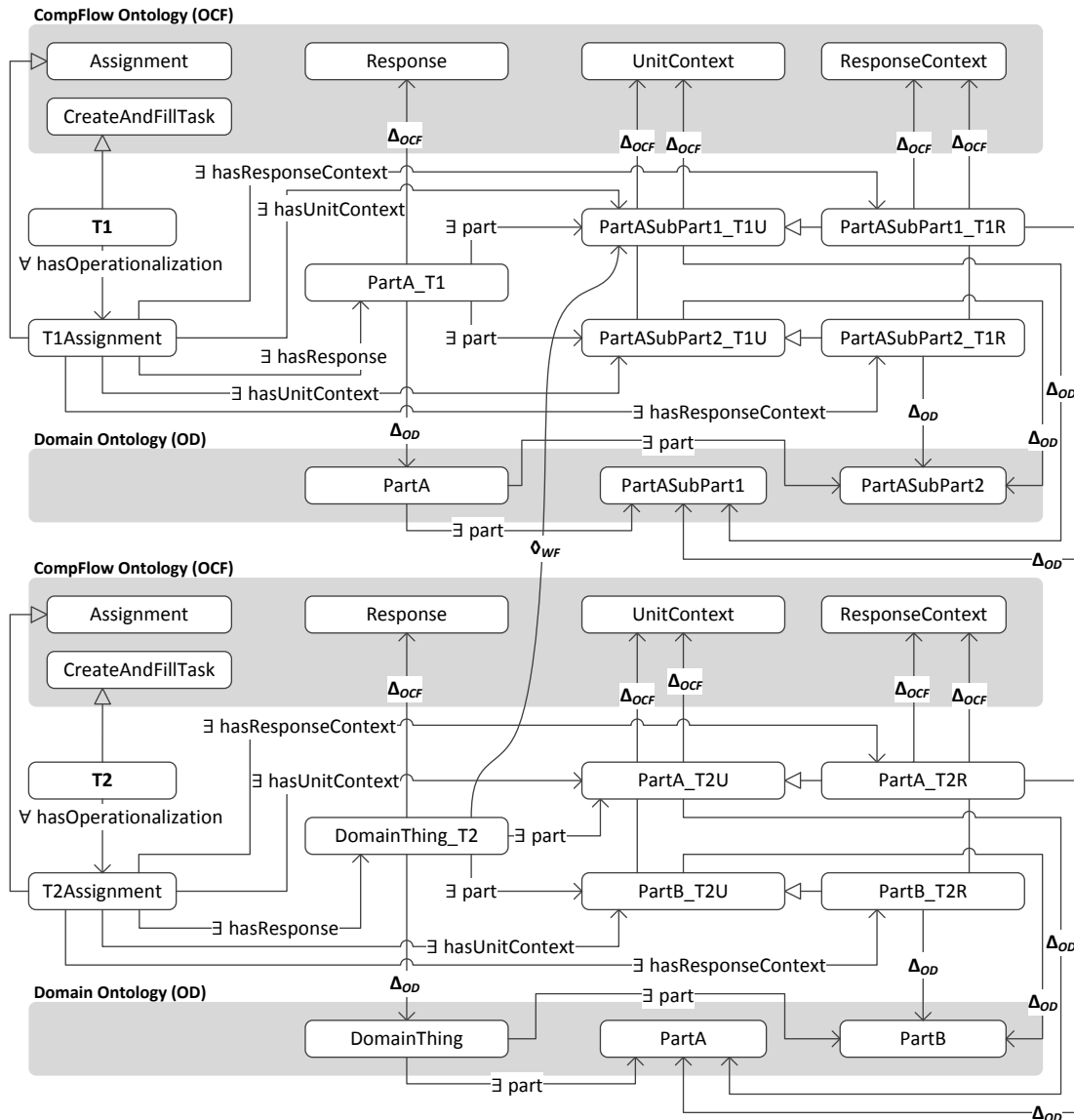


Figure 39: Result of the execution of an assembler pattern command.

```

// this routine builds a new task-def. that assembles all parts in DS into C
Start of routine BuildAssembler (C, DS)
Init. AT and AA with unique names derived from C and DS
Init. l and d with the unique name of AT

// create the new partial task-definition
Init. and Exec. Create-Task-Definition  $\delta_{CTDW} = (AT, AA, W, TI, tp, l, d, au, p)$ 

// create the response concept with domain type C
Init. O with unique name derived from C

```

```

Init. ARB such that  $[AA \sqsubseteq \exists hasResponse.O]_{ARB}$ 
Init. Create-Output-Concept  $\delta_{COCT} = (O, \{Response\}, \emptyset, ARB, \emptyset, \{C\}, \emptyset, p)$ 
Exec.  $\delta_{COCT}$ 

// it. all parts (concepts), D, and create a unit context concept
// and a dependent response context concept, which filters the input parts
 $\forall D \in DS$ 
  // first build the depended upon (lower-level) task-definitions and
  // retrieve their response concept in WC
Init and Exec. Save-Exit-Context Command
Invoke WC = BuildAssembler (D,  $\{E|(D \xrightarrow{r} E)_{SBox}\}$ )
Init. and Exec. Update-Task-Definition of AT

// create the unit context concept from D, dependent on the concept in WC
Init. I with unique name derived from D
Init. RTBox such that  $(AA \xrightarrow{hasUnitContext} I)_{RTBox}$  and:
   $minC(AA \xrightarrow{hasUnitContext} I)_{RTBox} = minC(C \xrightarrow{r} D)_{SBox}$ 
   $exactC(AA \xrightarrow{hasUnitContext} I)_{RTBox} = exactC(C \xrightarrow{r} D)_{SBox}$ 
   $maxC(AA \xrightarrow{hasUnitContext} I)_{RTBox} = maxC(C \xrightarrow{r} D)_{SBox}$ 
Init. and Exec. Create-Input-Concept
   $\delta_{CIC} = (I, \{UnitContext\}, \emptyset, RTBox, \emptyset, \{D\}, WC, \emptyset, p)$ 

// create the response context concept from D, dependent on I
Init. P with unique name derived from D
Init. OBox such that  $sameR(C \xrightarrow{r} D, O \xrightarrow{r} P)_{(SBox, OBox)}$ 
Init. RTBox such that  $(AA \xrightarrow{hasResponseContext} P)_{RTBox}$  and:
   $minC(AA \xrightarrow{hasResponseContext} P)_{RTBox} = minC(C \xrightarrow{r} D)_{SBox}$ 
   $exactC(AA \xrightarrow{hasResponseContext} P)_{RTBox} = exactC(C \xrightarrow{r} D)_{SBox}$ 
   $maxC(AA \xrightarrow{hasResponseContext} P)_{RTBox} = maxC(C \xrightarrow{r} D)_{SBox}$ 
Init. and Exec. Create-Output-Concept
   $\delta_{COCT} = (P, \{ResponseContext\}, \emptyset, RTBox, OBox, \{D\}, \{I\}, p)$ 

Init. and Exec. Save-Exit-Context Command
Return {O}
End of routine BuildAssembler

```

7.3.3.2 Suggestion

Similarly to partition pattern commands, assembler pattern commands are suggested to the creator in a workflow-definition context if a meronymic role structure is detected in the domain ontology. A set of assembler pattern commands, Φ_{AW} , is also suggested

through two different strategies: lexical verification and lexical and structural verification. The algorithms are the same as those used in the suggestion of partition pattern commands.

This assembler pattern command does not include smaller parts as contextual information of task-definitions that assemble higher-level parts. For instance, a *PartA-SubPart1* is not present as the *UnitContext* of task-definitions that assemble *PartA* and *PartB* into the *DomainThing*. Regardless, a type of assembler pattern command that considers such a scenario may be formulated as an extension of this assembler pattern command.

7.3.4 Other Pattern Commands

Any type of pattern command may be included in the CompFlow assisted construction process. In the specific case of the document ontology presented in [Figure 9](#), the process of translating text to another language can be seen as a transformation operation, which leads to the emergence of a new modified instance of the same concept. Such an abstraction can be modelled as a new, more complex and complete, transformation pattern command.

Other patterns may be highly domain specific and applicable to scenarios where the application domain or the domain ontology remains the same for several different workflow-definitions.

7.4 STRATEGIES

Strategies represent sets of atomic and pattern commands. Strategies can be defined in the context of a particular ontology construction methodology or in the context of a particular domain. In order to facilitate the construction of workflow-definition ontologies using a particular domain ontology, a strategy composed of several atomic and pattern commands that result in common task-definitions and sub-workflow-definitions for that domain can be defined. By using a particular strategy, creators can easily build workflow-definitions by composing and mixing these common task-definitions and sub-workflow-definitions.

Further abstractions from the domain of the workflow-definition can also be considered when defining strategies. For instance, the structural patterns employed in the con-

text of a particular ontology construction method may define common task-definitions and sub-workflow-definitions that employ general processes such as partitioning and assembling (as represented by the partition and assembler pattern commands).

More concrete possibilities are the definition of strategies according to change models or ontology evolution methods such as the ones proposed by Stojanovic [68] and Pittet et al. [57]. These approaches capture the possible atomic operations that can be performed to modify and evolve an ontology.

The CompFlow assisted construction process always provides a base strategy that includes all available atomic and pattern commands.

7.5 SUMMARY

Although the CompFlow workflow-definition method demands a certain amount of ontology engineering expertise from the creator, a construction process that aids creators and orients their focus towards the application domain has been proposed.

The proposed construction process relies on the structural and semantic analysis of the domain ontology to iteratively suggest workflow-definition construction commands. The set of possible commands can be extended and enriched with custom commands. Also, construction commands can be triggered by patterns in the domain and workflow-definition ontologies and aggregated into construction strategies.

A relatively small set of pattern commands is proposed. However, the amount of patterns found in the construction and alignment of ontologies [63] suggests that the potential growth of the pattern command and strategy sets is high.

Part III

POSTAMBLE

EVALUATION AND USE CASE SCENARIOS

To evaluate the proposed workflow-definition method and assisted construction process, an environment for the workflow-definition construction, instantiation and execution has been implemented. The prototype implementation includes two main components: a construction framework, and an instantiation and execution engine. Furthermore, the implementation has been tested through three use case scenarios in different application domains.

In this chapter, an overview of the implemented construction, instantiation and execution environment is given along with a detailed description of each use case scenario and the obtained results.

8.1 COMPFLOW CONSTRUCTION FRAMEWORK

The CompFlow construction framework provides a visual environment and an [API](#) for the assisted construction of workflow-definitions. It relies in the CompFlow iterative construction process to suggest construction commands to the creator through the analysis of the given domain ontology.

8.1.1 *Visual Workflow-Definitions*

The visual environment of the CompFlow construction framework features four different views (see [Figure 40](#)): (i) the definition view, (ii) the definition detail view, (iii) the suggested command view and (iv) the command detail view.

The definition view (i) provides a visual representation of the current activity-definition. This view adapts to the current construction context, i. e. it differs for each type of workflow-definition component (activity-definition or transition-definition). If the creator is building an activity-definition inside another workflow-definition, both the workflow-definition and the activity-definition will be presented by the definition view.

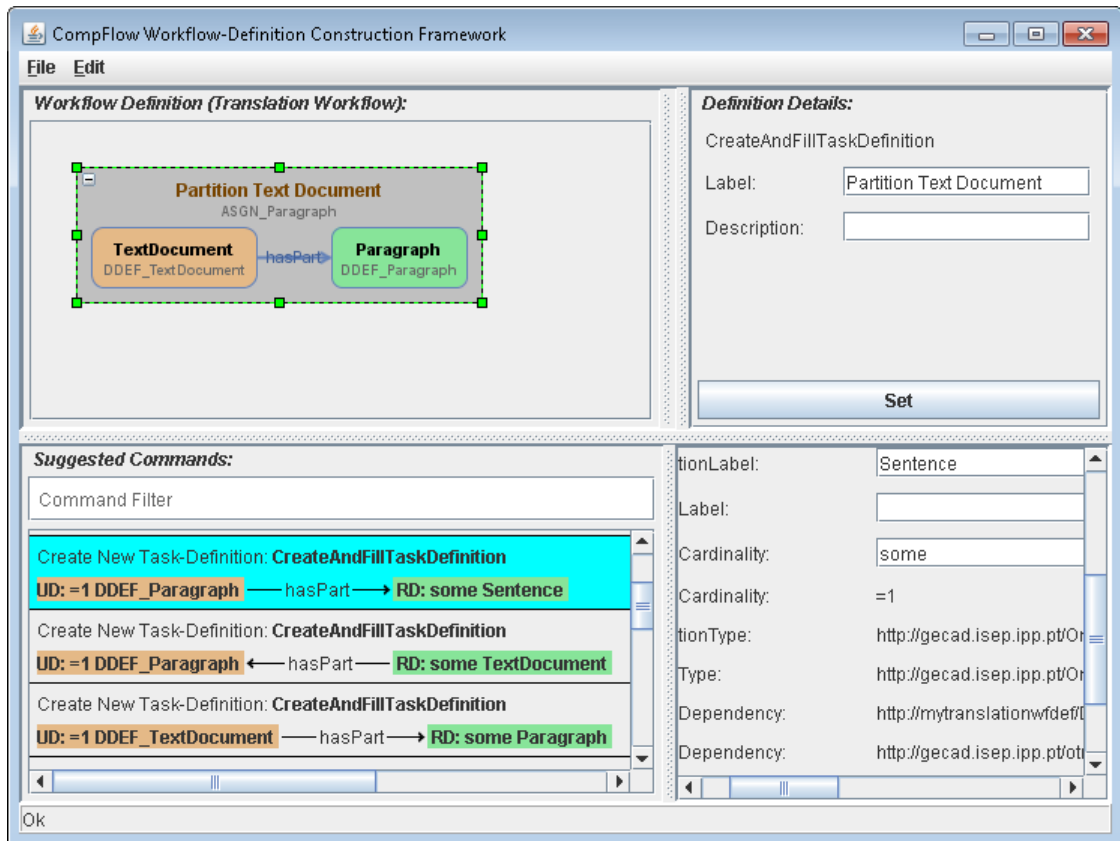


Figure 40: The visual workflow-definition construction environment prototype. Definition view (i) at the top-left area, definition detail view (ii) at the top-right area, suggested command view (iii) at the bottom-left area, and command detail view (iv) at the bottom-right area.

The definition detail view (ii) presents the attributes of the selected activity-definition or transition-definition.

The suggested command view (iii) provides a set of suggested commands to the creator. These commands are ordered according to their priority values and depend on the current construction context. Before accepting and executing a command, the creator may choose to set or modify the command attributes through the command detail view (iv).

