

Ontology of Functions

A Domain-independent Framework for Modeling Functions

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
eingereichte

D I S S E R T A T I O N

zur Erlangung des akademischen Grades
DOCTOR RERUM NATURALIUM
(Dr. rer. nat.)

im Fachgebiet
Informatik

vorgelegt
von Magister Patryk Burek

geboren am 6. Juli 1978 in Lublin
Leipzig, den 20. Juli 2006

Bibliographische Angaben

Burek, Patryk

Ontology of Functions: A Domain-independent Framework for Modeling Functions

Universität Leipzig, Dissertation

240 S., 214 Lit., 31 Abb., 8 Tab.

To Łucja

Abstract

In many domains entities are considered in terms of their functions, starting with the design of artifacts, through natural and social sciences and ending with folk theories and common sense knowledge. However, there is a lack of a domain-independent ontological framework for representing and modeling functions. Such a framework could be given by top-level ontologies, providing the specification of the most general, domain-independent concepts, in contrast to domain ontologies which describe conceptualizations of particular domains. However, current top-level ontologies such as DOLCE, SUMO or GFO either do not include the notion of function or handle it scantily.

The objective of this work is to develop a formal top-level ontology of functions (OF), applicable across various domains, and to incorporate it into a broader ontological framework. OF is concerned with five main issues, namely the representation of the structure of functions and their interrelations, the realization of functions, function ascription, and the incorporation of OF into the top-level ontology GFO. The first two issues are of relevance in functional modeling, where it is required to represent functions independently of the particular ways of their realization. Secondly, we find it important to provide the ontological foundations for the evaluation of entities against their capabilities of realizing functions. Thirdly, since the functional description is often a part of the knowledge about entities, it is important to provide conditions for assigning functions to entities. Finally, OF is incorporated into the wider framework of GFO which provides the means for a cohesive representation of both functional and non-functional knowledge.

The developed solution is intended to be applied in domain ontologies and conceptual models. For example, OF has been recognized to be beneficial for the Open Biomedical Ontologies (OBO) as a general framework for representing biological functions and together with GFO it is used as the foundation for a Biological Core Ontology. In addition, OF provides the basis on which an extension to the Unified Modeling Language (UML) has been proposed, the current de facto standard in object-oriented conceptual modeling, that is recently also proposed for ontological engineering. The extension is introduced to UML in the form of a profile, enabling the construction of functional models. The profile, among others, introduces graphical notations which allow for the visualization of functional models.

Acknowledgements

I am thankful to my wife, Łucja, and to my parents.

Moreover, I cannot imagine accomplishing this work without the support of the many people who I was lucky to be surrounded by during my stay in Leipzig. First of all I am grateful to my supervisor, Professor Heinrich Herre who guided my work with kindness, wisdom and patience. He introduced me to the subject of ontologies in computer science and helped me to shape the topic of my work. In critical moments he encouraged me to work further and was always ready to help me. I am thankful to him for the numerous inspiring discussions. I am also grateful to Dr. Barbara Heller, the tragically deceased second supervisor of mine, who supported my efforts especially in the early stages of my work on this thesis.

I am deeply indebted to the members of the Onto-Med Research Group established by Professor Herre and Dr. Heller. The collaboration within the group gave me many scientific inspirations as well as the chance to find good friends. In particular I am deeply grateful to Frank Loebe who read every page of this thesis and spent long hours on discussing issues relevant for this work. Moreover, I have learned a lot from Frank about roles and my results rely on his account of them. Hannes Michalek and his research on the issue of causality also became of great help in my work, especially in the context of function realization. I owe to Robert Hoehndorf the application of my results to biological ontologies, which greatly contributed to the evaluation and improvement of my results.

The activities of the Onto-Med Research Group gave me an opportunity to meet many inspiring people, in particular Giancarlo Guizzardi, Gerd Wagner and Roberto Poli. The ideas of Giancarlo Guizzardi and Gerd Wagner of applying top-level ontologies to conceptual modeling provoked my work on the development of the UML profile for functional modeling.

I also had a unique chance of having an inspiring cooperation within the Ontologies in Biomedicine Group. The sound feedback from the members of the group, in particular from Janet Kelso, Johann Visagie and Michael Lachmann helped me to verify my account of functions as well as gave me a better understanding of biological ontologies and the biological understanding of functions.

I am also thankful to the members of the Graduiertenkolleg Wissensrepräsentation, a doctoral programme in Knowledge Representation at the University of Leipzig. I would like to thank Hesham Khalil, Sören Auer and Rafał Graboś with whom I had not only an honor to work but also an exciting social life. Hesham stimulated my work and was always eager to share his rich scientific experience with me. Sören was my guide not only to the Semantic Web

but also to the social life of Leipzig and to the countryside of Saxony. Rafał was my roommate for three years and a true companion in many long lasting discussions on ontologies and artificial intelligence. Additionally, I feel indebted to Jerzy Warakowski for his help in the editorial work on this thesis.

Last but not least, I would like to thank to Deutsche Forschungsgemeinschaft and Professor Markus Löffler for the financial support, without which this work, as well as my stay in Leipzig, would not have been possible.

Leipzig,
July, 2006

Patryk Burek

Contents

List of Figures	xiv
------------------------	------------

List of Tables	xv
-----------------------	-----------

1	Introduction	1
1.1	Functions and Functional Knowledge	1
1.2	Ontologies	5
1.2.1	Definition and Classification	5
1.2.2	Top-level Ontologies	7
1.2.3	Representation Formalisms	14
1.2.4	Applications.....	15
1.3	Objectives	17
1.4	Structure.....	21
2	Related Works	22
2.1	Functional Device Representations	22
2.1.1	Functions as Input-Output Pairs	23
2.1.2	Function and Behavior	24
2.1.3	Functions as Intended Roles	33
2.1.4	Functions as Effects.....	35
2.1.5	Conclusions	37
2.2	Software Engineering and Business Modeling	39
2.2.1	Structured Methods	39
2.2.2	Object-Oriented Modeling and UML	41
2.2.3	Object-Process Methodology	45
2.2.4	Conclusions	48
2.3	Functions in Philosophy.....	48
2.3.1	Functions as Dispositions	49
2.3.2	Etiological Theories.....	50
2.3.3	Ontological Status of Functions	53
2.3.4	Conclusions	54
2.4	Requirements for an Ontology of Functions	54

3	Structure of Functions	57
3.1	Introduction	57
3.2	Label.....	57
3.3	Goal	59
3.3.1	Affected by the Function.....	61
3.3.2	Agent.....	62
3.3.3	Established by an Agent.....	63
3.3.4	Kinds of Establishing Goals.....	64
3.3.5	Priority of Goals and Functions	66
3.3.6	Arbitrary Chunk of Reality	67
3.3.7	Final State	68
3.3.8	Complexity of Functions.....	69
3.3.9	Restrictions on Functions.....	71
3.4	Requirements.....	72
3.5	Temporal Extensions of Functions	75
3.6	Functional Item.....	79
3.6.1	Role.....	81
3.6.2	Functional Item as a Role.....	82
3.6.3	Discussion.....	84
3.7	Side Effects.....	85
3.8	Summary	86
4	Relations between Functions	88
4.1	Introduction	88
4.2	Instantiation.....	88
4.2.1	Individual, Universal Functions and Instantiation	90
4.3	Taxonomic Relations.....	94
4.3.1	Introduction.....	94
4.3.2	Subsumption, Specialization and Individualization	96
4.4	Part-Whole Relation	98
4.4.1	Introduction.....	98
4.4.2	Function-Part	99
4.4.3	Sequence-Part	100
4.5	Additional Relations between Functions	101
4.5.1	Support, Enable and Prevent	102
4.6	Summary	103