8.1.2 Construction Framework Architecture

The implementation architecture of the CompFlow construction environment is presented in [Figure 41](#). It is built on top of the Apache Jena framework for Semantic Web applications¹ and contains the following modules:

- The Ontology Module - provides a set of tools that allow the structural analysis of [OWL](#) ontologies and provides a [DL](#)-based handling [API](#);
- The Command Module - handles all available commands and suggestions;
 - The Atomic Command Module - contains the set of implemented atomic commands;
 - The Pattern Command Module - contains the set of implemented pattern commands;
- The Strategy Module - contains the set of implemented strategies;
- The Context Management Module - handles the context hierarchy and timelines (for undo and redo operations);
- The Graphical User Interface ([GUI](#)) Module - provides a graphical interface to allow the construction of workflow-definitions.

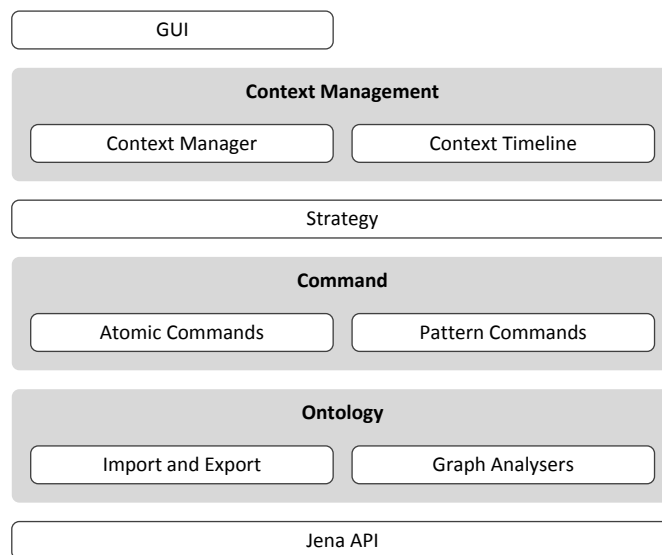


Figure 41: Implementation architecture of the CompFlow construction framework.

¹<https://jena.apache.org/>

Notice that the construction framework may be used as a library (without the GUI module), allowing, amongst other applications, the development of new back-end and alternative GUI implementations.

The ontology module analyses the structure of ontologies and provides a DL-based graph-oriented API to access and read the workflow-definition and domain ontologies. It also imports and exports workflow-definition projects from and to OWL ontology files.

The implementation of any kind of command is possible through the command module. The currently implemented commands correspond exactly to those specified in the assisted construction process (see Chapter 7).

Each context instance in the CompFlow construction framework provides a complete representation of the current state of the workflow-definition construction environment. In order to provide *undo* and *redo* actions, a timeline of contexts is kept. This timeline stores incremental changes to the workflow-definition.

8.2 COMPFLOW ENGINE

The CompFlow engine is a workflow-definition instantiation and execution engine and service, based on the CompFlow workflow-definition method (see Chapter 5).

Among the concepts found in the CompFlow ontology, two categories can be found: (i) concepts that affect and depend on the workflow-definition and (ii) concepts that affect and depend on the deployment of the instantiation and execution engine. Figure 42 denotes these categories.

Analogously, since each deployment of the instantiation and execution engine may interact with different workers and services through different types of interfaces, each deployment may contain its own extension of the CompFlow ontology: the deployment ontology. Since the possible types of *Job*, *State* and *Priority* are fixed for the instantiation

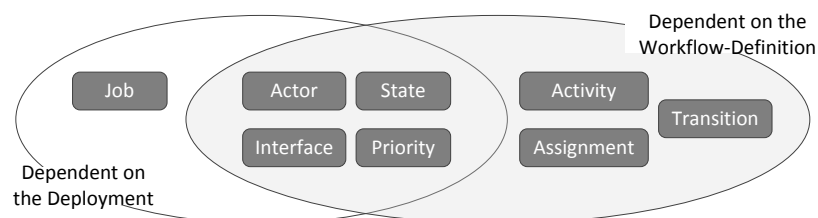


Figure 42: Deployment-dependent and workflow-definition-dependent concepts in the CompFlow ontology.

and execution engine, the purpose of the deployment ontology is to extend the *Actor* and *Interface* concepts accordingly.

A deployment of the CompFlow engine must include:

- The engine configuration file;
- The deployment ontology, with additional custom Interface sub-concepts, along with all its imported ontologies;
- A Java ARchive (JAR) file with the Java implementation for the custom types of interfaces declared in the deployment ontology;
- The interface mappings between *TaskInterface* sub-concepts in the deployment ontology and the corresponding Java class implementations in the JAR file;
- A location-mapping file, which maps the deployment ontology and its imported ontologies to a specific location.

While deployment-dependent concepts are typically extended in deployment ontologies, workflow-definition-dependent concepts are extended in workflow-definition ontologies.

8.2.1 Engine Architecture

The implementation architecture of the CompFlow engine is presented in [Figure 43](#). It consists of a base layer of external libraries and frameworks that include the Apache Jena framework, the Restlet framework² and the Freemarker template engine³. On top of these libraries, several modules with different purposes exist:

- The Ontology Module - allows the analysis of workflow-definition ontologies;
- The Extensible Interface API Module - allows new types of interfaces to be implemented and deployed;
- The Interface Management Module - handles all registered task and event interfaces;
- The Task Module - handles the state and execution of tasks;
- The Workflow Module - handles the state and execution of workflows;
- The Job Module - handles the top-level container of workflows (jobs);

²<http://restlet.com/>

³<http://freemarker.org/>

- The Engine Context - aggregates (and allows the access to) all the shareable components that form a CompFlow engine instance.

The interaction with the CompFlow engine is performed through REpresentational State Transfer ([REST](#))ful endpoints, which can be used to perform operations over jobs, job templates, workflows, tasks, interfaces and actors (workers or requesters).

Each of the modules found in the CompFlow engine architecture are further described in the following sub-sections.

8.2.2 The Ontology Module

The ontology module, as presented in the CompFlow construction framework, provides the necessary features to perform the structural and semantic analysis of workflow-definition ontologies according to the CompFlow workflow-definition method.

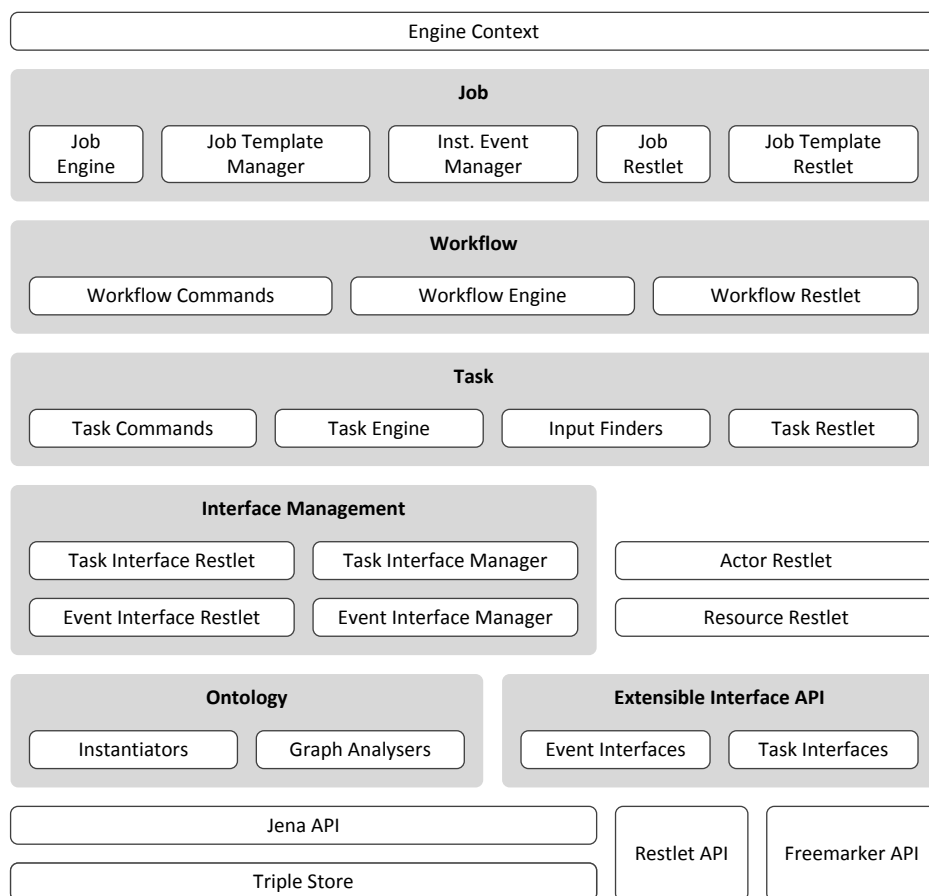


Figure 43: Implementation architecture of the CompFlow engine.

Additionally, the ontology module contains activity-definition and transition-definition *instantiators*. *Instantiators* perform an in-depth analysis of the workflow-definition ontology, and are responsible for the instantiation of activity-definitions and transition-definitions. They are used by the CompFlow engine to create new activities and establish transitions as the workflow execution progresses.

8.2.3 The Extensible Interface Module

The Extensible Interface module allows the integration of different types of custom interfaces. This is achieved by mapping a Java class implementation to an *Interface* sub-concept in the deployment ontology. These mappings define the implementation for a particular type of interface.

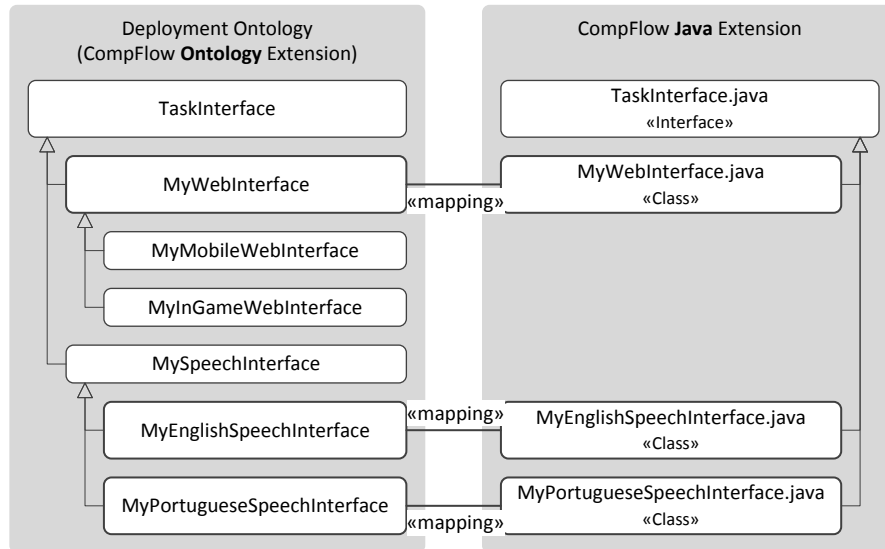
For each *Interface* sub-concept in the deployment ontology, multiple interface individuals can be registered in the engine. These individuals have an associated Java instance of the mapped Java class, which handle all interface related operations.

Java interface classes implement either:

- A type of interface itself, which directly interacts with machine or human workers (e. g. a Web interface);
- The interaction with a particular type of external interface (e. g. a speech interface or remote terminal interface) [41].

An *Interface* sub-concept in the deployment ontology must always have a direct or indirect mapping to a Java class implementation. For instance, as presented in [Figure 44](#), all *MyWebInterface* individuals will be handled by instances of the Java *MyWebInterface.java* class. Since a mapping does not exist for any of the *MyMobileWebInterface* and *MyInGameWebInterface* sub-concepts, the indirect mapping through *MyWebInterface* is considered. Consequently, all *MyMobileWebInterface* and *MyInGameWebInterface* individuals will also be handled by instances of the Java *MyWebInterface.java* class.

In the case of both *MyEnglishSpeechInterface* and *MyPortugueseSpeechInterface* sub-concepts, a direct mapping exists and overrides all the mappings of any parent *TaskInterface* sub-concept.

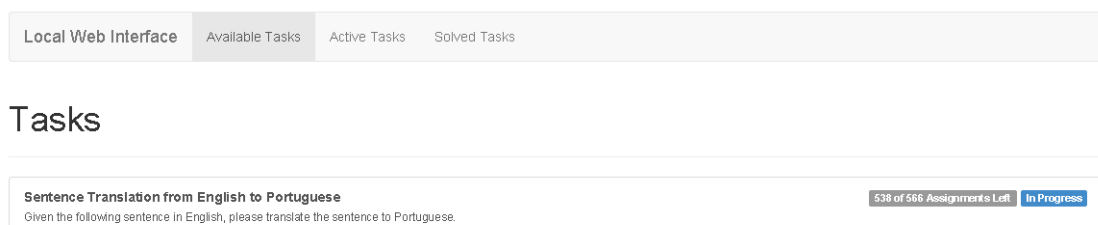
Figure 44: *TaskInterface* mapping example.

8.2.3.1 The *LocalWebCrowdInterface*

The current implementation of the CompFlow engine includes a single task interface implementation: the *LocalWebCrowdInterface*. The *LocalWebCrowdInterface* is a task interface similar to those of CrowdFlower and Mechanical Turk, i.e. it allows human workers to solve tasks and assignments in a Web environment. Figure 45 shows a screenshot of an instance of the *LocalWebCrowdInterface* with a list of available tasks presented to the worker.

The *LocalWebCrowdInterface* requires each task-definition to have an associated FreeMarker template file with the task-definition interface. This template must output **HTML** and is used to present and collect data to and from the worker.

In the FreeMarker template, all the task data and meta-data are available as a bundle of key-value pairs. Alternatively, the task data and meta-data can be accessed through a *JavaScript* object. Regardless of the approach, the data in these structures include the:

Figure 45: Screenshot of the *LocalWebCrowdInterface*.

- *interfaceUri* - the identifier of the interface;
- *workerUri* - the identifier of the worker;
- *jobUri* - the identifier of the job;
- *taskUri* - the identifier of the task;
- *assignmentUri* - the identifier of the assignment;
- *unitUris* - the list of unit identifiers;
- *unitContextUris* - the list of unit context identifiers;
- *responseTypeUris* - the list of response concept identifiers;
- *responseContextTypeUris* - the list of response context concept identifiers;
- *objs* - a key-value bundle of individuals (the data);
- *types* - a key-value bundle of concepts (meta-data).

The Freemarker template engine data structure can be used as presented in [Algorithm 33](#).

Algorithm 33: Usage example of the Freemarker template engine data structure.

```

<!--to list all unit individuals-->
<#list unitUris as unitUri>
<div class="unitItem ${objs[unitUri].friendlyUri}">
  <#if objs[unitUri].label??> ${objs[unitUri].label[0]}
  <#else> ${objs[unitUri].shortUri} </#if>
</div>
</#list>

<!--to present text fields for each datatype role of the first response concept-->
<div class="responseItem ${types[responseTypeUris[0]].friendlyUri}">
<#list types[responseTypeUris[0]].dataTypeProperties as dataTypeProperty>
  <label>
    <#if dataTypeProperty.label??> ${dataTypeProperty.label[0]}
    <#else> ${dataTypeProperty.shortUri} </#if>
  </label>
  <input type="text" class="responseItemProperty ${dataTypeProperty.friendlyUri}" />
</#list>
</div>

```

The same can be achieved through *JavaScript* and the *jQuery API*, as presented in [Algorithm 34](#).

Algorithm 34: Usage example of the JavaScript data structure with the jQuery API.

```

// to list all unit individuals
var unitContainerDiv = $('#unitContainer');
var unitUrisLen = data.unitUris.length;
for(var i=0; i<unitUrisLen; i++) {
  var unitUri = data.unitUris[i];
  var unitItemDiv = $('<div>', { class: 'unitItem ' +
    data.objs[unitUri].friendlyUri });
  if(data.objs[unitUri].label)
    unitItemDiv.text(data.objs[unitUri].label[0]);
  else unitItemDiv.text(data.objs[unitUri].shortUri);
  unitContainerDiv.append(unitItemDiv);
}

// to present text fields for each datatype role of the first response concept
var responseItemDiv = $('.' + data.types[responseTypeUris[0]].friendlyUri);
var propLen = data.types[data.responseTypeUris[0]].dataTypeProperties.length;
for(var i=0; i<propLen; i++) {
  var prop = data.types[data.responseTypeUris[0]].dataTypeProperties[i];
  var propLabel = $('<label>');
  if(prop.label) propLabel.text(prop.label[0]);
  else propLabel.text(prop.shortUri);
  responseItemDiv.append(propLabel);
  var propInput = $('<input>', { type: 'text',
    class: 'responseItemProperty ' + prop.friendlyUri });
  responseItemDiv.append(propInput);
}

```

Using one of these approaches, the UI creator can build HTML interfaces with access to all the assignment data and meta-data.

The submission of the assignment triggers a *JavaScript* algorithm that analyses the *id*, *name* and *class* attributes of the elements in the Document Object Model (DOM), and searches for annotations that may link these elements to the output concepts (and their roles) found in the *JavaScript* data structure. If these annotations are present (e.g. in input and select elements), their values are loaded into a data structure that is sent to the CompFlow engine restlet responsible for receiving assignment results.

8.2.3.2 *The RESTfulEventInterface*

A single type of event interface, the *RESTfulEventInterface*, is also available. The *RESTfulEventInterface* is a simple REST publish-subscribe interface implementation for events.

8.2.4 *The Interface Management Module*

Since each registered interface individual must have its own Java class instance, a manager that analyses the interface mappings between *Interface* sub-concepts and Java classes and provides the appropriate Java class instances for a given interface individual is necessary. In order to fulfil this purpose, the interface management module features a task interface manager and an event interface manager.

When the engine requires interaction with a particular interface, it retrieves a Java class instance from the appropriate interface manager.

CRUD-like operations on top of interface individuals can be performed through a RESTful interface provided by the task and event interface restlets. These operations allow new interfaces to be registered or updated in run-time.

8.2.5 *The Workflow and Task Modules*

The CompFlow engine handles the execution of workflows through two main sub-engines: the Workflow engine and the Task engine. The Workflow engine handles the execution of all types of activities. However, due to their increased complexity, the execution of tasks is delegated to the Task engine.

The Task engine can be deployed independently from the Workflow engine in order to exclusively execute tasks. The Workflow engine, however, requires a Task engine in order to dispatch and execute tasks. The operations performed by these two sub-engines are executed through asynchronous commands that run in a thread pool.

In order to analyse the task-definition and define the input of each assignment, the engine requires an *InputFinder*. The *InputFinder* is a component that heavily relies on the CompFlow workflow-definition method. It extracts the input data from the job's operational *ABox* and associates these data with each assignment.

8.2.6 *The Job Module*

The *Job* module handles jobs and job templates. A job template is a predefined set of workflow-definitions to be instantiated and executed in a new job. A job template contains:

- The job template configuration and descriptor;
- An optional input dataset;
- One or more workflow-definitions with:
 - The workflow-definition ontology;
 - All required imported ontologies;
 - The location mappings for each ontology;
 - A human interface Freemaker template for each task-definition (only required for human worker interfaces such as a *LocalWebCrowdInterface*).

A job template may be public or private. If it is private, the job template can only be accessed and used by its owner.

A job always belongs to a requester and can only be instantiated and executed from a previously created job template. In order to instantiate a job, the requester must select a job template and supply an input dataset (if one does not exist inside the job template).

Jobs are handled by the job sub-engine. The job sub-engine is slightly different from the workflow and task sub-engines since it does not deal with an activity, but instead with an environment that establishes the execution context for a set of workflows. Once a job is started, the execution of all its workflows is delegated to the workflow engine.

8.2.7 *External Libraries*

The Apache Jena and the Restlet frameworks have been chosen for their high flexibility and integration with multiple technologies.

The Jena framework for building Semantic Web applications is able to interact with triple stores through different persistence technologies such as SPARQL Database ([SDB](#)) for relational databases, Tuple Database ([TDB](#)) for single machine triple stores, and OWL In Memory ([OwlIM](#)) for high-performance semantic repositories. Also, it features

an extensible node [API](#), which allows the implementation of domain-specific models (analogous to data access layers).

The Restlet framework for Web [APIs](#) features a wide variety of extensions and allows the deployment of the implemented web services through several technologies such as an Apache Tomcat container.

The Freemarker Java Template Engine is used to process interface templates associated with each task-definition.

8.3 USE CASE SCENARIOS

The CompFlow construction framework and engine implementations have been used and tested through three different use case scenarios: the document translation scenario, the ontology alignment scenario and the Catalan constitution refinement scenario.

8.3.1 *The Document Translation Scenario*

The document translation scenario considers common applications of crowdsourcing such as translation, edition and proofing of text documents. Through the document domain ontology presented in [Figure 9](#), a divide-and-conquer strategy was employed to the translation problem, resulting in the sequence of task-definitions presented in [Figure 46](#):

- T₁ - partitions documents into paragraphs, and is performed by machine workers;
- T₂ - partitions paragraphs into sentences, and is performed by human workers;
- T₃ - translates sentences, and is performed by human or machine workers capable of translating sentences;
- T₄ - assembles previously translated sentences into paragraphs, and is performed by human workers;
- T₅ - assembles previously translated paragraphs into a text document, and is performed by machine workers.

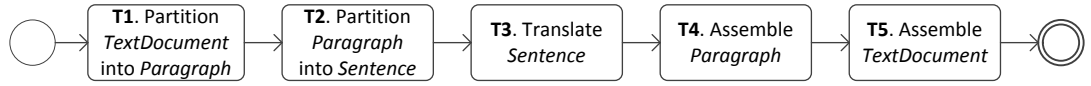


Figure 46: Overview of the document translation workflow-definition.

8.3.1.1 The Workflow-Definition

Taking into account the requirements and the conceptualization of the translation process, a workflow-definition was built using the CompFlow construction framework. To achieve the workflow-definition depicted in Figure 47, the sequence of commands presented in Table 17 were executed from the set of suggested commands.

Each task-definition has an assignment per unit value of one (i. e. $au = 1$).

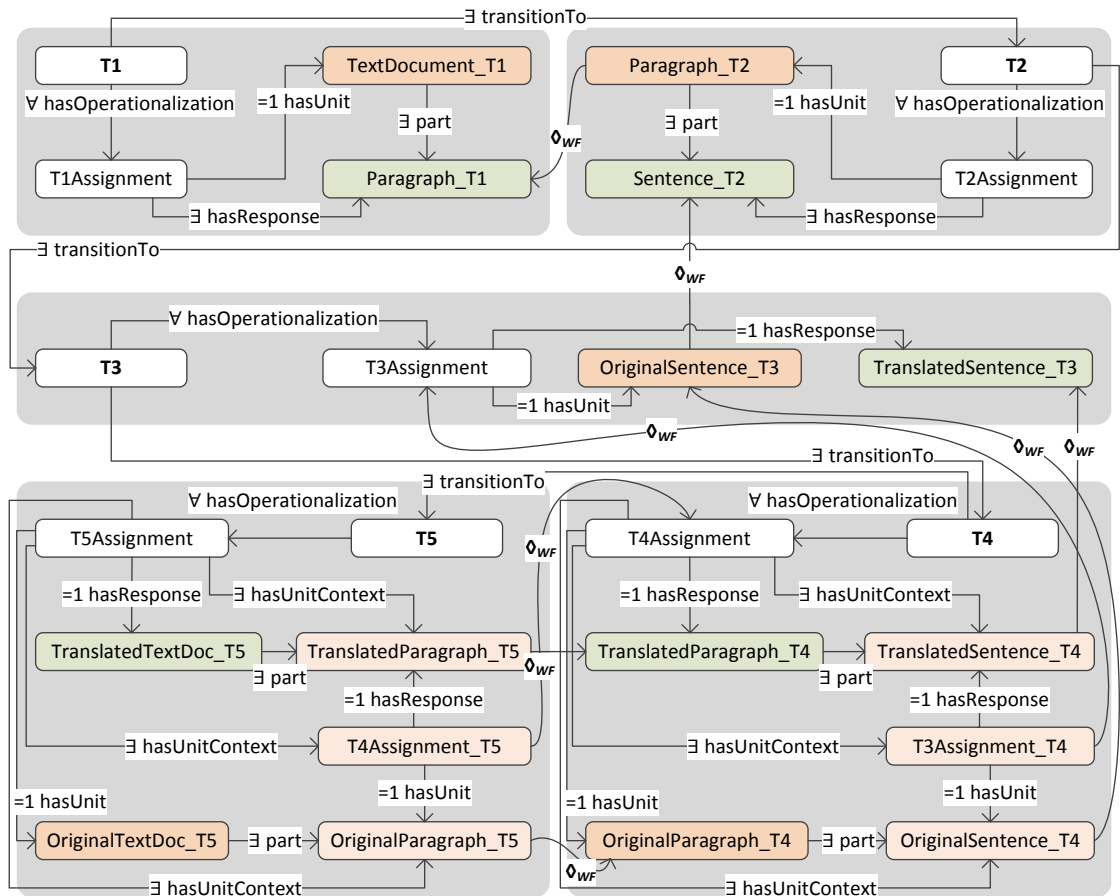


Figure 47: Task-definitions in the document translation workflow-definition.

#	CMD	NEW CONCEPTS	DEPENDENCIES (\diamond_{WF})	IOBox, WORKERS AND TASK INTERFACES
		T1, T1Assignment TextDocument_T1 Paragraph_T1	- - -	T1MachineWorker, T1TaskInterface
1	PW	T2, T2Assignment Paragraph_T2 Sentence_T2 T3, T3Assignment OSentence_T3 TSentence_T3	- Paragraph_T1 - - Sentence_T2 -	TextDocument_T1 $\sqsubseteq \exists hasPart.Paragraph_T1$ T2PersonWorker, T4LocalWebTaskInterface Paragraph_T2 $\sqsubseteq \exists hasPart.Sentence_T2$ T3Worker, T3TaskInterface
2	CTDW	T4, T4Assignment OParagraph_T4 TParagraph_T4	- Paragraph_T1 -	
3	CICT	T3Assignment_T4 OSentence_T4 TSentence_T4	T3Assignment OSentence_T3 TSentence_T3	
4	COCT	-	-	
5	SECT	-	-	
6	CTDW	T5, T5Assignment OriginalTextDoc_T5 TranslatedTextDoc_T5	- T4Assignment OP_T4 TP_T4	T4PersonWorker, T4LocalWebTaskInterface
7	CICT	-	-	
8	COCT	-	-	
9	CICTuc	-	-	
10	CICTuc	-	-	
11	CICTuc	-	-	
12	CRT	-	-	
13	CRT	-	-	
14	SECT	-	-	
15	CTDW	-	-	
16	CICT	-	-	
17	COCT	-	-	
18	CICTuc	-	-	
19	CICTuc	-	-	
20	CICTuc	-	-	
21	CRT	-	-	
22	CRT	-	-	
23	SECT	-	-	

Table 17: Command sequence for the construction of the document translation workflow-definition. UC stands for *UnitContext*, *OSentence* for *OriginalSentence*, *TSentence* for *TranslatedSentence*, *OParagraph* for *OriginalParagraph* and *TParagraph* for *TranslatedParagraph*

8.3.1.2 The Task-Definition UI Templates

The construction of task-definition UI templates is only required for task-definitions with human workers. In this sense, UI templates were built for T2 (exemplified in Figure 48), T3 (exemplified in Figure 49) and T4 (exemplified in Figure 50).

The T2 template asks the worker to split a paragraph into sentences. The amount of sentences may vary for each paragraph. Thus, new sentences must be added as deemed necessary.

The T3 template asks the worker for a translation of a given sentence. The presented instructions are partially taken from the description and label of the task-definition. For instance, the title, which indicates the translation language, is the label of the task-definition.

Finally, the T4 template asks for an assembled paragraph, given a set of previously translated sentences. The worker must validate the translation and submit a refined translated paragraph.

Partition a Paragraph into Sentences

Given the following paragraph:

Paragraph Text

Although several task-oriented CS systems have emerged in a short period of time, the employed approaches are often the same. In fact, a generalization of the CS process (for simple and complex tasks) using a common/mapped terminology is proposed.

Please provide each of its sentences separately and in order:

+ Add Sentence

Sentence #1 Text

Although several task-oriented CS systems have emerged in a short period of time, the employed approaches are often the same.

Sentence #2 Text -

In fact, a generalization of the CS process (for simple and complex tasks) using a common/mapped terminology is proposed.

Submit Assignment

Figure 48: Example assignment with the UI template of T2 in the document translation scenario.

Sentence Translation from English to Portuguese

Given the following sentence in English:

The screenshot shows a web form for translating an English sentence to Portuguese. The form has three main sections: an input field for the English sentence, a prompt for translation, and an output field for the Portuguese sentence. The English sentence is 'Micro-task CS is a research field with a wide variety of application domains.' The output field is currently empty with the placeholder text 'Sentence text'. A blue button labeled 'Submit Assignment' is located at the bottom of the form.

Figure 49: Example assignment with the UI template of T₃ in the document translation scenario.

8.3.1.3 Instantiation and Execution

Through the translation workflow-definition, the paper entitled “A Survey of Task-oriented Crowdsourcing” [44] was translated from English to Portuguese⁴. The partitioning of the paper resulted in 215 paragraphs and 566 sentences.

The execution of the workflow included the participation of 16 different workers (one of them a machine worker using the Google Translator tool). The distribution of workers and assignments throughout each task is presented in Table 18.

TASK	# ASSIGNMENTS	# WORKERS		# ASSIGNMENTS PER WORKER	
		HUMAN	MACHINE	MEAN	STD. DEVIATION
T ₁	1	0	1	1	0
T ₂	215	1	0	215	0
T ₃	566	14	1	37.73	59.43
T ₄	215	9	0	23.89	17.38
T ₅	1	0	1	1	0

Table 18: Distribution of workers and assignments throughout each task in the document translation workflow.