5	Realization	106
5.1	Introduction.....	106
5.2	Individual Actual Realization	108
5.2.1	Actual Culminative Realization.....	109
5.2.2	Actual Non-culminative Realization	112
5.2.3	Actual Situational Realization	113
5.3	Minimal Actual Realization and its Components.....	115
5.3.1	Minimal Actual Realization.....	115
5.3.2	Means of Realization	116
5.3.3	Actual Realizer	117
5.3.4	Dynamic and Passive Realizer	119
5.4	Universal Realization and Realizer	120
5.4.1	Universal Minimal Realization.....	120
5.4.2	Universal Realizer	122
5.5	Dispositional Realizer and Realization	122
5.5.1	Dispositional Realizer	122
5.5.2	Dispositional Realization.....	124
5.5.3	Strong Dispositional Realizers	125
5.6	Functional Realization	126
5.7	Realization-dependent Relations between Functions.....	127
5.7.1	Trigger.....	128
5.7.2	Improve	128
5.8	Summary	129
6	Ascription of Functions	132
6.1	Introduction.....	132
6.2	Actual and Dispositional Function.....	134
6.3	Intended Function	135
6.3.1	Inherited Intended Function.....	141
6.4	Universal Has-Function	141
6.5	Function Bearers	143
6.6	Functionality and Multiple Function Ascriptions	144
6.7	Malfunctions	145
6.7.1	Malfunction with respect to Intended Function.....	146
6.7.2	Malfunctions with respect to History	147
6.7.3	Malfunction in Comparison.....	148
6.7.4	Priorities of Malfunctions.....	149
6.7.5	Side-Effect Malfunction	150
6.8	Summary	151

7	Ontological Status of Functions, Classifications and Architecture	152
7.1	Introduction	152
7.2	Characteristics of Functions	152
7.2.1	Context and Subjectivity	152
7.2.2	Teleology and Goal-Orientedness	154
7.3	Candidates for Functions	155
7.3.1	Functions as Processes	155
7.3.2	Functions as Goals	156
7.3.3	Functions as Intentional Entities	157
7.4	Classifications of Functions	158
7.4.1	Intrinsic Classifications	159
7.4.2	Extrinsic Classifications	161
7.4.3	Reconstruction of the Current Classifications	163
7.5	Architecture	166
7.5.1	Non-functional Layer	167
7.5.2	Pure Functional Layer	167
7.5.3	Realization Layer	168
7.5.4	Impure Functional Layer	169
7.6	Summary	171
8	A UML Profile for Functional Modeling founded on OF	172
8.1	Introduction	172
8.2	UML and Ontology Engineering	172
8.3	Objectives	174
8.4	Method	175
8.5	Overview of the Architecture	175
8.5.1	Ontology Profile	177
8.5.2	Functions Package	178
8.5.3	Functional Relations Package	183
8.5.4	Function Ascriptions Package	188
8.5.5	Malfunctions Package	194
8.5.6	Impure Function Structures Package	197
8.5.7	Impure Functional Relations Package	198
8.6	Discussion and Conclusions	200
9	Conclusions	203
9.1	Problem	203
9.2	Solution	203
9.3	Advantages	203
9.4	Applications	206

9.4.1	Conceptual Modeling	206
9.4.2	Biological Ontologies.....	207
9.5	Future Work.....	211
Appendix A: GFO Terms and Definitions		213
References		220
Index of Symbols		238

List of Figures

Figure 1.	Categorization of Ontologies.....	6
Figure 2.	Taxonomy of SUMO topmost level categories.	9
Figure 3.	Taxonomy of DOLCE topmost level categories	11
Figure 4.	Taxonomy of GFO topmost level categories.....	13
Figure 5.	Three architectures of functional ontologies	20
Figure 6.	Part of the hierarchy of functional knowledge.	24
Figure 7.	CFRL definition of function.....	28
Figure 8.	Types of plant decomposition used in MFM.....	34
Figure 9.	Hierarchy of functions in the CASE*Method.	40
Figure 10.	Use Case diagram.....	42
Figure 11.	Business Goal Allocation Pattern.....	44
Figure 12.	Functions in OPM	47
Figure 13.	Time-extents of Functions.....	77
Figure 14.	Business Actor-Role Pattern	82
Figure 15.	Function Decomposition	105
Figure 16.	Realization of Function	109
Figure 17.	Functional Item and Realizer	123
Figure 18.	Universal Minimal Realization and Individual Realization	125
Figure 19.	Architecture of OF.....	167
Figure 20.	Universal Functional Item, Individual Functional Item and Realizer	170
Figure 21.	Dependencies between Profiles.....	175
Figure 22.	Dependencies within the Function Ontology Profile	176
Figure 23.	Dependencies within the Functions Package	176
Figure 24.	Class diagram of the Functions Package	178
Figure 25.	Class diagram of the Functional Relations Package.....	183
Figure 26.	Class diagrams of the Function Ascriptions Package.....	188
Figure 27.	Class diagram of the Has-Functions Package	192
Figure 28.	Class diagram of the Malfunctions Package	194
Figure 29.	Class diagram of the Impure Function Structures Package	197
Figure 30.	Class diagram of the Impure Relations Package	198
Figure 31.	Exemplary Biological Functions employing OF.....	208

List of Tables

Table 1.	Examples of GFO entities.....	14
Table 2.	List of the requirements for a top-level ontology of functions.	56
Table 3.	GFO categories as function goals.	68
Table 4.	Exemplary function bearers	143
Table 5.	Categories of OF belonging to the Pure Functional Layer.	168
Table 6.	Categories of OF belonging to the Realization Layer.....	169
Table 7.	Categories of OF belonging to the Impure Functional Layer.	170
Table 8.	Reference list of GFO terms and definitions.	219

1 Introduction

This work is concerned with the representation of functions and functional knowledge in ontologies, in particular in top-level ontologies. In the current section we give the preliminaries, motivations and objectives of our work. In section 1.1 we introduce our understanding of the notions of function and functional knowledge, and demonstrate their relevance across various domains. Section 1.2 briefly refers the issue of ontologies and in particular top-level ontologies. In section 1.3 the motivations and objectives of the work are discussed. Finally, section 1.4 provides an overview of the structure of this work.

1.1 Functions and Functional Knowledge

The current work is concerned primarily with the notion of function. However, our interests do not concern the mathematical understanding of function as the binary relation, such that for every element x the element y is uniquely determined. Rather we are interested in the common-sense usage of the term.

To illustrate our understanding of the notions of functions and functional knowledge we will consider the example of a house. Different views of a house can be taken. One could consider a house from the perspective of its history asking about the origins of the house, the circumstances of its construction, the history of people living in it etc. In another view of a house one could focus on the physical features like height, shape or the color of elevation. Additionally, a house could be decomposed to components like foundations, walls and roof. The interior of a house can be partitioned into a cellar, staircases and rooms, which in turn can be specified as dining rooms, bedrooms etc. Finally, a house can be decomposed into its subsystems e.g. a ventilation system or a heating system. The view of the house in which we are here interested is a *functional* one. Roughly speaking, the functional view of some item is oriented at representing functions, taken as purposes or goals associated with this item, instead of representing other aspects of the item, such as the physical structure or history.

An item as a whole may be interpreted in functional terms, i.e. a house may be understood as a physical object that has the function of providing shelter against precipitation, wind, heat, cold and intruding humans and animals. But also parts of an item may be explained functionally, e.g. a staircase may be defined as an indoor space having the function of enabling

transportation between floors, a sleeping room as a room to sleep in. In fact, items are often decomposed into parts due to the function those parts play. Whereas a staircase and a sleeping room are distinguished due to their function *and* their physical composition (both are kinds of indoor space) some parts are distinguished exclusively on the basis of their function. Such parts are often called *systems* and *subsystems*. For example, the reason for perceiving a net of devices generating fresh air, ventilators, tunnels, windows and doors as a whole (as a ventilation system), is the function to which all those entities contribute, namely the function of the maintenance of the acceptable indoor air quality.

It is not only the parts of an item that may be functionally described and explained, but also its properties and the properties of its parts, e.g. the thickness and the material of the external walls is chosen, among other things, in order to avoid the waste of heat.

The knowledge referring to functions, rather than to other aspects like physical features, structure or behavior, is called the *functional knowledge*. It is clearly observable that functional knowledge is present in all domains of our life. In the first, place all artifacts are described in functional terms. It is common to understand both simple and complex technical artifacts not in terms of their physical properties, like structure or behavior, but in terms of the functions they have. Functional knowledge has been recognized as useful at least at three stages of the process of design and redesign of artifacts [Shimomura et al., 1995]:

1. *Requirements specification.* A part of device requirements is specified in terms of functional concepts. In fact, functional requirements at the initial state of design play a crucial role, since at that stage non-functional requirements are often not taken into consideration yet.
2. *Designed object representation.* Functional knowledge provides an alternative to the physical description of a device. An association of a physical description with a functional description helps in understanding the former. Functional models explain the structure and the behavior of an object: they record and communicate the designer's intentions ([Stone, Wood, 2000], p. 359) and bridge them with the physical structure and the behavior of the object [Umeda, Tomiyama, 1995], p. 71). Explicitly stated functions help to understand the modularization of the structure of a device, since it is often driven by the functional decomposition ([Stone, Wood, 2000] p. 359).
3. *Evaluation.* The design of an artifact is evaluated from the functional point of view and thus verification of an artifact against the intended functions is the foundation of the evaluation. In simulation and diagnosis an explicit functional knowledge permits to check whether the objectives are achieved. Moreover, the comparison of artifacts is commonly based on the comparison of functionality ([Stone, Wood, 2000] p. 359).