⁴The translated document can be found at http://mobaton.com/nluz/CS_AIR_PT.pdf

Paragraph Translation from English to Portuguese

Given the following paragraph in English and its previously translated sentences:

English Paragraph Text

Fig. 4 - Process diagram of a common (complex) task CS and HC system. Dashed steps may, or may not, exist in different systems.

#	English Sentence	Portuguese Sentence
#1	Fig. 4 - Process diagram of a common (complex) task CS and HC system.	Figura 4 - Diagrama do processo de uma tarefa comum (complexa) de CS e HC system.
#2	Dashed steps may, or may not, exist in different systems.	Etapas rápidas podem, ou não, existir em diferentes sistemas.

Please verify and assemble a translated Portuguese paragraph:

Portuguese Paragraph Text

Figura 4 - Diagrama do processo de uma tarefa comum (complexa) de CS e HC system. Etapas rápidas podem, ou não, existir em diferentes sistemas.

Submit Assignment

Figure 50: Example assignment with the UI template of T₄ in the document translation scenario.

The execution of the translation workflow was performed successfully. Not only was the given input described according to the document ontology, but also the output was retrieved and described according to the document ontology.

However, in this particular scenario, several improvements are required in terms of the provided data and the UI templates. In particular, workers have indicated that T₃ (the sentence translation task) does not provide enough contextual information. Furthermore, given the scientific and specialized nature of the translated text, many terms were translated differently and ambiguously.

In order to assess the overall opinion of the workers for T₃ and T₄, an enquiry was performed.

Regarding T₃, the following set of questions were placed to all of its workers (14) and answered by 6 of them:

- T₃Q₁ - How confident are you in the quality of your translations (in a scale from 1 to 5)?
- T₃Q₂ - How strongly did you feel the need to get more contextual information (in a scale from 1 to 5)?
- T₃Q₃ - What kind of contextual information do you think would be useful (multiple choices)? The possible answers to this question are:
 - A₁ - A glossary of domain-specific terms;
 - A₂ - A fixed set of translations for domain-specific terms;
 - A₃ - Translating sentences is not worth it; paragraphs should be translated directly;
 - A₄ - Other (free input answer).
- T₃Q₄ - Did you use any external translation tool (e. g. Google Translator)?

Figure 51 provides an illustration of the answers given to T₃Q₁, T₃Q₂, T₃Q₃ and T₃Q₄.

Answers to T₃Q₁ suggest that workers were positively confident of their translations. However, answers to T₃Q₂ show that most workers felt the need for additional contextual information. Also, as shown by the answers to T₃Q₄, most workers felt the need to resort to external translation tools.

Answers to T₃Q₃ suggest that much can be done in order to provide additional contextual information.

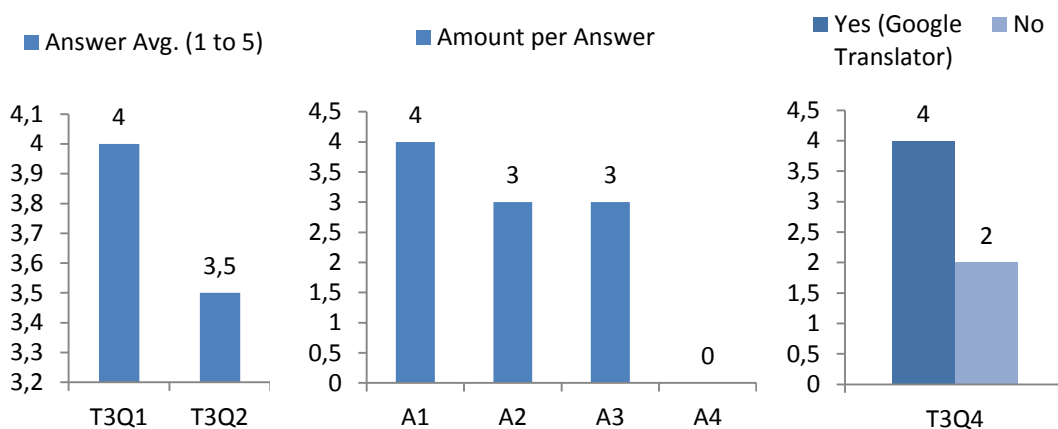


Figure 51: Answers to T₃Q₁ and T₃Q₂ (left), T₃Q₃ (center), and T₃Q₄ (right) in the document translation scenario.

Regarding T₄, the following questions were placed to all of its workers (9) and answered by 5 of them:

- T₄Q₁ - How do you rate the quality of the originally presented paragraph translations (in a scale from 1 to 5)?
- T₄Q₂ - How do you rate the amount of effort spent editing and fixing the paragraph translations (in a scale from 1 to 5)?
- T₄Q₃ - Did you feel the need for more contextual information?
- T₄Q₄ - Did you use any external translation tool (e. g. Google Translator)?

Figure 52 provides an illustration of the answers given to T₄Q₁, T₄Q₂, T₄Q₃ and T₄Q₄.

Answers to T₄Q₁ and T₄Q₂ show that the quality of the translated sentences is mediocre and required some improvement effort. This is expected, since answers to questions regarding T₃ establish a need for more contextual information and support from external translation tools.

Answers to T₄Q₃ show that regardless of providing a broader translation context through paragraphs (instead of sentences), most workers still felt the need for more contextual information. This is likely related to the required technical and scientific background knowledge for the understanding and interpretation of the document. Since there are no prior conditions demanding that workers must possess this knowledge, adding additional background information (e. g. an initial coherent translated terminology, definitions for key concepts) becomes an important requirement.

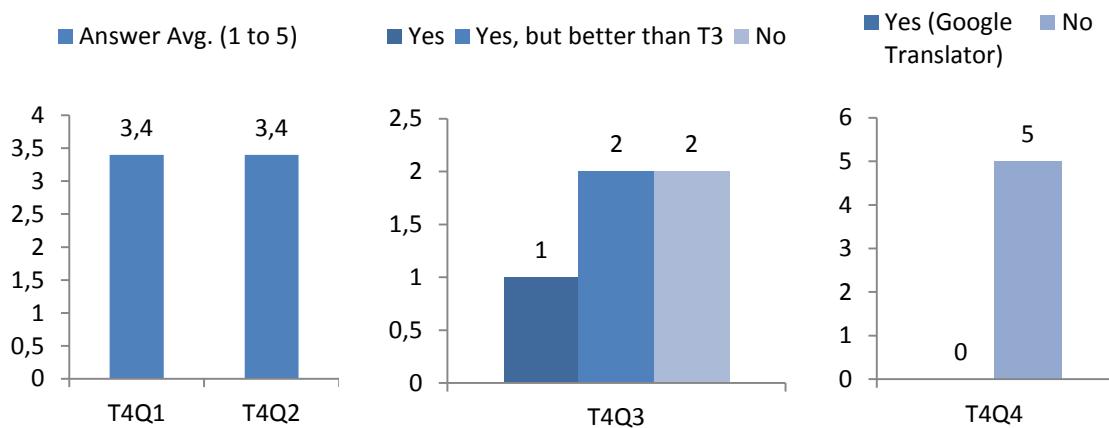


Figure 52: Answers to T₄Q₁ and T₄Q₂ (left), T₄Q₃ (center), and T₄Q₄ (right) in the document translation scenario.

Finally, answers to T4Q4 show that none of the enquired workers resorted to external translation tools. This is also expected, since the inherent operations performed while solving an assignment of T3 are quite different from those performed while solving an assignment of T4. More precisely, while T3 is a typical translation task, T4 is an assembly and refinement task, where the translation work is assumed to be mostly complete.

8.3.2 *The Ontology Alignment and Construction Scenario*

The ontology alignment and construction scenario is not common in crowdsourcing applications. Since it typically involves checking all possible concept and role combinations between two ontologies, a high recall value can be achieved through crowdsourcing. However, the number of assignments easily scales to impractical amounts. For big ontologies, this drawback can be tackled with the aid of machine algorithms that pick likely candidates for a match [62].

In order to build an ontology alignment and construction workflow, the OWL meta-model was used as domain ontology. Through the OWL meta-model ontology, the workflow of tasks presented in Figure 53 was executed for the creation of a new ontology that represents the alignment and refinement of two ontologies, *onto1* and *onto2*:

- T1 - asks if two top-level concepts (or classes) are equivalent or sub-concepts (if one subsumes the other);
- T2 (loop) - asks if two next-level concepts (or classes) are equivalent or sub-concepts (if one subsumes the other);
- T3 - asks if two roles (or properties) are equivalent;
- T4 - asks if two concepts are related through a new role (or property) restriction, i. e. if one is a meronym of the other.

To reduce the amount of possible matches and assignments in this scenario, a divide-and-conquer approach is employed. First, only the top-level concepts in the hierarchy are matched (in T1 with four assignments per unit). From the resulting top-level concept matches, sub-concepts are matched in subsequent tasks, up to the fourth level of the hierarchy (in the T2 loop with four assignments per unit). Finally, the properties of the previously matched concepts are also matched (in T3 with four assignments per unit).

An additional (construction) task with the purpose of establishing new roles and restrictions between concepts of the two ontologies (T4 with one assignment per unit) runs in parallel with T1, T2 and T3.

8.3.2.1 The Workflow-Definition

In order to make the distinction between concepts and roles of *onto1* and those of *onto2*, the concepts *O1Class* and *O2Class* (sub-concepts of *Class*), and *O1Property* and *O2Property* (sub-concepts of *Property*) were created. The concepts *Match* and *PropertyMatch* were also added to the domain ontology in order to represent matches between two concepts and two roles. The additional concepts in the domain *TBox* are presented in Figure 54.

The domain ontology also contains the *RBox* axioms $matchFrom \equiv from^-$ and $matchTo \equiv to^-$, which establish the inverse roles of *from* and *to*.

The workflow-definition for the ontology alignment and construction process was built using the CompFlow construction framework and the Protégé ontology editor (to establish role restriction unions). The resulting workflow-definition is presented in Figure 55 and Figure 56.

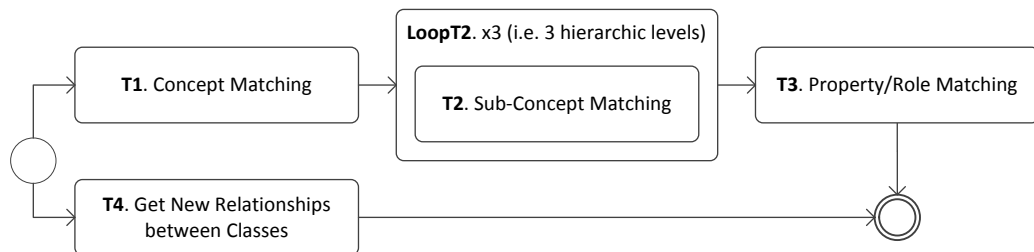


Figure 53: Overview of the ontology alignment and construction workflow-definition.

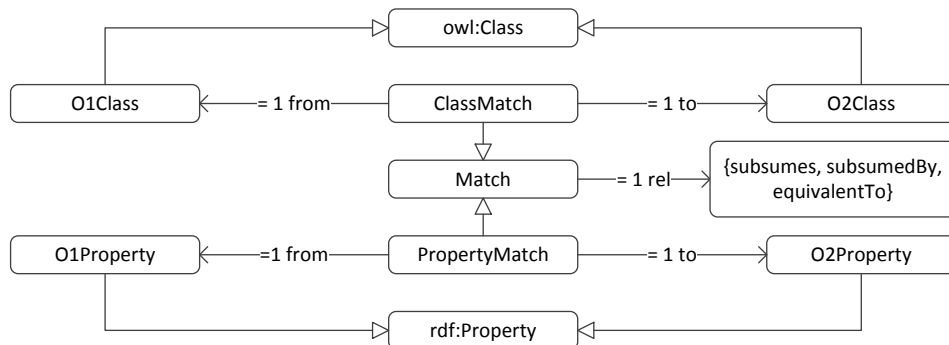


Figure 54: Additional concepts in the ontology alignment and construction domain ontology.

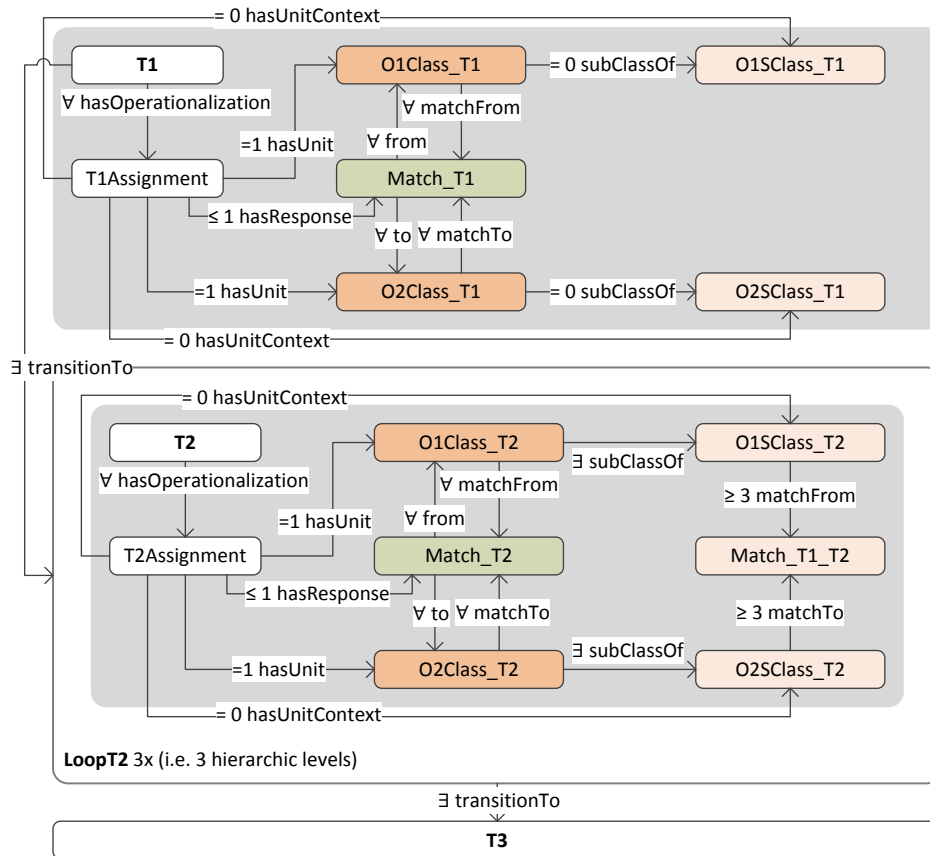


Figure 55: Task-definitions in the ontology alignment and construction workflow-definition (part 1).