As [Rosenman, Gero, 1999] observe, the importance of functions increases even more in the multidisciplinary design. If many domains are involved in the design, it may be difficult or even impossible to grasp the whole design in its technical complexity, but often it is enough to understand the intuitions about the functions of an artifact and its components.

Although the above benefits of functions and of the functional knowledge have been discussed primary in the area of technical artifact modeling, they concern all types of artifacts. Let us consider here two additional cases: business system design and software engineering.

In business modeling it is commonly agreed to distinguish between an enterprise's *business function* (i.e. what it does in order to achieve its objective) and *business process* (i.e. the activities performed by an enterprise in order to realize its business functions). In business system design business functions play an analogous role to the functions in technical design. They provide the means to specify requirements and play a crucial role in the first phases of system design. The modeling of functions is also of use in software engineering, in particular in structured system analysis, where functional decompositions and functional dependencies are modeled.

In fact, functional knowledge is not only of importance in the context of artifact design. Also natural and social science use functions to describe the phenomena in their domains of interest. A well known example of functional explanation in the field of biology is that given by the British ecologist H. B. D. Kettlewell concerning the peppered moth living on the trees in the area of Liverpool. During the industrial revolution the pollution darkened the naturally light bark of the trees. The color of the peppered moth that populated the trees darkened as well. When the pollution was reduced and the bark of the trees lightened again, the peppered moth went back to its previous color as well. An intuitive explanation of that phenomenon, followed by many evolutionary biologists, is that the color of the peppered moth serves as a camouflage against predators. In this sense the explanation of the change of color is given in functional terms, i.e. by the reference to the function it should realize – the protection from predators¹.

Reference to function can often also explain the presence of an organ or a trait. For instance the presence of the hemoglobin in blood or the presence of the heart in mammal organisms can be justified by their functions, that is transporting oxygen from the lungs to other parts of the body and pumping blood, respectively. Both functions are essential for the survival of an organism, and thus the presence of the organs and substances realizing them is justified.

¹ Functional explanation in biology is not free from difficulties. The functional explanation concerning the peppered moth given by Kettlewell is subject to some criticism, e.g. in [Majerus, 1998].

Functional explanation is present also in biochemistry, e.g. the actions of macromolecules are described functionally and intentionally - macromolecules build, repair, recognize, and make errors: all these characteristics have functional and even intentional nature.

Last not least, functions play an important role not only in engineering and natural science but also in common sense knowledge. The structure of common sense knowledge and the principles of human categorizations have been the subject of cognitive science and experimental psychology for decades. A significant role of functions in human categorization has been found especially in the context of psychological essentialism. Psychological essentialism is the result of a series of experiments (including [Keil, 1989; Gelman, Wellman, 1991]) which have shown that human concepts are not mere composites of necessary and sufficient characteristics but instead possess a particular structure. According to this approach people act as if concepts have some “essential” properties that are both criterial for category membership and responsible for other “surface” features of concepts [Medin, Ortony, 1989]. According to psychological essentialism essential features drive human categorization and are believed to determine the rest of the features. Among the candidates for essential features, especially in the context of the categorization of artifacts, are functional features. A number of experiments (e.g. [Barton, Komatsu, 1989; Gelman, Bloom, 2000; Matan, Carey, 2001; Keleman, 1999]) seem to provide the evidence that what guides people’s intuitions about artifact category membership is intended function, which is thus the essence of artifact concepts. It is claimed that people categorize artifacts with respect to their function. For example, [Barton, Komatsu, 1989] as a result of their experiments found out that the objects which are not able to perform their functions are not considered by people to continue being the same kind of artifact. For example a mirror that did not reflect an image was not considered by the subjects of the experiments as a mirror. In addition, the experiments of Barton and Komatsu have shown that function is a sufficient condition for the categorization of objects as kinds of the same artifact. The object which was neither hard nor made of glass but did reflect an image was considered as a mirror.

Function not only determines the kind of an artifact, but also its properties, its actual usage and the classification of artifacts. For example, the function of a knife, to cut, dictates both the shape and the actual usage of knives. Moreover, variation of the functions of a knife results in different kinds of knives, e.g. bread knife, hunting knife or scalpel.

Although functionally interpreted essence is not free from difficulties² it nevertheless provides a strong evidence for the significance of the notion of function in human categorization, as well as in folk theories.

In conclusion we can see that functions and functional knowledge are present across a vast spectrum of our knowledge, starting with design of artifacts, through natural science, and ending with folk theories and common sense knowledge. Functional knowledge not only provides an answer to the question of what purpose a given entity serves; in addition, function often explains the presence of an entity, its structure and behavior.

1.2 Ontologies

After demonstrating briefly our understanding of functions and their relevance across domains now we are about to give a short introduction to ontologies, particularly to top-level ontologies.

1.2.1 Definition and Classification

Although the term *ontology*, with its origins in philosophy, is hardly new, in computer science it has gained a widespread popularity over about the last two decades. One of the first attempts to define the term came from [Neches et al., 1991; Gruber, 1993]. Although until today there is no universally accepted definition (for discussion see [Guarino, Giaretta, 1995; Guarino, 1997a]), a popularly cited definition is Gruber's: "an ontology is an explicit specification of a conceptualization" [Gruber, 1993]. Here *conceptualization*, according to [Studer et al., 1998], should be understood as an abstract model of some phenomenon, which provides the relevant concepts. [Guarino, 1998] argues against understanding conceptualization in purely extensional terms and stresses its intensional character. The term *explicit* refers to the fact that the concepts used in the ontology are explicitly defined.

Ontologies are also considered to have the properties of being *formal* and *shared* (e.g. [Gruber, 1994; Borst, 1997]). The former refers to the fact that ontologies are represented by means of some formal language which makes them machine-readable. The latter refers to the fact that ontologies are not private but they are the result of an agreement or some common understanding of the phenomena.

² See section 2.3.2 for Keil's [Keil, 2003] argumentation against essence considered as an intended function.

The term *ontology* gained an increasing popularity, which resulted in applying it to various information systems ranging from simple catalogs, through glossaries, thesauruses, taxonomies and collections of frames to formal logical theories [Smith, Welty, 2000]. This mishmash called for a systematization, and the introduction of various flavors of ontologies.

A number of such systematizations have been proposed (for an overview see [Gomez Perez et al. 2004]). Particularly two criteria for systematizing ontologies are taken into consideration by [Gomez Perez et al., 2004]: (1) the richness of the internal structure of the ontology (2) the subject of conceptualization. The first criterion introduced by [Lassila, McGuinness, 2001] systematizes the wide range of systems discussed by [Smith, Welty, 2000] (see figure 1) and shows that nowadays the degree of formality varies significantly across different ontology types. Most informal ontologies are the controlled vocabularies which are mere catalogs of terms, and glossaries which additionally provide natural language specifications of terms. To be the most formal are considered ontologies which permit to specify first-order logic constraints such as disjointness or inverse. The second criterion permits to distinguish among others [Guarino, 1997b]:

- Top-level ontologies describing the most general concepts present across domains under which the more specific concepts can be underpinned. Typical concepts of a top-level ontology are *process*, *object*, or *role*.
- Domain ontologies specifying the conceptualization of particular domains (e.g. enterprise, service, chemistry, etc). The concepts of domain ontologies are the specializations of the concepts of top-level ontologies (e.g. *employee* is a specialization of *role*)
- Task ontologies specify the conceptualization related to a given task or activity (e.g. diagnosing, selling, etc.).
- Application ontologies are application dependent specializations of task and domain ontologies.