T1 provides an assignment for each pair of concepts, $C1$ in *onto1* and $C2$ in *onto2*. The assignment consists in asking the worker to select if two concepts are related through a subsumption or equivalence relationship (i. e. a match). Matching the two concepts is optional, meaning that the worker may opt not to establish a relationship. Also, both $C1$ and $C2$ cannot have atomic super-concepts (i. e. they must be top-level concepts in their hierarchies).

T2 is very similar to T1 since the worker must also establish a match between two concepts, $C1$ in *onto1* and $C2$ in *onto2*. However, these concepts must be sub-concepts of previously matched concepts. Notice that $C1$ and $C2$ must be sub-concepts of previously matched concepts. A match between the two super-concepts is only considered in T2, if at least three (out of four) workers have previously established the match.

T2 is instantiated and executed three times, thus covering possible matches between three hierarchic levels (under the root level) of both input ontologies. The covered hierarchic depth can be easily broadened by increasing the amount of iterations performed by the LoopT2.

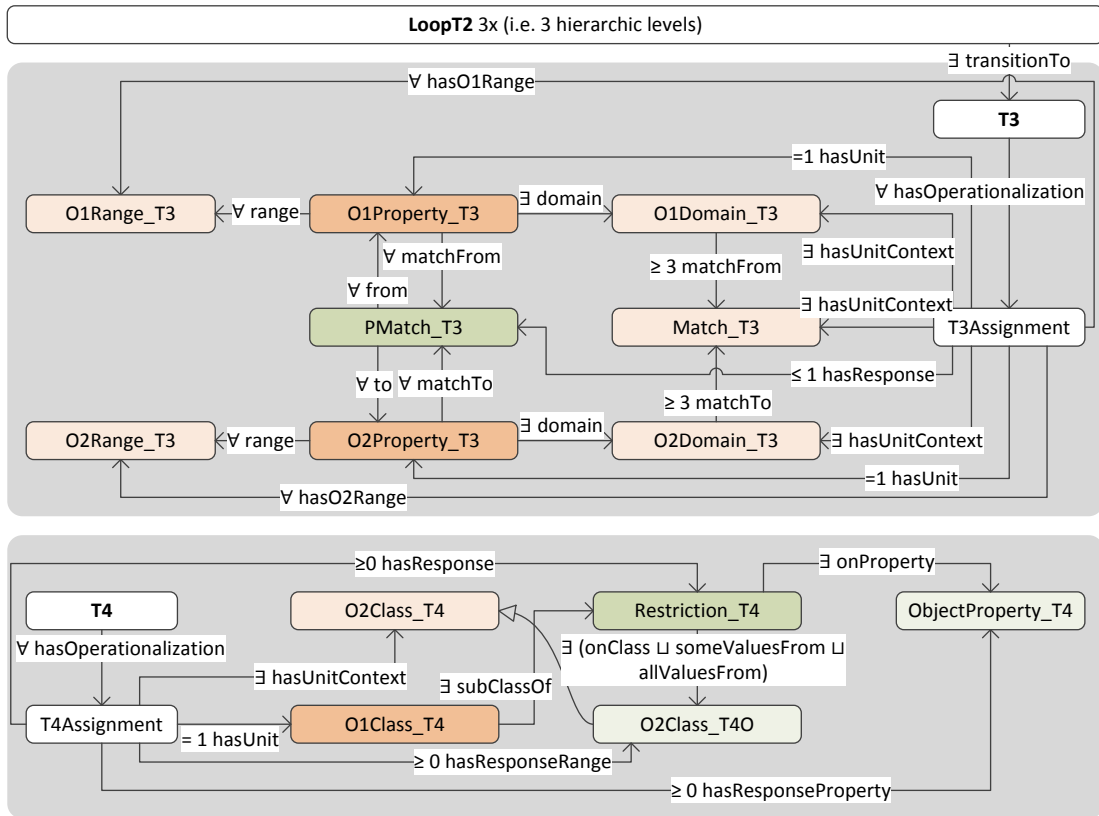


Figure 56: Task-definitions in the ontology alignment and construction workflow-definition (part 2).

In T3, the worker may establish an equivalence relationship between two roles, $R1$ in $onto1$ and $R2$ in $onto2$. Only roles from previously matched concepts (specified during the execution of T1 and the T2 loop) are presented to the worker. This substantially reduces the amount of assignments.

T4 is not typically found in ontology alignment processes. It represents an ontology construction operation rather than an ontology alignment operation. The definition of such a task is possible since the [OWL](#) meta-model is used as domain ontology.

The workflow-definition was built through the execution of the command sequence presented in [Table 19](#) and [Table 20](#).

8.3.2.2 The Task-Definition UI Templates

Each task-definition in the ontology alignment and construction workflow-definition has an [UI](#) template. The [UI](#) template of T1 is depicted in [Figure 57](#). To allow non-expert workers to participate in the matching process, straightforward natural language sentences are presented to workers. Workers must then indicate if they are true or false.

#	CMD	NEW CONCEPTS	DEPENDENCIES (\diamond_{WF} OR \sqsubseteq)	NEW IN TBox, WORKERS AND TASK INTERFACES
1	CTDW	T1, T1Assignment	-	T1Worker, T1TaskInterface
2	CICT	O1Class_T1	-	-
3	CICT	O2Class_T1	-	-
4	COCT	Match_T1	-	Match_T1 $\sqsubseteq \forall from.O1Class_T1$
5	CRT	-	-	Match_T1 $\sqsubseteq \forall to.O2Class_T1$
6	CRT	-	-	O1Class_T1 $\sqsubseteq \forall matchFrom.Match_T1$
7	CRT	-	-	O2Class_T1 $\sqsubseteq \forall matchTo.Match_T1$
8	CICT _{uc}	O1SClass_T1	-	O1Class_T1 $\sqsubseteq = 0subClassOf.O1SClass_T1$
9	CICT _{uc}	O2SClass_T1	-	O2Class_T1 $\sqsubseteq = 0subClassOf.O2SClass_T1$
10	SECT	-	-	-
11	CWDW	LoopT2	-	-
12	CTDW	T2, T2Assignment	-	T2Worker, T2TaskInterface
13	CICT	O1Class_T2	-	-
14	CICT	O2Class_T2	-	-
15	COCT	Match_T2	-	Match_T2 $\sqsubseteq \forall from.O1Class_T2$
16	CRT	-	-	Match_T2 $\sqsubseteq \forall to.O2Class_T2$
17	CRT	-	-	O1Class_T2 $\sqsubseteq \forall matchFrom.Match_T2$
18	CRT	-	-	O2Class_T2 $\sqsubseteq \forall matchTo.Match_T2$
19	CICT _{uc}	O1SClass_T2	-	O1Class_T2 $\sqsubseteq \exists subClassOf.O1SClass_T2$
20	CICT _{uc}	O2SClass_T2	-	O2Class_T2 $\sqsubseteq \exists subClassOf.O2SClass_T2$
21	CICT _{uc}	Match_T1_T2	-	O1SClass_T2 $\sqsubseteq \geq 3matchFrom.Match_T1_T2$
22	CRT	-	-	O2SClass_T2 $\sqsubseteq \geq 3matchTo.Match_T1_T2$
23	SECT	-	-	-
24	SECW	-	-	-

Table 19: Command sequence for the construction of the ontology alignment and construction workflow-definition (part 1). UC stands for *UnitContext* and RC stands for *ResponseContext*.

#	CMD	NEW CONCEPTS	DEPENDENCIES (\diamond WF OR \sqsubseteq)	NEW IN TBox, WORKERS AND TASK INTERFACES
25	CTDW	T3, T3Assignment	-	T3Worker, T3TaskInterface
26	CICT	O1Property_T3	-	-
27	CICT	O2Property_T3	-	-
28	COCT	PMatch_T3	-	PMatch_T3 \sqsubseteq \forall from.O1Property_T3 PMatch_T3 \sqsubseteq \forall to.O2Property_T3
29	CRT	-	-	O1Property_T3 \sqsubseteq \forall matchFrom.PMatch_T3
30	CRT	-	-	O2Property_T3 \sqsubseteq \forall matchTo.PMatch_T3
31	CRT	-	-	O1Property_T3 \sqsubseteq \forall range.O1Range_T3
32	CICTuc	O1Range_T3	-	O2Property_T3 \sqsubseteq \forall range.O2Range_T3
33	CICTuc	O2Range_T3	-	O1Property_T3 \sqsubseteq \exists domain.O1Domain_T3
34	CICTuc	O1Domain_T3	-	O2Property_T3 \sqsubseteq \exists domain.O2Domain_T3
35	CICTuc	O2Domain_T3	-	O1Domain_T3 \sqsubseteq \exists matchFrom.Match_T3 O2Domain_T3 \sqsubseteq \exists \geq 3matchTo.Match_T3
36	CICTuc	Match_T3	-	-
37	CRT	-	-	-
38	SECT	-	-	T4Worker, T4TaskInterface
39	CTDW	T4, T4Assignment	-	-
40	CICT	O1Class_T4	-	-
41	CICTuc	O2Class_T4	-	-
42	COCT	O2Class_T4O	O2Class_T4	O1Class_T4 \sqsubseteq \exists subclassOf.Restriction_T4 Restriction_T4 \sqsubseteq \exists R.O2Class_T4O with R \equiv onClass \sqcup someValuesFrom \sqcup allValuesFrom Restriction_T4 \sqsubseteq \exists onProperty.ObjectProperty_T4
43	COCT _{Rc}	Restriction_T4	-	-
44	CRT	-	-	-
45	COCT _{Rc}	ObjectProperty_T4	-	-
46	SECT	-	-	T1 \sqsubseteq \exists transitionTo.LoopT2
47	CRDW	-	-	-
48	SECR	-	-	LoopT2 \sqsubseteq \exists transitionTo.T3
49	CRDW	-	-	-
50	SECR	-	-	-

Table 20: Command sequence for the construction of the ontology alignment and construction workflow-definition (part 2). UC stands for *UnitContext* and RC stands for *ResponseContext*.

Their truthfulness establishes if a subsumption relationship exists between the two concepts.

Similarly, the UI template of T2 (depicted in Figure 58) asks workers to establish a match through the evaluation of natural language sentences.

The UI template of T3 (depicted in Figure 59) provides a similar interface to that of the UI template of T1. However, instead of concepts, it asks workers to establish equivalences between two roles of *onto1* and *onto2*. Additional information on roles, such as their domain is also presented to the worker.

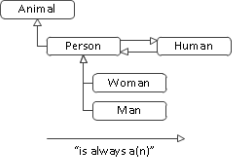
Finally, the UI template of T4 (depicted in Figure 60) allows the worker to submit and add new relationships between the concepts of *onto1* and *onto2*.

8.3.2.3 Instantiation and Execution

The ontology alignment and construction workflow starts with an ABox filled with the concepts and roles of both *onto1* and *onto2*. In this execution, two ontologies from the conference track of the Ontology Alignment Evaluation Initiative (OAEI) were used. These are the Conference Management Toolkit (CMT) ontology (*onto1*) and the Conference ontology (*onto2*).

The initial ABox was built by merging both ontologies and classifying their concepts and roles according to *O1Class*, *O2Class*, *O1Property* and *O2Property*.

Task 1: Select the relationships between the two entities



```

graph TD
    Animal --> Person
    Animal --> Human
    Animal --> Woman
    Animal --> Man
    Person --> Woman
    Person --> Man
    Human --> Person
  
```

"is always a(n)"

Example

The entities Woman, Man, Person, Human and Animal are related as follows:

- A "Woman" is always a "Person"
- A "Man" is always a "Person"
- A "Person" is always an "Animal"
- A "Human" is always a "Person" and a "Person" is always a "Human". Thus, they are equivalent or synonyms.

Taking into account the previous example, please **select if the following sentences are true or false**:

- Click on top of the sentences to switch from true to false and vice-versa
- All entities are involved in scientific or business conferences (for example, the entity "Person" always refers to a person involved in a conference)
- Relationships between the same entities are always true (for example, the sentence, "Person" is always a "Person", is true)
- Only mark a sentence as true if you are certain of its truthfulness

A(n) Person is always a(n) Committee	False
A(n) Committee is always a(n) Person	False

Submit Assignment

Figure 57: Assignment with the UI template of T1 in the ontology alignment and construction scenario.

Task 2.1: Select the relationships between the two entities

Example

The entities Woman, Man, Person, Human and Animal are related as follows:

- A "Woman" is always a "Person"
- A "Man" is always a "Person"
- A "Person" is always an "Animal"
- A "Human" is always a "Person" and a "Person" is always a "Human". Thus, they are equivalent or synonyms.

Taking into account the previous example, please **select if the following sentences are true or false**:

- Click on top of the sentences to switch from true to false and vice-versa
- **All entities are involved in scientific or business conferences** (for example, the entity "Person" always refers to a person involved in a conference)
- **Relationships between the same entities are always true** (for example, the sentence, "Person" is always a "Person", is true)
- Only mark a sentence as true if you are certain of its truthfulness

A(n) ConferenceMember is always a(n) Conference_contributor

False

A(n) Conference_contributor is always a(n) ConferenceMember

True

Things that are a(n) ConferenceMember:

- **Author**
- **Reviewer**

ConferenceMember is always a(n):

- **Person**

Things that are a(n) Conference_contributor:

- **Invited_speaker**
- **Regular_author**

Conference_contributor is always a(n):

- **Person**

Submit Assignment

Figure 58: Assignment with the UI template of T2 in the ontology alignment and construction scenario.

Task 3: Select if the two properties are equivalent or the same

Example

Properties are used to describe entities. For example, the properties hasAttribute and hasCharacteristic may describe a Person, while the properties hasCarPart and hasCarEngine may describe a Car.

For a Person to have an attribute or to have a characteristic is the same or equivalent. Thus, "hasAttribute" is the same as or equivalent to "hasCharacteristic".

Taking into account the previous example, please **select if the following sentence is true or false**:

- Click on top of the sentence to switch from true to false and vice-versa
- **All entities and properties are involved in scientific or business conferences**
- **Relationships between the same properties are always true** (for example, "hasName" is the same as or equivalent to "hasName")
- Only mark the sentence as true if you are certain of its truthfulness

The property hasConflictOfInterest, describing a(n) Person, is the same as or equivalent to contributes, describing a(n) Person

False

Submit Assignment

Figure 59: Assignment with the UI template of T3 in the ontology alignment and construction scenario.

Specify a relationship between two concepts

Please indicate if there is a relationship between Co-author and any of the following given concepts:

- There may not be any relationship between these concepts
- Ignore the relationships: "is a(n)", "equivalent to" and "type of"
- Focus on [relationships of meronymy/holonymy](#)

Figure 60: Assignment with the UI template of T4 in the ontology alignment and construction scenario.

The execution of the workflow counted with 16 different workers. The distribution of workers and assignments throughout each task is presented in [Table 21](#).

The workflow was executed successfully and resulted in a new ontology that merged both the [CMT](#) and the Conference ontologies. The new ontology not only represents an alignment, but also a refinement with new roles and relationships between the concepts of both ontologies.

TASK	# ASSIGNMENTS	# WORKERS		# ASSIGNMENTS PER WORKER	
		HUMAN	MACHINE	MEAN	STD. DEVIATION
T1	448	8	0	56	43.71
T2.1	176	7	0	25.14	12.72
T2.2	72	6	0	12	6.81
T2.3	0	0	0	0	0
T3	144	4	0	36	0
T4	29	5	0	5.8	7.36

Table 21: Distribution of workers and assignments throughout each task in the ontology alignment and construction workflow.

ALIGNMENT TASKS: The resulting assignment, extracted from the output data of T₁, T₂ and T₃, is presented in Table 22. These matches are compared to the matches of the reference alignment⁵ found in Table 23.

Notice that incorrect (or lack of) matches in the initial tasks have a great impact in the following tasks. Thus, the lack of matches in the second iteration of T₂ (T_{2.2}), led to the non-existence of assignments in the third iteration of T₂ (T_{2.3}).

Overall, the root-level concepts were successfully matched according to the reference alignment. However, several improvements are necessary to properly establish matches at the lower hierarchic levels.

One of the drawbacks of this approach is the impossibility of establishing matches for previously matched concepts, and between concepts in different hierarchic levels. For instance, the match, *ProgramCommittee* \sqsubseteq *Committee* was correctly established, since *ProgramCommittee* \equiv *Program_committee* belongs to the reference alignment and *Program_committee* \sqsubseteq *Committee*. However, the proper direct match, found in the ref-

TASK	CMT CONCEPT	MATCH	CONFERENCE CONCEPT
T ₁	SubjectArea	3x \equiv , 1x \sqsubseteq	Topic
	Person	4x \equiv	Person
	Preference	3x \equiv , 1x \sqsubseteq	Review_preference
	Document	3x \equiv , 1x \sqsubseteq	Conference_document
	ProgramCommittee	4x \sqsubseteq	Committee
	Conference	4x \equiv	Conference
T _{2.1}	ExternalReviewer	4x \sqsubseteq	Reviewer
	Paper	3x \sqsubseteq	Conference_contribution
	Chairman	3x \sqsubseteq	Committee_member
	Review	4x \equiv	Review
	ConferenceMember	4x \sqsubseteq	Reviewer
	Chairman	4x \sqsubseteq	Track-workshop_chair
	ConferenceMember	1x \sqsubseteq , \sqsubseteq , \equiv	Conference_participant
	Chairman	3x \sqsubseteq	Conference_participant
ProgramCommitteeMember	4x \sqsubseteq	Committee_member	
T ₃	email	3x \equiv	has_an_email

Table 22: Resulting alignment between the CMT and Conference ontologies in the ontology alignment and construction workflow.

⁵part of the conference track dataset of the OAEI

CMT CONCEPT	MATCH	CONFERENCE CONCEPT	IN RESULT
SubjectArea	≡	Topic	✓
Person	≡	Person	✓
Document	≡	Conference_document	✓
ProgramCommittee	≡	Program_committee	◆
Conference	≡	Conference	✓
Conference	≡	Conference_volume	✗
Chairman	≡	Chair	◆
Review	≡	Review	✓
Preference	≡	Review_preference	✓
Author	≡	Regular_author	✗
Co-author	≡	Contribution_co-author	✗
PaperAbstract	≡	Abstract	✗
email	≡	has_an_email	✓
assignedByReviewer	≡	invited_by	✗
assignExternalReviewer	≡	invites_co-reviewers	✗

Table 23: Reference alignment between the CMT and Conference ontologies in the ontology alignment and construction workflow. ◆ means that an incomplete or partial match was established.

erence alignment, is never formulated as an assignment because concepts of different hierarchic levels are never considered.

Abstract concepts, such as *Conference_contributor*, *Conference_participant* and *ConferenceMember*, were particularly hard to match. For instance, most of the workers were sure that a match between *ConferenceMember* and *Conference_participant* exists, however, they did not agree on the type of match between the two concepts.

Other matches that should have been established, such as between *ConferenceMember* and *Conference_contributor* were not established by the majority. This led to the exclusion of all of their possible lower-level matches, which included some of those present in the reference alignment.

CONSTRUCTION TASK: As presented in Table 24, the construction task, T4, led to 39 new role restrictions established between the concepts of the CMT ontology, and the concepts of the Conference ontology. These new role restrictions were not submitted to any verification or validation process.

Although the majority of the role restrictions are already represented in at least one of the ontologies, the results were properly structured according to the [OWL](#) meta-model, demonstrating that ontology construction tasks can be modelled through the [CompFlow](#) workflow-definition method.

8.3.2.4 *Alternative Alignment Workflow-Definitions*

There are several ways available to model an ontology alignment workflow. For instance, these may be based on different domain ontologies such as the Expressive and Declarative Ontology Alignment Language ([EDOAL](#)) [63] or the Semantic Bridge Ontology ([SBO](#)) [46]. Also, a pattern-based approach can be followed instead of the traditional concept-to-concept or role-to-role matching approach [60, 64, 65].

The [SBO](#), in particular, already defines several complex ontology alignment patterns, represented by semantic bridges (concept and property bridges). For instance, in the workflow-definition depicted in [Figure 55](#), T1 could ask workers to submit new concept bridges between two concepts, while T3 would establish the related property bridges for the previously submitted concept bridges. Furthermore, the expressiveness of the [SBO](#) allows workers to establish mappings that are not possible to establish through the [OWL](#) meta-model (e. g. mappings that need the transformation and manipulation of datatype values).

The [EDOAL](#) is another alternative to the construction of ontology alignment workflow-definitions. Following a very similar approach to the one presented in the previous alignment and construction workflow-definition, the [EDOAL](#) establishes mappings through equivalence and subsumption relationships between concepts and roles. Since this format is used by automatic alignment approaches, each mapping is typically accompanied by a confidence value.

In order to use these ontologies to represent the static domain dimension of workflow-definitions they must be built as a [DL](#) ontology artefact.

8.3.3 *The Catalan Constitution Refinement Scenario*

The Catalan constitution refinement scenario aims to take an initial ontology-based dataset containing the Catalan constitution and crowdsource its refinement effort. The construction of this scenario is an ongoing joint effort with Marta Poblet from the Royal Melbourne Institute of Technology ([RMIT](#)) University and the Universitat Autònoma de

CMT CONCEPT	ROLE RESTRICTION TO CONFERENCE CONCEPT (\sqsubseteq)
<i>Reviewer</i>	\exists <i>hasReviewed.Paper</i> \exists <i>hasExpertise.Review_expertise</i> \exists <i>hasReviewed.Conference_document</i> ≥ 1 <i>hasReviewed.Conference_document</i> ≥ 1 <i>hasReviewed.Paper</i>
<i>PaperFullVersion</i>	≥ 1 <i>has.Topic</i> \exists <i>isContained.Conference_proceedings</i> \exists <i>has.Review</i> ≥ 1 <i>has.Contribution_1th-author</i> $= 1$ <i>contains.Abstract</i>
<i>ProgramCommittee</i>	≥ 1 <i>reviewsPapersFrom.Track</i> ≤ 1 <i>belongsTo.Organizing_committee</i> ≥ 1 <i>acceptedA.Camera_ready_contribution</i> ≥ 1 <i>makes.Review</i> ≥ 1 <i>reviewed.Reviewed_contribution</i> ≥ 1 <i>hasReviewPreference.Review_preference</i> $= 1$ <i>reviewsA.Conference_contribution</i> \exists <i>reviewed.Extended_abstract</i> ≤ 1 <i>gives.Invited_talk</i> ≥ 1 <i>accepted.Accepted_contribution</i> ≥ 1 <i>hasPreference.Topic</i> $= 1$ <i>belongsTo.Committee</i> $= 1$ <i>fromConference.Conference</i> ≥ 1 <i>issuedA.Call_for_participation</i> ≥ 1 <i>reviews.Paper</i> ≥ 1 <i>hasReviewExpertise.Review_expertise</i> ≤ 1 <i>belongsTo.Organization</i> $= 1$ <i>issuedA.Call_for_paper</i> ≤ 1 <i>makes.Presentation</i>
<i>Meta-Review</i>	\exists <i>contains.Review</i>
<i>ConferenceMember</i>	\exists <i>memberOf.Conference</i>
<i>Paper</i>	$= 1$ <i>contains.Abstract</i>
<i>Person</i>	\exists <i>memberOf.Organization</i> \exists <i>memberOf.Committee</i> \exists <i>partOf.Conference</i>
<i>ProgramCommitteeChair</i>	\exists <i>belongsTo.Program_committee</i>
<i>Decision</i>	$= 1$ <i>givenTo.Paper</i> \exists <i>basedOn.Review</i>
<i>Document</i>	$= 1$ <i>Contains.Abstract</i>

Table 24: T4 results in the ontology alignment and construction workflow.

Barcelona. In this scenario, the Constitute project ontology⁶ is used as the static domain dimension. The Constitute project⁷ provides ontology-based datasets with the constitutions of several countries according to a single ontology. The resulting refinement process, as depicted in Figure 61, contemplates the following tasks, which are all performed by human workers:

- T1 - evaluates sections of the current constitution document, and is performed by any human worker;
- T2 - revises and updates sections of the current constitution document marked in the previous task, and is performed by expert workers;
- T3 - selects the best version of a section from the set of proposed sections in the previous task, and is performed by any human worker.

The Constitute project ontology represents the constitution document through sections. A partial TBox of the Constitute project ontology is presented in Figure 62. An additional evaluation TBox was added to the static domain dimension in order to represent the opinion and the assessment of the constitution sections.

8.3.3.1 The Workflow-Definition

The constitution refinement workflow-definition was built using both the CompFlow construction framework prototype implementation and the Protégé ontology editor. The Protégé ontology editor was used to establish some common axioms that are not yet featured by the construction framework, such as the union of input and output con-

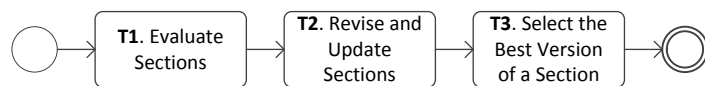


Figure 61: Overview of the Catalan constitution refinement process.

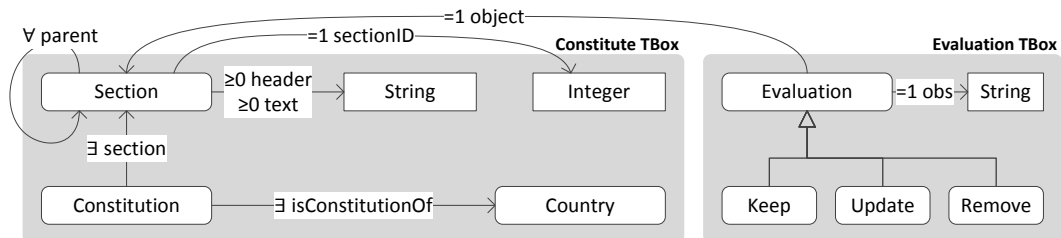


Figure 62: Partial TBox of the Constitute project ontology.

⁶<https://www.constituteproject.org/ontology>

⁷<https://www.constituteproject.org/>

cepts, and inverse roles. A detailed illustration of the workflow-definition is presented in Figure 63.

Notice that in T2, X represents the amount of evaluations that request an update of a section. Thus, for any section to be possibly considered a unit of T2, the amount of assignments per unit (au) of T1 must be greater or equal to X . Otherwise, there will never be enough evaluations, and T2 will not have assignments since the role restriction is never satisfied.

The use of the role transitive closure onto the *parent* role allows all descendant sections of the unit section to be included in the assignment and shown to the worker. Also, regular expressions may be used to restrict the value of datatype roles. Such is the case of the value of the *header* role in T1 ($Section_T1 \sqsubseteq header : "/^Article/"$).

T3 has the particularity of establishing the concept Y , which represents the union of $Section_T3$ and $ProposedSection_T3$. Since the response concept is dependent on Y , the worker will have to select one section from the set of sections that are either a $Section_T3$ or a $ProposedSection_T3$. That is, the original constitution section will remain eligible as the best section.

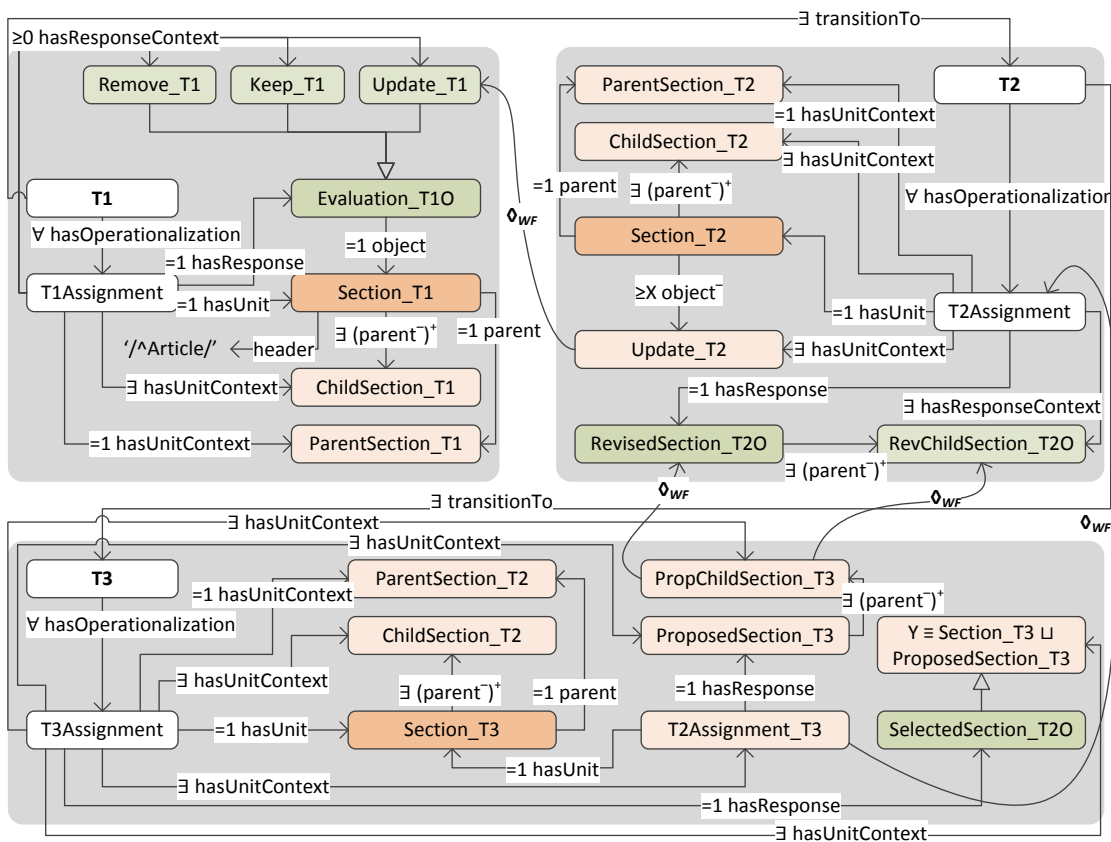


Figure 63: Task-definitions in the constitution refinement workflow-definition.