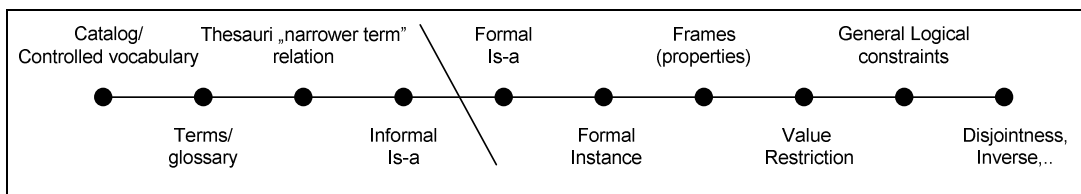


Figure 1. Categorization of ontologies (from [Lassila, McGuinness, 2001]).

In the present work we are concerned with ontologies which are considered by Lassila, McGuinness, 2001 to be the most formal namely the ontologies having the form of logical theories permitting to express general logical constraints and providing specifications of most general concepts. A formal ontology we understand as follows:

“it (formal ontology) consists of a structured vocabulary $V(\text{Ont})$, called ontological signature, and a set of axioms $Ax(\text{Ont})$ about $V(\text{Ont})$ which are formulated in a formal language $L(\text{Ont})$. Hence, an ontology (understood as a formal object) is then a system $\text{Ont} = (L, V, Ax)$; the symbols of V denote categories and relations between categories or between their instances. L can be understood as an operator which associates to a vocabulary V a set $L(V)$ of expressions which are usually declarative formulas.”([Heller et al., 2005], p. 8)

In accordance with the above we understand an ontology as the computer-based artifact, i.e. a formal model or a theory applicable in computer systems all of whose elements are artifacts referring to the objects of the real world and/or conceptualizations of those objects shared by a group of people. It is important to note that all ontological categories and individuals (later referred to as entities) are not considered here as the entities of the world, but as artifacts (elements of an ontological model) which describe/refer to the world. This understanding of an ontology is far from the philosophical sense of the term in which Ontology is considered as the discipline studying existence and being.

Defining ontologies in terms of formal models referring to some conceptualization makes them similar to database schemas, which can be understood as particular models of conceptualizations of some domain of interest. However, some authors find ontologies to be different from database schemas. For instance, [Spyns et al., 2002] observe that ontologies are domain-oriented while the database schemas are application-oriented. In this sense, ontologies ought to be reusable across different application in a given domain, whereas database schemas are not necessarily so. Following that line we could delimit ontologies from database schemas as the models oriented more on the actual organization of things in the world or the currently accepted conceptualizations, rather than on the temporary application needs.

1.2.2 Top-level Ontologies

In the current work we are interested in particular in top-level ontologies, also called foundational or upper-level ontologies (ULO). Top-level ontologies are not concerned with providing a complete description of everything, which is a doomed enterprise. As mentioned above they are concerned with the formal specification of most generic categories such as process, object, time or space, which provide the backbone of specific domain ontologies and are reusable across a broad range of domains. The generic character of the categories defined in ULO makes the development of those ontologies a cross domain effort of engineers, computer scientists, philosophers, library scientists and linguists. ULOs do not contain concepts specific to particular domains but instead provide a foundation, upon which domain

ontologies can be constructed. For instance on the basis of the general concept of process the domains dependent notions of business process or biological process may be introduced.

Several top-level ontologies have been developed in recent years. Some of them are developed as stand-alone ontologies, e.g. that of J.F. Sowa [Sowa, 2000], or the upper ontology of Russell and Norvig [Russell, Norvig, 1995]. Some are modules of bigger projects and are often aligned with domain ontologies, e.g. Suggested Upper Merged Ontology (SUMO) [Niles, Pease, 2001; Pease, Niles, 2002] developed by the Standard Upper Ontology Working Group at IEEE [SUO, 2005], Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [Masolo et al., 2003; Masolo et al., 2003] being a module of the WonderWeb Foundational Ontologies Library (WFOL) [WonderWeb, 2005], upper-ontology developed within the Smartkom Project [Smartkom Project, 2005], General Formal Ontology (GFO), being a component of the Integrated System of Foundational Ontologies (ISFO), which in turn is a part of the Integrated Framework for the Development and Application of Ontologies (IFDAO), the upper-level part of the ontology used in the MultiNet semantic network [Helbig, 2001], or the upper-level of the Cyc Project ontology [Cyc Project, 2005].

Below we give a short informal overview of the selection of the above ontologies, namely of Sowa's ontology, SUMO, DOLCE and in particular GFO, which the present work refers to. We present the most top-level categories and track the most significant ontological choices done in those ontologies.

Sowa's Ontology

Sowa's upper-level ontology was strongly influenced by the philosophical works of Peirce, Plato and Whitehead. The lattice of categories was developed pursuing a combinatorial approach based on three orthogonal distinctions: (1) *physical* vs. *abstract*, (2) *firstness*, *secondness*, *thirdness*, (3) *continuant* vs. *occurent*.

Concerning the first distinction Sowa understands abstract entities, in the spirit of Plato and Whitehead, as eternal, mathematical objects, which do not have a location in space or in time. In contrast to this, physical entities are located in space and time. The relation that holds between physical and abstract entities is that of characterization/ representation. An abstract entity characterizes, and is represented, in zero or more physical entities.

Concerning the second distinction Sowa's ontology is founded on Peircean notions of firstness, secondness and thirdness. Firstness is an Independent category which is represented in logic by a monadic predicate $P(x)$, "which describes an entity x by its inherent qualities, independent of anything external to x " ([Sowa, 2000], p. 61). Secondness is a Relative category, which can be represented as a dyadic predicate. The Relative grasps the external

relationship to some other entity. Thirdness is a mediating category that can be represented by means of a triadic predicate. The Mediating binds together the Independent and the Relative.

Finally, continuants are contrasted to occurrent on the basis of their relation to time. “A continuant has stable attributes or characteristics that enable its various appearances at different times to be recognized as the same individual. An occurrent is in a state of flux that prevents it from being recognized by a stable set of attributes. Instead, it can only be identified by its location in some region of space-time.” ([Sowa, 2000], p. 71)

The application of those three orthogonal distinctions results in 27 basic categories, which in turn are specialized further into more specific categories. Sowa provides a textual and a semi-formal specification of all of the categories of his ontology. Moreover he provides a pictorial language called Conceptual Graphs (CG) for representing ontologies, which is a system of logic based on the existential graphs of Charles Sanders Peirce and the semantic networks of artificial intelligence. CG are intended to merge logical precision and human readable form.

SUMO

Suggested Upper Merged Ontology (SUMO) [Niles, Pease, 2001; Pease, Niles, 2002], created at the Teknowledge Corporation, was intended to merge upper-level ontologies of Sowa, Russel and Norvig, Allen’s temporal axioms [Allen, 1984], the theory of holes by Casati and Varzi [Casati, Varzi, 1995], Smith’s ontology of boundaries [Smith, 1996; Smith, 1994], formal mereotopology developed in [Borgo et al., 1996; Borgo et al., 1997], and several representations of plans and processes, including Core Plan Representation [Pease, Carrico, 1997] and Process Specification Language [Schlenoff et al., 2000]. SUMO is developed in a variant of KIF called SUO-KIF and has been translated into various representation formalisms, including OWL and LOOM.

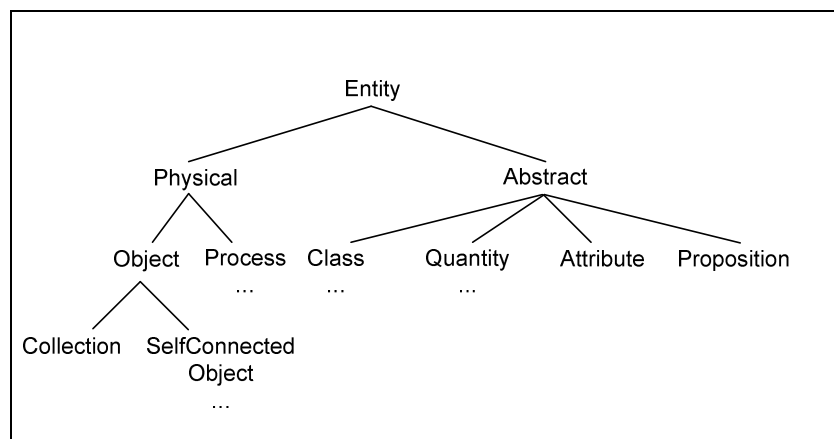


Figure 2. Taxonomy of SUMO most top level categories.

Categories of SUMO are organized into a taxonomy (figure 2), where the root category *entity* is subsumed by *abstract* and *physical entities*, just as it is the case in Sowa's ontology. Physical entities are divided into *processes* and *objects*, which is an adoption of the 3D view³ and resembles Sowa's distinction between continuant and occurrent. Abstract entities are in turn divided into (1) a *class* understood as a set together with its intensional condition of membership, (2) a *proposition*, which represents the notion of semantic or informational content, (3) an *attribute* covering all qualities, properties which are not reified as objects, and (4) a *quantity* covering physical quantities and numbers.

DOLCE

Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) is introduced as a first module of the WonderWeb foundational ontologies library developed within the WonderWeb Project. The aim of that ontology, as the authors admit in ([Masolo et al. 2002], p.8) is not to develop a universal standard ontology but rather give a starting point for comparison and making explicit the assumptions of other modules of WFOL. The axiomatization is given in modal first order logic and was translated into KIF, DAML+OIL, RDFS and OWL. DOLCE was integrated with methodologies for ontology building e.g. OntoSpec [Kassel, 2005] and was applied in the development and alignment of a number of ontologies and glossaries, e.g. WordNet [Gangemi et al., 2003].

The basic distinction of ontological categories in DOLCE is made between *endurants*, *perdurants*, *qualities*, and *abstract* entities. The distinction between endurants and perdurants is the adoption of the 3D view and, similarly to Sowa's ontology, refers to the behavior of an entity in time: (1) endurants are entirely present at any time of their existence, perdurants are only partially present at any time of their existence; (2) endurants *are* in time, whereas perdurants *happen* in time, (3) endurants can genuinely change in time, perdurants cannot undergo a change. The relation that holds between endurants and perdurants is the relation of *participation* - endurants participate in perdurants.

³ The 3D view postulates a basic ontological distinction between entities that are completely present at any moment of their existence, called objects or endurants, and entities that are not, called processes. The 3D view is contrasted to the 4D view regarding both objects and processes as four dimensional entities extended in time and space.

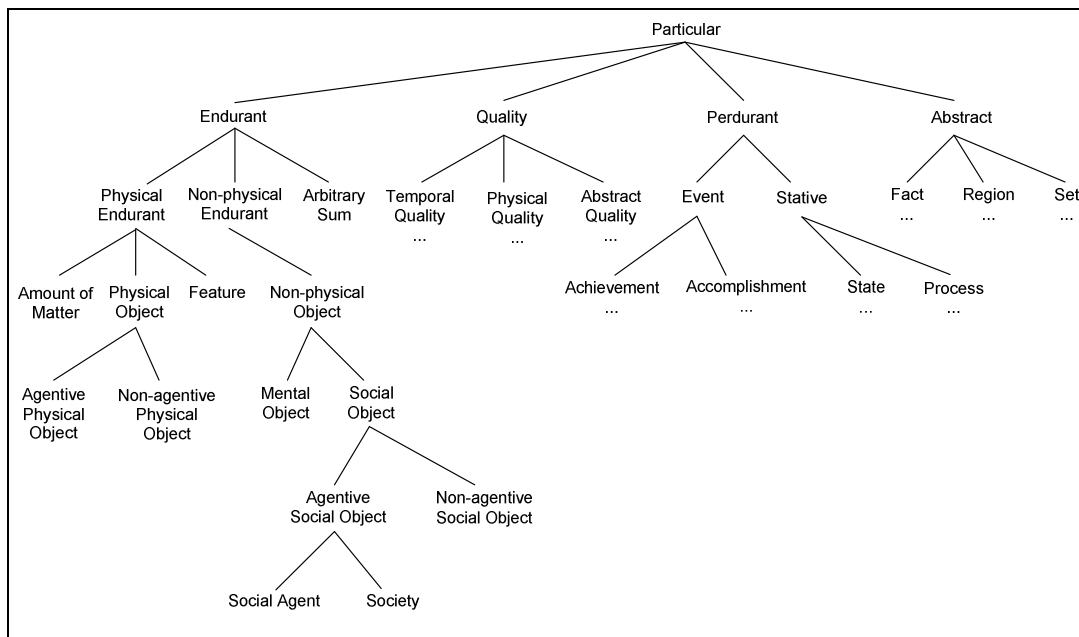


Figure 3. Taxonomy of DOLCE topmost level categories (from [Masolo et al., 2003], p. 14)

Qualities are the entities that we can perceive and measure. They are inherited to the entities they describe: every entity has some qualities with which it comes. Qualities are related to so-called quality spaces which correspond to qualitative sensorial experiences of humans [Gärdenfors, P. 2000]. Abstract entities are contrasted to perdurants and endurants as the entities located neither in space nor in time.

Perdurants, endurants, qualities and abstract entities are further decomposed by means of subsumption to more detailed categories (figure 3).

GFO

In the current section we give the overview of the General Formal Ontology (GFO), which the current work is built upon. GFO is the first ontology of the Integrated System of Foundational Ontologies (ISFO) being developed at the University of Leipzig by the research group Onto-Med [OntoMed, 2006]. ISFO is a part of an Integrated Framework for the Development and Application of Ontologies (IFDAO) whose predecessor was the GOL project [Degen et al., 2001; Heller, Herre, 2004; Heller et al., 2005] launched in 1999 as a collaborative research effort of the Institute for Medical Informatics, Statistics and Epidemiology (IMISE) and the Institute for Informatics (IfI) at the University of Leipzig. IFDAO covers not only the area of foundational ontologies but also the meta-ontological analysis. In particular, the topics of conceptual structures and principles of category building are discussed in [Herre, Loebe, 2005; Burek, 2004; Burek, 2005; Burek, Grabos, 2005].

GFO provides a taxonomy of entities, a taxonomy of relations, and the set of axioms describing them in FOL. Entities of GFO are organized into three levels of the description of

reality, namely *material*, *mental* and *social* level, in such a way that every entity of GFO participates in at least one of those levels (for further discussion of levels of reality see [Poli, 2001; Poli, 2002]). The material level comprises the perspective taken in biology, chemistry or physics. The mental level is organized around the representation of the psychological phenomenon and comprises most of what is studied in cognitive science. The social level, in turn, groups all entities referring to the social phenomenon, such as agent, organization, or society.

The root element in the GFO taxonomy is *entity*⁴ (figure 4) Entities are the basic elements of the ontology which refer to the items of reality. Three general kinds of entities are considered in GFO, namely *sets*, *categories*, and *individuals*.

Categories, also called *universals*, are intensional counterparts of classes. They are such entities that may be predicated of other entities and may be expressed and represented in terms of language. Moreover, they may be instantiated by other categories or by individuals.

Individuals instantiate categories but cannot be instantiated themselves. The distinction between individuals and universals can be seen by the analysis of the number of items of the real world they refer to. Universals may refer to more than one item of the world, whereas individuals refer to exactly one item of the world. Thus, the ontological distinction between universals and individuals should not be identified with the distinction between objects and classes of object-oriented paradigms, since not every object must refer to exactly one item of reality, e.g. *ape : species*, refers to all apes.

The main branch of the GFO taxonomy concerns individuals, but one should notice that primitive categories can be organized in the isomorphic taxonomy by means of individuals instantiating them. Individuals are organized into *entities of space and time* (*time entities* and *space entities*), *abstract* and *concrete individuals*.

GFO provides a build-in representation of time and space. Both time and space are organized into regions with explicit boundaries, called *spatial regions* (specialized to *topoids*), and *time regions* (specialized to *chronoids*) respectively. Temporal relations of *before*, *after* etc. are represented by the relations between the boundaries of time regions occupied by entities.

⁴ Table 1 provides the list of examples of GFO entities and appendix A - the list of all GFO constructs used in the current work, their definitions and corresponding axioms. The account of the GFO entities given here slightly differs, mostly terminologically (although see footnote 5) from the most current specification of GFO [Heller et al., 2006], which was under construction during the time of writing of the present work. Moreover, due to the topic of the present study, which does not require an introduction of GFO in all its complexity, some of the notions of GFO are simplified here.