8.3.3.2 The Task-Definition UI Templates

Since they are solved by human workers, all the task-definitions in the Catalan constitution refinement workflow-definition must include an UI template.

The UI template of T₁, as depicted in Figure 64, presents the unit section, its parent section, and all its descendant sections to the worker. The worker is then asked to evaluate the contents of the section and assess if the section needs to be updated, removed or does not need any modifications.

The UI template of T₂ presents the unit section in the same way as T₁ (see Figure 65 for an illustration). Additionally, the observations given by workers during T₁ are also presented. The expert worker is then asked to submit a new revised section that takes into account the observation given during T₁.

Finally, the UI template of T₃, as depicted by Figure 66, presents each of the previously submitted sections (during T₂), along with the original section, and asks the worker to select the best version.

Evaluate Sections of the Catalan Constitution

Given the following section of the Catalan constitution:

CHAPTER 2: Drafting of Bills

Article 83

The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.

Please read and evaluate if the section requires modifications. If so, explain where and why:

- If you pick "Update" or "Remove", be as detailed and clear as possible in your suggestions
- Select "Remove" only if you mean that the whole section should be removed from the constitution. For any other changes select "Update"

Evaluation of Article 83

A new list item should be added with...

Submit Assignment

Figure 64: Assignment with the UI template of T₁ in the Catalan constitution refinement scenario.

Revise and Update Sections of the Catalan Constitution

Given the following section of the Catalan constitution:

CHAPTER 2: Drafting of Bills

Article 83
The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.

Update Suggestions

A new list item should be added with...

Please read and revise the section according to the given suggestions:

- The whole section must be transcribed (done automatically) and updated in the following text box
- Each line corresponds to a sub-section
- Use **sets of two spaces at the beginning of each line to provide hierarchy**
- Root level sub-sections have no spaces at the beginning of the line, first level sub-sections have two, second level four and so on
- Ordered lists should be styled with a) ... b) ... c) ... and so on
- Unordered lists should be styled with - ... - ... - ... and so on

Revision of Article 83

The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.
- c) Lorem ipsum ...

Example

This is a root level sub-section.
The following is an ordered list:

- a) List items are sub-sections of the sub-section that introduces the list;
- b) This is another list item.

Going back to the root level.

Unordered lists are styled as follows:

- This is a list item.
- This is another list item.

Submit Assignment

Figure 65: Assignment with the UI template of T2 in the Catalan constitution refinement scenario.

Pick the Best Version of a Section of the Catalan Constitution

Given the following section of the Catalan constitution:

CHAPTER 2: Drafting of Bills

Article 83
The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.

Please read and pick the best version of this section from the following list:

Article 83
The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.

Article 83
The acts of basic principles may in no case:

- a) Authorize the modification of the act itself.
- b) Grant power to enact retroactive regulations.
- c) Lorem ipsum...

Submit Assignment

Figure 66: Assignment with the UI template of T3 in the Catalan constitution refinement scenario.

8.3.3.3 Instantiation and Execution

The Catalan constitution refinement scenario is still an ongoing work. Thus, and since the execution of the workflow has not yet been performed, no instantiation results are presented.

8.3.4 Integration with External Projects

The core implementation of the CompFlow instantiation and execution engine was part of a joint effort with Carlos Pereira from the University of Aveiro [41]. The engine is being used in the context of the Dynamic Evaluation as a Service (DynEaaS) [56] evaluation platform, which is “capable of evaluating user performances in dynamic environments by allowing evaluation teams to create and conduct context-aware evaluations”. The instantiation and execution engine is used to run evaluation workflows, and distribute and retrieve tasks through several user interfaces.

8.4 SUMMARY

The construction framework and engine implementations show that practical applications of the proposed CompFlow workflow-definition method and assisted construction process are possible. Furthermore, the described implementations provide a flexible and extensible [API](#) and are oriented toward distributed architectures where humans and machines alike can fulfil the role of workers.

The implementation of the command architecture and the suggestion of commands in the CompFlow construction framework reduces the amount of steps that the creator must perform in order to build workflow-definitions. In particular, the potential growth of the strategic and the pattern command layers suggests that the construction of workflow-definitions can be further automated, while retaining all the choices that provide control to the creator.

The ability to add an unrestricted amount of different types of interfaces to the CompFlow engine yields several application scenarios where:

- Web services are considered as machine workers;
- External [CS](#) platforms (e. g. CrowdFlower) are used to dispatch tasks;
- External worker communities (e. g. Facebook) are used to solve tasks;
- Multi-modal interfaces exist.

CONCLUSIONS

This thesis presents an ontology-based approach to representing, building, instantiating, and executing workflows of tasks, capable of harnessing the full domain static and dynamic semantics of the workflows. This approach is entitled, CompFlow.

The CompFlow approach considers two steps (see [Chapter 3](#)): (i) the representation of the workflow and construction of the workflow-definition, and (ii) the instantiation and execution of the workflow-definition.

The first step (i) is tackled through (a) the definition structures and processes found in the workflow-definition method (see [Chapter 5](#)), and through (b) the assisted construction process (see [Chapter 7](#)). The workflow-definition method (a) establishes a formal method of building workflow representations. Rooted on the workflow-definition method, an assisted construction process (b) is proposed. This process reduces the amount of steps and expertise required to build workflow-definitions.

The second step (ii) is tackled through the instantiation and execution structures and processes found in the workflow-definition method (see [Chapter 6](#)).

In order to evaluate the proposed approach the CompFlow assisted construction framework, and the CompFlow instantiation and execution engine have been implemented (see [Chapter 8](#)). The CompFlow assisted construction framework features the representation of workflows and the assisted construction of workflow-definitions. The CompFlow instantiation and execution engine features the instantiation and execution of workflow-definitions. Three different scenarios that rely on these implementations have been presented, demonstrating that practical applications of the CompFlow approach are possible.

In this chapter, it is argued that this thesis answers the research questions presented in [Chapter 1](#). Furthermore, an overview of the contributions is presented along with a discussion of the future directions and work.

9.1 CONTRIBUTIONS

The overarching goal of this thesis is to investigate if:

Workflows of micro-tasks can be represented and semi-automatically built through a formal and seamless full integration of semantically-enriched dynamic and static domain dimensions, in such a way that they are interpretable and executable by both human and machine actors in a human-machine execution environment.

This goal raises several research questions and establishes multiple requirements.

RESEARCH QUESTION NO.1: *Can the structure and semantics of micro-tasks and micro-task workflows be fully represented in terms of their dynamic and static domain dimensions?*

It has been demonstrated by the CompFlow workflow-definition method that the operations and data involved in a micro-task can be modelled and fully represented through DL ontologies in terms of their dynamic and static domain dimensions [42, 41]. Consequently, the specific implementation of the dynamic and procedural dimension of each task through programming languages is not required.

The workflow-definition method, and in particular the CompFlow ontology, absorbs concepts and lessons learnt by traditional workflow and business process approaches to represent different types of activities and transitions.

Although the CompFlow workflow-definition method establishes relationships not present in the DL language, these relationships can be reduced to DL subsumptions and equivalences, providing a unified and a full DL representation of the workflow-definition.

The automatic instantiation and execution of workflow-definitions is also possible through the proposed set of algorithms.

RESEARCH QUESTION NO.2: *Is there a way to incrementally build the micro-task workflow dynamic dimension based on the static domain dimension?*

The proposed assisted construction process makes it possible to incrementally build workflow-definitions based on domain ontologies [43]. This process harnesses the in-

creasing amount of domain ontologies present in the Semantic Web and re-uses their semantics in establishing the workflow-definition.

The command-based architecture allows the creator of the workflow-definition to increasingly automate its construction while retaining the control provided by the atomic commands. It also provides a highly extensible approach that can be adapted to multiple domains of knowledge and construction methodologies.

RESEARCH QUESTION NO.3: *Can a workflow of micro-tasks, and the involved actors and interfaces, be represented through an unique and platform-independent representation mechanism, understandable and interpretable to both humans and machines?*

The enriched structure and semantics of workflow-definitions built using the CompFlow method provides a detailed representation of the involved data and the operations that must be performed by workers. Also, its formal core based in DL establishes a representation and semantics that are both interpretable by machines and close to the human conceptual level. These aspects of CompFlow workflow-definitions provide a better integration of the operations performed by both humans and machines in a seamless execution environment.

Overall, the requirements for this research question are fulfilled, since an unique and platform-independent representation of workflows is proposed, which features:

- The expressive representation of the workers involved in micro-tasks through concepts and roles (i. e. types of workers);
- The similarly expressive representation of the interfaces involved in each micro-task through concepts and roles.

Following the CompFlow method, a construction, instantiation and execution environment for workflows has been implemented. The implementation includes a construction framework that employs the presented construction process in order to incrementally build workflow-definitions. A workflow-definition instantiation and execution engine is also included in the implementation. It interprets a workflow-definition, creating new instances of the workflow-definition (or workflows) and executing them on top of a domain knowledge base.

The CompFlow method and construction process, along with the presented implementation, can be used to build and execute workflows in multiple application domains and scenarios [39, 40]. Its application can be oriented towards crowdsourcing, towards internal organizational workflows or even both.

9.2 LIMITATIONS

There are several limitations to the current CompFlow approach, which are mainly related to the following aspects:

- *Lack of features often found in traditional workflow or business process approaches.* For instance, the timed execution of activities (i. e. the execution with timeouts);
- *Required expertise in order to build workflow-definitions.* Although the construction process aids the creator in establishing workflow-definitions, the complexity of the process remains complex compared to simple and single-task crowdsourcing platforms, and still requires some degree of expertise in ontology engineering;
- *Expressiveness of the DL language.* The great expressiveness of domain ontologies and of the wide set of available DL languages also demand for the continuous revision and evolution of the CompFlow workflow-definition method. This will allow its expressiveness to grow alongside the developments and expressiveness of DL ontologies. Thus, the expressiveness of the workflow-definition method is currently limited in terms of the base DL language;
- *Complexity and scalability of the proposed algorithms.* The analysis of domain ontologies often requires algorithms with high computational complexity. The current implementation does not scale well when domain ontologies of significant size are used.

Since the expressiveness of the DL language is still to be fully exploited, several aspects can be added or improved in the workflow-definition method. These include:

- The proper distinction, inside loops, of unit individuals that have been assigned in previous tasks;
- Establishing restrictions that match role values of two different concepts (e. g. the role value of $R1$ for the unit concept $C1$ must be the same as the role value of $R2$ for the unit context concept $C2$);
- Supporting negations (this is mostly an implementation issue since negations are already well established in DL languages);
- Distinguishing between individuals with different namespaces (such a feature would be useful, for instance, in the ontology alignment scenario in order to distinguish between the concepts and roles of the two ontologies).

Other aspects that are currently limited and should be further explored are:

- The use of assignment sub-concepts to represent output data (e. g. through data-type roles);
- Adding other required commands to the construction framework and creating a command-based construction language;
- Creating new strategies that harness multiple construction patterns.

9.3 FUTURE WORK

Besides the continuous evolution of the proposed method and implementation, this research opens several paths that are analogous to the general research directions of crowdsourcing platforms and applications and traditional workflow approaches.

The CompFlow method requires continuous research in the expressiveness of workflow-definitions. Furthermore, new types of transitions, events and tasks can be added to the CompFlow ontology. In particular, new types of filter tasks that employ different filter strategies are necessary.

The conditions applied onto *Loop* workflow-definitions and conditional transitions also need to be formalized. Currently, the representation of these conditions depends exclusively on the implementation of the instantiation and execution engine.

Further research in ontology patterns and construction methods may reveal multiple patterns that can be used in the construction of workflow-definitions. In this sense, new pattern commands and strategies need to be established.

Further research directions not explored in this work are also present. These include:

- The automatic or semi-automatic generation of UI templates through the analysis of the workflow-definition ontology;
- The application of quality control mechanisms and their representation in the CompFlow ontology;
- The generation of reports with global and quality indicators;
- A distributed architecture for instantiation and execution engines.

Finally, a refined implementation of the instantiation and execution engine, and of the construction framework is required. In particular, the construction framework requires significant work in usability and in the visual representation of the workflow-definition. A friendly user interface is needed in order to further aid the creator in the construction of workflow-definitions.

Also of particular interest, is to feature a distributed architecture where multiple CompFlow instantiation and execution engines can be deployed with the ability to connect to multiple remote triple stores (operational ABoxes). This steers the implementation towards fitting the vision of the Semantic Web and of Linked Open Data.

BIBLIOGRAPHY

- [1] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 53–64, Santa Barbara, CA, USA, 2011. URL <http://dl.acm.org/citation.cfm?id=2047203>.
- [2] Bernd Amann, Catriel Beeri, Irini Fundulaki, and Michel Scholl. Ontology-based integration of XML web resources. In *The Semantic Web-ISWC 2002*, pages 117–131. Springer, 2002. URL http://link.springer.com/chapter/10.1007/3-540-48005-6_11.
- [3] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2 edition, 2007. ISBN 9780521781763.
- [4] Eva Blomqvist. OntoCase - a pattern-based ontology construction approach. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, number 4803 in LNCS, pages 971–988. Springer, 2007. ISBN 978-3-540-76846-3, 978-3-540-76848-7. URL http://link.springer.com/chapter/10.1007/978-3-540-76848-7_64.
- [5] Daren C. Brabham. Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the international journal of research into new media technologies*, 14(1):75–90, 2008. URL <http://con.sagepub.com/content/14/1/75.short>.
- [6] Daren C. Brabham. Moving the crowd at iStockphoto: The composition of the crowd and motivations for participation in a crowdsourcing application. *First Monday*, 13(6):1–22, 2008. URL <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/viewArticle/2159>.
- [7] Liliana Cabral, Barry Norton, and John Domingue. The business process modelling ontology. In *Proceedings of the 4th international workshop on semantic business process management*, pages 9–16. ACM, 2009. URL <http://dl.acm.org/citation.cfm?id=1944971>.