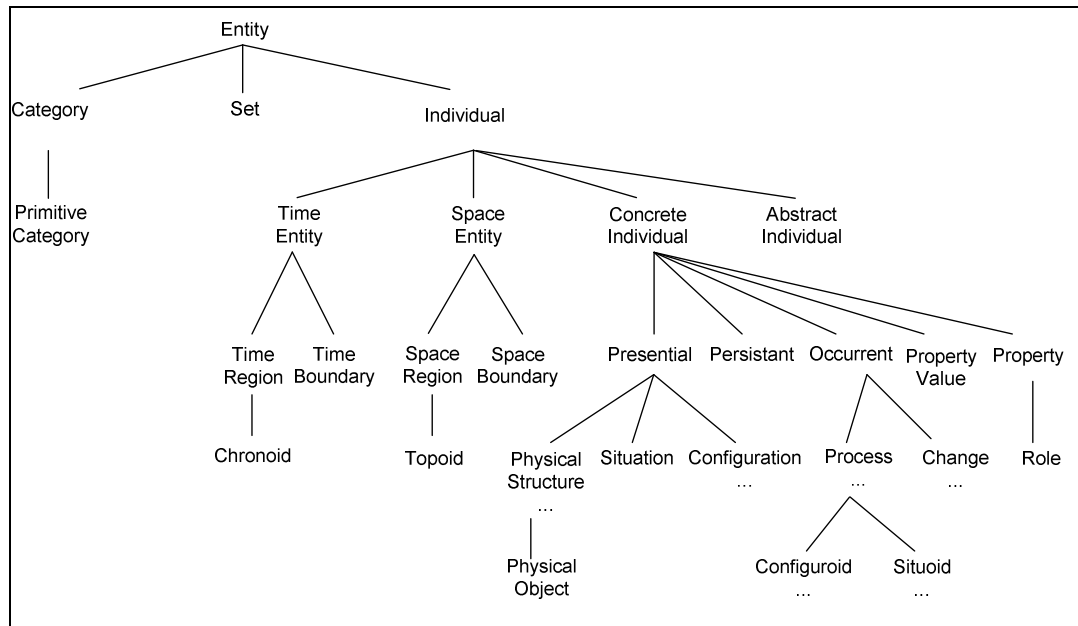


Figure 4. Taxonomy of GFO topmost-level categories. This tree is a simplified and modified variant of ([Heller et al., 2005], p. 145-147).

Concrete individuals, in contrast to *abstract individuals*, are the individuals located in time and space. The main distinction is made between *occurents* and *presentials*. The former include processes being entities that are extended in time and resemble processes in SUMO and Sowa's ontology, or occurrences in DOLCE, whereas the latter are not extended in time, and in contrast to endurants, they do not persist through time but are time-flat. They can be considered as time slices of processes. Persistence is accounted in GFO by means of *persistants*, which are entities extended in time but distinct from processes. A persistant can be seen as a construct which binds presentials having the same identity, though located at different time boundaries⁵, into one entity persisting through time.

GFO construct	Example
Universal	Human being
Individual	John
Chronoid	Time of duration of a soccer match
Chronoid boundary	Time of the end of the match
Topoid	3D location occupied by John

⁵ It should be mentioned that the presented account of persistants is a simplified variant of GFO [Heller et al., 2006]. Originally, persistants are not individuals but universals instantiated by ontically connected individual presentials, and are introduced to GFO as a response to the problems yielded by the typical understanding of endurants as the entities enduring in time and wholly present at every moment of their existence. For further discussion see ([Heller et al., 2006], p. 25).

Process	Movement of the ball
Presential	The ball at the beginning of the match
Persistent	The ball
Property	The color of the ball
Property value	White
Role	John as an attacker in the soccer match
Fact	John hitting the ball
Situoid	Soccer match

Table 1. Examples of GFO entities.

More complex entities composed of collections of presentials and their interrelations are handled by the notions of *facts*, *configurations* and *situations*. The time-extended counterparts of the latter two are called *configuroids* and *situoids*, respectively⁶.

Individuals of all kinds may have assigned *properties* and *property values* which are considered to be dependent on their bearers. Similarly *roles* are considered to be entities dependent on their *role fillers* and the *context* in which they are played by the role fillers⁷.

Entities of GFO are related by a net of ontological relations, including *instantiation*, *has-quality*, *part-of*, *occupation*, and others. Several other relations, including *causality*, are still under development [Michalek, 2005; Michalek, 2006].

1.2.3 Representation Formalisms

Various types of logics have commonly been used for representing ontologies, e.g. FOL in the case of GFO and Sowa's ontology or modal logic [Hughes, Cresswell, 1996] in DOLE. However, representing ontologies in a purely logical manner is not a straight task, especially for untrained domain experts, who are often involved in ontology development process. To simplify ontology design a number of ontology specification languages have been developed, merging familiar frame-oriented and object-oriented paradigms with logic, for example F-Logic [Kifer, 1995], OCML [Domingue et al., 1999], or LOOM [Loom, 1995].

Those languages are sometimes referred to as *traditional* ontology representation languages, and are contrasted with *web based* languages, which are dedicated to ontology

⁶ Complex entities are discussed in more detail in section 3.3.7.

⁷ Roles are discussed in more detail in section 3.6.1.

representation in the context of the Semantic Web [Corcho, 2001; Corcho, Gomez-Perez, 2000]. Web-based ontology languages are intended to provide machine-readable semantics of the content on the web. Those languages are either HTML- based, e.g. SHOE, or XML-, RDF- and RDF(S)- based, e.g. OIL, DAML +OIL or OWL. The current World Wide Web Consortium [W3C, 2006] recommendation for an ontology specification language on the Web is the Web Ontology Language (OWL) [OWL, 2004], which comes in three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The first two sublanguages have the formal semantics of SHIF(D) and SHOIN(D) description logics, respectively. OWL Full uses all OWL language primitives and permits arbitrary combination of those with RDF and RDF(S). However, the high expressivity of OWL Full is achieved at the cost of its computational intractability, which is a serious shortcoming especially in the light of the aim of the Semantic Web.

Yet another group of the languages for representing ontologies are the languages constructed for the development of (and based on) particular ontologies, e.g. CycL developed in the context of the Cyc project. Those languages often have some build in ontology.

Finally, there are also proposals to use as ontology specification languages the graphical modeling languages from outside of the ontology community. For example, it is suggested in e.g. [Baclawski et al., 2001; Kogut et al., 2002] to apply the Unified Modeling Language (UML) [Rumbaugh et al., 1999], developed and successfully applied in the software engineering area, into ontological engineering. The issue of combining ontologies with conceptual modeling languages, in particular with UML, is discussed in chapter 8. Moreover, chapter 8 contains a specification of the UML profile constructed by means of the developed ontology applicable to functional modeling.

1.2.4 Applications

The list of application areas of ontologies has increased in recent years. [Guarino, 1998; Corcho et al., 2001] report a few of them, including: knowledge engineering, knowledge management and knowledge representation, qualitative modeling, language engineering, database design, e-commerce and e-services, information modeling and integration, database design, natural language processing, knowledge reuse and of special impact today the Semantic Web. The upper-level ontologies in turn bring profit in particular to integration of information, reuse of knowledge, domain ontology engineering, and conceptual modeling.

Concerning the first, since ULOs provide the specification of the most common categories their application is not limited to particular applications, tasks or domains but can be used and shared in various different information systems. This permits to apply ULOs for

information integration, providing a commonly used set of the most basic notions. Domain ontologies may be merged and integrated by virtue of commonly shared top-level concepts.

Secondly, top-level ontologies reduce the time and effort of the domain ontology design and alignment. Instead of developing a structure of most top-level categories and relations individually for each domain ontology one may use an available top-level ontology. In this sense, a top-level ontology can be used as a framework for developing domain ontologies. Moreover, ULOs applied as the foundations of domain ontologies improve their quality and help in their formalization. As reported in [Borgo, Leitão, 2004] domain ontologies developed in frames of ULOs gain the benefit of strong formal and semantic foundations. We see therefore that the use of ULOs not only precipitates and simplifies the process of the creation (and the maintenance) of domain ontologies but also leads to fewer errors and a better understanding of the domain concepts.

Thirdly, and particularly importantly for our work, ontologies have been recognized to be adequate for providing sound foundations for conceptual modeling. Conceptual modeling is concerned with the construction of computer-based semi-formal or formal abstractions of part of the world. In particular, conceptual modeling is used in software engineering for the purpose of depicting both the domain of interest - the information structure on which the system under development is intended to operate, i.e. the data model, and the components of the system itself. In software engineering conceptual models are developed by means of so-called conceptual modeling languages which are typically formal or semi-formal diagrammatic languages or notations, such as Unified Modeling Language [OMG, 2005], Entity Relationship diagrams [Chen, 1976], or Object Role Modeling notation [Halpin, 1997]. Usually those languages are based on a limited number of constructs, which are sometimes defined in the form of a meta-language, e.g. UML.

Many conceptual languages, although successfully used in common-day practice of software engineering, lack a rigorous definition of used constructs, and often their ontological correctness is doubted (see e.g. [Wand, 1999; Guizzardi et al., 2002a; Guizzardi et al., 2002b; Guizzardi et al., 2004]). Those limitations may result in the ambiguities and even incorrectness of the conceptual models developed by means of such languages. It seems that top-level ontologies, which are formal axiomatic theories can provide sound foundations for conceptual modeling languages. ULOs provide both formally defined and semantically justified categories, which can serve as the foundation for the definitions of conceptual modeling languages constructs.

In addition, one could observe that ULOs resemble conceptual modeling languages when comparing the categories of the former with the constructs of the latter. For example, the common ontological categories shared by many ULOs (see section 1.2.2) such as object,

process, time, activity, role are the constructs of conceptual modeling languages, e.g. UML or Object Process Methodology (OPM) [Dori, 2002]. In this sense ULOs could not only be applied as the foundations of the conceptual modeling languages, but they can themselves be directly applied as conceptual modeling languages of high formal and ontological precision.

Today's obstacle in applying top-level ontologies directly as conceptual modeling languages is the lack of an appropriate representation. Conceptual modeling languages are mostly pictorial representations, which are intuitive not only for modelers but also for domain experts involved in the process of conceptual modeling. Top-level ontologies, on the other hand, are formal logical theories, not easily comprehensible for untrained users, i.e. for domain experts involved in the process of conceptual modeling. Thus, it seems profitable to develop pictorial (and simple) representations of top-level ontologies which may be used as the tools for conceptual modeling. For that purpose one could for instance extend the current modeling languages by means of build-in extension mechanisms, such as UML Profiles [OMG, 2004]. This technique was applied in [Guizzardi et. al., 2002; Guizzardi, 2005] where GFO and UFO were used for the extension of the UML structural meta-model. Extending UML with a top-level ontology gains the twofold benefit of providing a pictorial representation of top-level categories based on the well known syntax of UML, and secondly of bringing formal and ontological sound foundations for UML.

Note that the idea of developing a UML profile for a top-level ontology should not be confused with the initiative of developing a UML profile for ontology development, as it is proposed in example in [Kogut et al., 2002]. In the latter UML is intended to be used as a language for modeling ontologies, whereas the former is aimed to extend UML with (top-level) ontologies.

1.3 Objectives

Now, after the preliminary remarks concerning our understanding of function and the outline of top-level ontologies we are ready to introduce the problem and the objectives of the current work.

Problem: Function is an important cross-domain notion and there is a lack of a domain independent framework for representing and modeling functions.

The examples presented at the beginning of this chapter show that the notion of function is of importance in various domains. However, as we will demonstrate in the next chapter, most of the research done in AI and computer science in recent years seems to be concerned with the

representation of functions in the context of the design of technical devices and computer systems only.

There has been a recognized need to provide a general framework for representing functions especially in the context of a multidisciplinary design [Chandrasekaran, Josephson, 2000; Rosenman, Gero, 1999]. As Chandrasekaran and Josephson point out, the lack of such a framework causes that “the representation of function in one domain, say chemical engineering, may not be compatible with the definition, say, in electrical engineering”([Chandrasekaran, Josephson, 2000], p. 1).

We follow the idea of a generally applicable notion of function, but we extend it beyond the technical device design. We think that the notion of function can be extended in such a way that it coherently covers not only the design of non-technical artifacts, i.e. in the field of business modeling, but moreover it is applicable to describing non-artifacts, e.g. biological organisms or social entities. In our opinion such a general approach would permit a true multidisciplinary functional knowledge representation. For example, if by means of the same formalism one can represent the functionality of the human organism and the functionality of technical artifacts, then the multidisciplinary design of implants could be simplified. In the same way, one common representation of the functions of software, technical artifacts and the business systems composed of them would simplify the business modeling and the integration of all three. The current work is aimed at solving the defined problem by reaching two objectives:

Objective 1. To develop a formal top level ontology of functions applicable across various domains.

In the current work we propose to consider the notion of function not as a domain notion of technical artifact design, but instead as a top-level category analogous as time, space, process, object etc. Thus, in our understanding the ontology of function is not considered as a kind of mediating ontology between top-level and domain level, as is proposed for example in [Kitamura, Mizoguchi, 2004]. Nor is it considered as a domain ontology for technical engineering as in [Chandrasekaran, Josephson, 2000], but it is understood as a top-level ontology instead (see figure 5 for comparison).

However, the current top-level ontologies do not include the notion of function, or treat it cursorily. For example DOLCE, GFO and Sowa’s ontology lack the notion of function or any correlated notion. SUMO introduces the *hasPurpose* relation, which has the meaning that a physical thing has a desired or expected purpose. The notion of a purpose is distinguished from the notion of an outcome, which does not need to be expected or desired. The intended purpose in SUMO could be interpreted as a function of an entity. However, purposes in SUMO

are assigned to physical objects only, whereas it seems that non-physical entities may have functions ascribed as well. In Sowa's ontology the concept of purpose is also present, however it is not considered as a function but as the relation gluing an agent, his act and his intention concerning that act.

Concluding, we see that the need for supporting functions and functional modeling by the currently developed top-level ontologies is not satisfied. Thus, the aim of this work is to provide the domain-independent, top-level ontological framework for function representation. Such a framework in our opinion is not limited to providing the standalone category of functions but is rather concerned with the exploration of the conceptual structure of functions, which includes the web of categories and their interrelations.

However, our aim should not be seen as the replacement of the domain-specific notions of functions but rather as an attempt to develop a general notion covering the greatest number of specific cases. A list of detailed requirements of the function ontology is composed on the basis of an analysis of the current state of the art and is presented in section 2.4.

Objective 2. To incorporate the top level ontology of functions into a wider ontological framework.

A functional knowledge seldom stands in isolation, but mostly it is combined with a non-functional knowledge. Thus we find it important not to develop the isolated representation of functions, but to embody it in a wider ontological framework. Hence, the functional ontology is designed as a module of GFO and is formalized in first-order logic, in conformance with the existing axiomatization of GFO presented in [Heller et al., 2005].

Additionally we think that the ontology of functions can provide sound foundations for conceptual languages for function modeling. Therefore we recognize an additional objective which is considered rather as a by-product of the developed ontology:

Objective 3: To provide a ready-to-use modeling formalism permitting the ontologically-driven conceptual modeling of functional knowledge.