- [8] Timothy Chklovski. Learner: A system for acquiring commonsense knowledge by analogy. In *Proceedings of the 2nd ACM International Conference on Knowledge Capture*, pages 4–12, Sanibel Island, FL, USA, 2003. URL <http://dl.acm.org/citation.cfm?id=945650>.
- [9] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, and Foldit Players. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307): 756–760, August 2010. ISSN 0028-0836. doi: 10.1038/nature09304. URL <http://www.nature.com/nature/journal/v466/n7307/full/nature09304.html>.
- [10] Chiara Di Francescomarino, Chiara Ghidini, Marco Rospocher, Luciano Serafini, and Paolo Tonella. Reasoning on semantically annotated processes. In *Service-Oriented Computing—ICSOC 2008*, pages 132–146. Springer Berlin Heidelberg, 2008.
- [11] Anhai Doan, Raghu Ramakrishnan, and Alon Y. Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011. URL <http://dl.acm.org/citation.cfm?id=1924442>.
- [12] Marlon Dumas and Arthur HM Ter Hofstede. UML activity diagrams as a workflow specification language. In *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, pages 76–90. Springer, 2001. URL http://link.springer.com/chapter/10.1007/3-540-45441-1_7.
- [13] Marlon Dumas, Wil M. Van der Aalst, and Arthur H. Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005. URL <http://www.google.com/books?hl=pt-PT&lr=&id=ZENNdQq8p74C&oi=fnd&pg=PR7&dq=process+aware+information+systems&ots=ZhQK4PMRfK&sig=tA60QS-mBJKINo9RzT4sAjjVH9A>.
- [14] Patrick Th Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2): 114–131, 2003. URL <http://dl.acm.org/citation.cfm?id=857078>.
- [15] Siamak Faridani, Björn Hartmann, and Panagiotis G. Ipeirotis. What’s the right price? pricing tasks for finishing on time. In *Proceedings of AAAI Workshop on Human Computation*, 2011. URL <http://www.aaai.org/ocs/index.php/WS/AAAIW11/paper/download/3994/4269>.
- [16] Aldo Gangemi. Ontology design patterns for semantic web content. In *The Semantic Web - ISWC 2005*, number 3729 in LNCS, pages 262–276. Springer, 2005. URL http://link.springer.com/chapter/10.1007/11574620_21.

- [17] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and parallel Databases*, 3(2):119–153, 1995. URL <http://link.springer.com/article/10.1007/BF01277643>.
- [18] Michael F. Goodchild and J. Alan Glennon. Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3(3):231–241, 2010. URL <http://www.tandfonline.com/doi/abs/10.1080/17538941003759255>.
- [19] Tom Gruber. Collective knowledge systems: Where the social web meets the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):4–13, 2008. URL <http://www.sciencedirect.com/science/article/pii/S1570826807000583>.
- [20] Karl Hammar and Kurt Sandkuhl. The state of ontology pattern research: A systematic review of ISWC, ESWC and ASWC 2005-2009. In *Workshop on Ontology Patterns: Papers and Patterns from the ISWC Workshop*, pages 5–17, Shangai, China, 2010. URL <http://hj.diva-portal.org/smash/record.jsf?pid=diva2:370448>.
- [21] Christopher G. Harris. Dirty deeds done dirt cheap: A darker side to crowdsourcing. In *IEEE International Conference on Privacy, Security, Risk and Trust*, pages 1314–1317, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6113302.
- [22] Thomas Hornung, Agnes Koschmider, and Jan Mendling. Integration of heterogeneous BPM schemas: The case of XPD and BPEL. In *CAiSE Forum*, volume 231, 2006. URL <http://ww.w.mendling.com/publications/TR-Caise06.pdf>.
- [23] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006. URL http://sistemas-humano-computacionais.wikidot.com/local--files/capitulo: redes-sociais/Howe_The_Rise_of_Crowdsourcing.pdf.
- [24] Jeff Howe. *Crowdsourcing: Why the Power of the Crowd is Driving the Future of Business*. Century, 2008.
- [25] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 64–67, 2010. URL <http://dl.acm.org/citation.cfm?id=1837906>.

- [26] Aniket Kittur, Ed H. Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 453–456, 2008. URL <http://dl.acm.org/citation.cfm?id=1357127>.
- [27] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 43–52, Santa Barbara, CA, USA, 2011. URL <http://dl.acm.org/citation.cfm?id=2047202>.
- [28] Aniket Kittur, Susheel Khamkar, Paul André, and Robert Kraut. CrowdWeaver: visually managing complex crowd work. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1033–1036, 2012. URL <http://dl.acm.org/citation.cfm?id=2145357>.
- [29] Ryan KL Ko, Stephen SG Lee, and Eng Wah Lee. Business process management (BPM) standards: a survey. *Business Process Management Journal*, 15(5): 744–791, 2009. URL <http://www.emeraldinsight.com/journals.htm?articleid=1811155&show=abstract>.
- [30] Ryan KL Ko, Eng Wah Lee, and S. G. Lee. Business-OWL (BOWL)-a hierarchical task network ontology for dynamic business process decomposition and formulation. *Services Computing, IEEE Transactions on*, 5(2):246–259, 2012. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5989787.
- [31] Ioannis Konstas, Vassilios Stathopoulos, and Joemon M. Jose. On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 195–202, Boston, MA, USA, 2009. ACM. ISBN 978-1-60558-483-6. doi: 10.1145/1571941.1571977. URL <http://portal.acm.org/citation.cfm?id=1571977>.
- [32] Anand P. Kulkarni, Matthew Can, and Bjoern Hartmann. Turkomatic: Automatic recursive task and workflow design for mechanical turk. In *Proceedings of the 2011 Annual Conference on Extended Abstracts on Human Factors in Computing Systems*, pages 2053–2058, Vancouver, BC, Canada, 2011. URL <http://dl.acm.org/citation.cfm?id=1979865>.
- [33] Peter Lawrence. *Workflow handbook 1997*. John Wiley & Sons, Inc., 1997. URL <http://dl.acm.org/citation.cfm?id=272945>.

- [34] Sheen S. Levine and Robert Kurzban. Explaining clustering in social networks: Towards an evolutionary theory of cascading benefits. *SSRN eLibrary*, 27(2-3). URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=890232.
- [35] Joseph Carl Robnett Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, (1):4–11, 1960. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4503259.
- [36] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 68–76, 2010. URL <http://dl.acm.org/citation.cfm?id=1837907>.
- [37] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurkIt: Human computation algorithms on mechanical turk. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology*, pages 57–66, New York, NY, USA, 2010. URL <http://dl.acm.org/citation.cfm?id=1866040>.
- [38] Shuangling Luo, Haoxiang Xia, Taketoshi Yoshida, and Zhongtuo Wang. Toward collective intelligence of online communities: A primitive conceptual model. *Journal of Systems Science and Systems Engineering*, 18(2):203–221, 2009. URL <http://link.springer.com/article/10.1007/s11518-009-5095-0>.
- [39] Nuno Luz, Nuno Silva, Paulo Maio, and Paulo Novais. Ontology alignment through argumentation. In *2012 AAI Spring Symposium Series*, Palo Alto, CA, USA, 2012. URL <http://www.aaai.org/ocs/index.php/SSS/SSS12/paper/download/4335/4692>.
- [40] Nuno Luz, Nuno Silva, and Paulo Novais. Social networked multi-agent negotiation in ontology alignment. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, volume 2, pages 310–315. IEEE, 2012.
- [41] Nuno Luz, Carlos Pereira, Nuno Silva, Paulo Novais, António Teixeira, and Miguel Oliveira e Silva. An ontology for human-machine computation workflow specification. In *Lecture Notes in Artificial Intelligence*, volume 8480, Salamanca, Spain, 2014. Springer.
- [42] Nuno Luz, Nuno Silva, and Paulo Novais. A method for defining human-machine micro-task workflows for gathering legal information. In *AI Approaches to the Complexity of Legal Systems*, pages 275–289. 2014.

- [43] Nuno Luz, Nuno Silva, and Paulo Novais. Generating human-computer micro-task workflows from domain ontologies. In *Human-Computer Interaction. Theories, Methods, and Tools*, pages 98–109. Springer, 2014. URL http://link.springer.com/chapter/10.1007/978-3-319-07233-3_10.
- [44] Nuno Luz, Nuno Silva, and Paulo Novais. A survey of task-oriented crowdsourcing. *Artificial Intelligence Review*, pages 1–27, 2014. URL <http://link.springer.com/article/10.1007/s10462-014-9423-5>.
- [45] Hao Ma, Dengyong Zhou, Chao Liu, Michael R. Lyu, and Irwin King. Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 287–296, 2011. URL <http://dl.acm.org/citation.cfm?id=1935877>.
- [46] Alexander Maedche, Boris Motik, Nuno Silva, and Raphael Volz. Mafra - a mapping framework for distributed ontologies. In *Knowledge engineering and knowledge management: ontologies and the semantic web*, pages 235–250. Springer, 2002. URL http://link.springer.com/chapter/10.1007/3-540-45810-7_23.
- [47] David Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, and others. OWL-s: semantic markup for web services. w3c member submission (2004). Retrieved from <http://www.w3.org/Submission/OWL-S>, 2012.
- [48] Winter Mason and Duncan J. Watts. Financial incentives and the performance of crowds. *ACM SigKDD Explorations Newsletter*, 11(2):100–108, 2010. URL <http://dl.acm.org/citation.cfm?id=1809422>.
- [49] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE intelligent systems*, 16(2):46–53, 2001. URL <http://www.computer.org/csdl/mags/ex/2001/02/x2046.pdf>.
- [50] Gregory Mentzas, Christos Halaris, and Stylianos Kavadias. Modelling business processes with workflow systems: an evaluation of alternative approaches. *International Journal of Information Management*, 21(2):123–135, 2001. URL <http://www.sciencedirect.com/science/article/pii/S0268401201000056>.
- [51] Christine Natschläger. Towards a bpmn 2.0 ontology. In *Business Process Model and Notation*, pages 1–15. Springer, 2011.

- [52] Leo Obrst, Howard Liu, and Robert Wray. Ontologies for corporate web applications. *AI Magazine*, 24(3):49, 2003. URL <http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/1718>.
- [53] Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. *Science of Computer Programming*, 67(2-3):162–198, July 2007. ISSN 0167-6423. doi: 10.1016/j.scico.2007.03.002. URL <http://www.sciencedirect.com/science/article/pii/S0167642307000500>.
- [54] Gabriele Paolacci, Jesse Chandler, and Panagiotis Ipeirotis. Running experiments on amazon mechanical turk. *Judgment and Decision Making*, 5(5):411–419, 2010. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1626226.
- [55] Carlos Pedrinaci, John Domingue, and Amit P. Sheth. Semantic web services. In *Handbook of semantic web technologies*, pages 977–1035. Springer, 2011. URL http://link.springer.com/10.1007/978-3-540-92913-0_22.
- [56] Carlos Pereira, António J. S. Teixeira, and Miguel Oliveira e Silva. Live evaluation within ambient assisted living scenarios. In *7th ACM Conference on Pervasive Technologies Related to Assistive Environments - PETRA*, Rhodes, Greece, May 2014.
- [57] Perrine Pittet, Christophe Cruz, and Christophe Nicolle. A structural\ mathcal {SHOIN (D)} ontology model for change modelling. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*, pages 442–446. Springer, 2013.
- [58] J. Porter. *Designing for the Social Web*. Peachpit Press, 2008. ISBN 0321534921, 9780321534927.
- [59] Alexander J. Quinn and Benjamin B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems*, pages 1403–1412, 2011. URL <http://dl.acm.org/citation.cfm?id=1979148>.
- [60] Dominique Ritze, Johanna Völker, Christian Meilicke, and Ondrej Šváb-Zamazal. Linguistic analysis for complex ontology matching. *Ontology Matching*, 1, 2010.
- [61] Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow data patterns: Identification, representation and tool support. In *Conceptual Modeling-ER 2005*, pages 353–368. Springer, 2005. URL http://link.springer.com/chapter/10.1007/11568322_23.

- [62] Cristina Sarasua, Elena Simperl, and Natalya F. Noy. CrowdMap: Crowdsourcing ontology alignment with microtasks. In *The Semantic Web - ISWC 2012*, number 7649 in LNCS, pages 525–541. Springer, 2012. URL http://link.springer.com/chapter/10.1007/978-3-642-35176-1_33.
- [63] François Scharffe, Ondřej Zamazal, and Dieter Fensel. Ontology alignment design patterns. *Knowledge and Information Systems*, 40(1):1–28, 2014. URL <http://link.springer.com/article/10.1007/s10115-013-0633-y>.
- [64] François Scharffe. *Correspondence patterns representation*. PhD thesis, University of Innsbruck, 2009.
- [65] François Scharffe and Dieter Fensel. Correspondence patterns for ontology alignment. In *Knowledge Engineering: Practice and Patterns*, pages 83–92. Springer, 2008.
- [66] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. *Process-Aware Information Systems*, pages 119–146, 2005. URL <http://www.google.com/books?hl=pt-PT&lr=&id=ZENNdQq8p74C&oi=fnd&pg=PA119&dq=event+process+chains&ots=ZhQK3YRZdJ&sig=3VEwyDU9f0g00sclKR0vTQq77FQ>.
- [67] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. In *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, number 2519 in LNCS, pages 1223–1237. Springer, 2002. URL http://link.springer.com/chapter/10.1007/3-540-36124-3_77.
- [68] Ljiljana Stojanovic. *Methods and tools for ontology evolution*. 2004.
- [69] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1):161–197, 1998. URL <http://www.sciencedirect.com/science/article/pii/S0169023X97000566>.
- [70] James Surowiecki. *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies and nations*. New York: Doubleday, 2004.
- [71] Vuong Xuan Tran and Hidekazu Tsuji. OWL-t: an ontology-based task template language for modeling business processes. In *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*, pages 101–108. IEEE, 2007. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4296924.

- [72] W. M. P. Van der Aalst. Petri-net-based workflow management software. In *Proceedings of the NFS Workshop on Workflow and Process Automation in Information Systems*, pages 114–118. Citeseer, 1996.
- [73] Wil MP van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998. URL <http://www.worldscientific.com/doi/abs/10.1142/S0218126698000043>.
- [74] Wil MP Van der Aalst and Arthur HM Ter Hofstede. YAWL: yet another workflow language. *Information systems*, 30(4):245–275, 2005. URL <http://www.sciencedirect.com/science/article/pii/S0306437904000304>.
- [75] Wil MP Van Der Aalst, Arthur HM Ter Hofstede, and Mathias Weske. Business process management: A survey. In *Business process management*, pages 1–12. Springer, 2003. URL http://link.springer.com/chapter/10.1007/3-540-44895-0_1.
- [76] Luis Von Ahn. Human computation. In *46th ACM IEEE Design Automation Conference*, pages 418–419, 2009. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5227025.
- [77] S Wasserman and K Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.
- [78] Wesley Willett, Jeffrey Heer, and Maneesh Agrawala. Strategies for crowdsourcing social data analysis. In *Proceedings of the 2012 ACM Annual conference on Human Factors in Computing Systems*, pages 227–236, 2012. URL <http://dl.acm.org/citation.cfm?id=2207709>.
- [79] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. A survey of crowdsourcing systems. In *IEEE International Conference on Privacy, Security, Risk and Trust*, Boston, MA, USA, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6113213.
- [80] Man-Ching Yuen, Irwin King, and Kwong-Sak Leung. Task matching in crowdsourcing. In *Internet of Things (iThings/CPSCoM), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 409–412, 2011. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6142254.