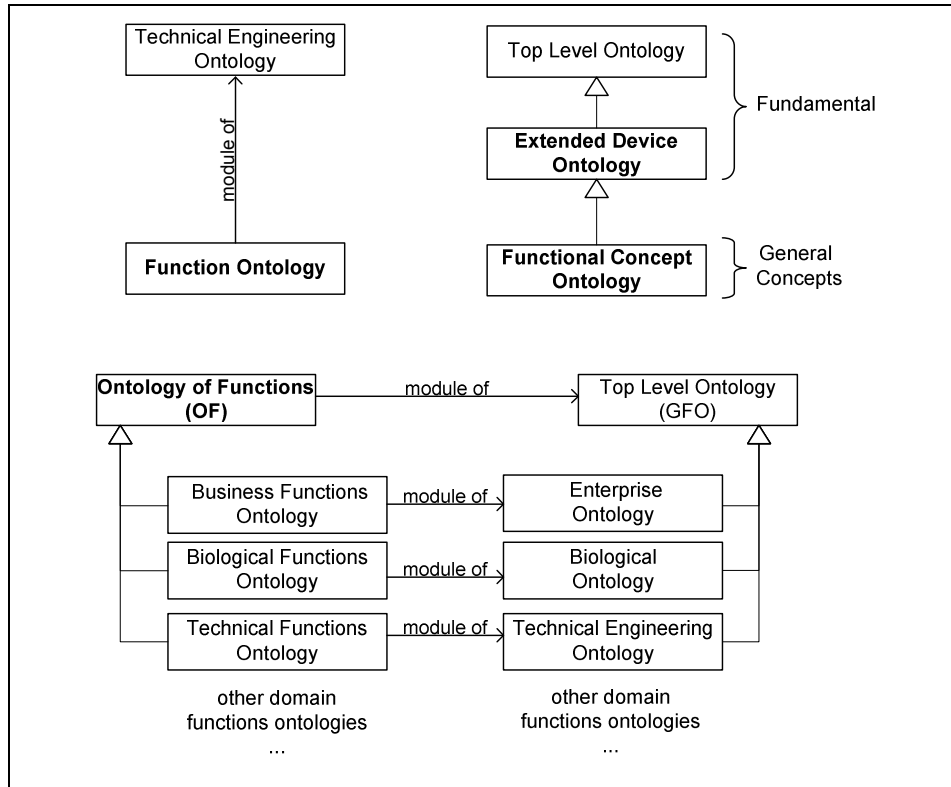


Figure 5. Three architectures of the functional ontology. The top left part of the figure represents the architecture underlying the approach of [Chandrasekaran, Josephson, 2000], where the ontology of functions is considered as (a module of) a domain ontology for technical engineering. The top right of the figure represents the architecture underlying the approach of [Kitamura, Mizoguchi, 2004]. The functional concepts are defined in the Extended Device Ontology, which is a specialization of Top-level Ontology from the device-centered view, and the Functional Concept Ontology specifies functional concepts as an instance of the concept of “function” defined in the Device Ontology [Kitamura, Mizoguchi, 2004]. The above diagram is the interpretation of the architecture, the original diagram of this architecture is depicted in figure 6 of the current work. The bottom part of the figure presents the architecture underlying our approach. Here, in contrast to the two previous architectures the ontology of functions (OF) is neither considered as the domain ontology nor as the fundamental (but nevertheless domain) ontology of devices. Instead it is a top-level ontology, being a module of a wider ontological framework. Domain function ontologies are considered as specializations of OF.

In our opinion the current formal top-level ontologies could be used for the purpose of conceptual modeling as formal, precise and well-founded modeling languages. However, they lack an appropriate graphical representation which is a typical part of conceptual modeling languages nowadays. We aim to provide the graphical representation of the developed ontology of functions. In particular, on the basis of the developed ontology we construct a UML profile for function modeling. Using a UML profile which is an extension mechanism of UML not only serves this purpose but it also makes the developed representation compatible with the current de facto standard for conceptual modeling.

1.4 Structure

The remaining part of the current work is structured into eight chapters. In chapter 2 we analyze the notion of function across domains and formalisms. In particular we investigate AI approaches from the field of functional device modeling, as well as the conceptual modeling paradigms. In addition we briefly outline the discussion concerning functions in the field of philosophy, especially in the context of the philosophy of biology. The results of those analyses are used to shape the list of the issues which the ontology of functions in our opinion should address.

Chapters 3 to 7 present the Ontology of Functions. Chapter 3 specifies the structure of function, lists function determinants and introduces most fundamental kinds of functions. In addition, it addresses the issues of a function's side effects and restrictions on functions. In chapter 4 several relations by which functions can be glued into functional models are defined. Some of those relations are typical ontological relations such as subsumption, or partonomy, adopted for functions; some others are specific for functions relations. The issue of function realization is discussed in chapter 5. It contains the specification of different modes or realizations, which exceeds most typical cases of processual realization. The results of chapter 5 serve as the basis for the distinction of various types of function ascription addressed in chapter 6. Moreover, chapter 6 introduces the notion of malfunction, crucial in the context of artifact evaluation. Chapter 7 contains a discussion on the ontological status of functions, in which the characteristics of functions as well as some of the candidates for function definition are considered. In addition, it provides the most top-level categorization of functions constructed on the basis of the developed framework. Finally, it specifies a modularization of the developed ontology.

Chapter 8 provides a specification of the UML profile constructed on the basis of the developed ontology and suited for functional modeling.

The study ends with conclusions containing the identification of the advantages of the developed framework, its applications in domain ontologies and conceptual modeling as well as suggestions for future work. In turn, definitions and symbols of the GFO notions used in the text are provided in appendix A.

2 Related Works

The purpose of the current chapter is double. On the one hand we give an overview of the notion of function across domains. In particular in section 2.1 we investigate functions in areas of computer science such as device representations and design, in section 2.2 software engineering and business modeling. In addition, in section 2.3 we briefly survey philosophical discussion over functions. On the other hand, on the basis of the state of the art discussed we provide in section 2.4 a list of requirements which in our opinion a top-level ontology of functions should meet.

2.1 Functional Device Representations

In AI there is a vast area of research concerned with the functional representation in the context of technical artifact design. The design is concerned not only with the problems of structural and behavioral organization of artifacts, but with the intentions of designers concerning the functionality of artifacts, i.e. those concerning the problems of ‘why a given component is here, and why the artifact behaves in a certain way?’. [Iwasaki, Chandrasekaran, 1992] distinguish a function-oriented and a behavior-oriented approach to modeling. The former is focused on the question of *what* a device is supposed to do, i.e. its function, whereas the latter is concerned with the expected behavior of a device and deals with the question of *how* a device is supposed to achieve an intended function.

There are a number of approaches in artifact modeling that handle functional knowledge. They differ from one another not only due to their interpretation of the notion of function, for which a universally accepted definition has not been developed so far, but also in their underlying purposes, ranging from design verification, e.g. [Iwasaki, Chandrasekaran, 1992] to automatic functionality identification [Kitamura et al., 2002]. Despite those differences in the current work we will consider them all under one common notion of the *functional device representations* (FDR). A typical feature of FDR approaches is that functions are considered in the context of devices and are contrasted with the behavior and structure of a device.

In the coming sections we will review the main interpretations of the notion of function introduced in the field of FDR. These are:

- Function as the input-output pair.

- Function in the context of behavior.
- Function as the intended role.
- Function as the effect.

Although the list of references is not exhaustive, yet to our knowledge most of the approaches in the FDR field can be classified under at least one of the above interpretations.

2.1.1 Functions as Input-Output Pairs

In design research and in functional device representations the input-output approaches have a long tradition, see e.g. [Rodenacker, 1971]. As reported in [Britton et al., 2000] the input-output approaches define functions by means of rigorous mathematical description e.g. [Schmekel, 1989; Welch, Dixon, 1992]. Here, however, we are more interested in the underlying conceptual model. Let us consider two examples: [Rodenacker, 1971] defines a function as a relationship between input and output of energy, material and information, while in [Wirth, O’Rourke, 1993] function is defined by the input flux, the output flux, the source component, the destination component and the function carrier. For example, the function of a hot-cold water faucet can be understood in those terms as follows: the input fluxes are hot and cold water, while the mixture of both is the output flux. The person controlling the water flow is the source component. The destination component reflects the purpose for which the output flux serves. In case the output flux is used for washing hands, the hands are the destination component. The faucet is the function carrier⁸.

Among the limitations of the input-output approaches, according to [Britton et al. 2000], is the fact that mostly they focus only on useful outcomes and ignore negative side effects. In turn, [Rosenman, Gero, 1998] admit the applicability of that solution for the purpose of mechanistic aspects of functions, however they argue that it does not fit well with the social view of functions. The authors write: “there is confusion when function is used at one time to describe ‘telling the time’ and ‘transforming an analog signal into digital signal’” ([Rosenman, Gero, 1998], p. 170). Moreover, it is questionable whether for all functions such a mechanical transformation can be found, e.g. it is hard to find mechanical transformation between the input and the output in the function of a bolt and a nut which is to fix parts [Umeda et al., 1990].

⁸ The example cited after [Qian, Gero, 1996].

2.1.2 Function and Behavior

In FDR functions are often considered in reference to behavior. Here, we overview several approaches which consider functions in the context of behavior, namely: Functional Concept Ontology [Kitamura et al., 2002] (called also a Functional Ontology [Kitamura, Mizoguchi, 1998]), Functional Representation [Sembugamoorthy, Chandrasekaran, 1986; Chandrasekaran et al., 1993; Chandrasekaran, 1994a; Chandrasekaran, 1994b], Causal Functional Representational Language [Iwasaki et al., 1995], Function Behavior State [Umeda et al., 1990; Umeda, Tomiyama, 1995], and finally Bonnet's [Bonnet, 1992] and Salustri's approaches [Salustri, 1998; Bo, Salustri, 1999]. We give a short overview of those approaches with a special emphasis of the understanding of the notion of a function.

Functional Concept Ontology

Kitamura et al. developed a framework for modeling functional knowledge comprising the hierarchy of functional knowledge [Kitamura, Mizoguchi, 2004] and the language for modeling functions and behaviors of a device called the Function and Behavior Representation Language (FBRL) [Sasajima et al., 1995]. For our interests, crucial in this hierarchy are the Top-level Ontology, the Functional Concept Ontology and the Extended Device Ontology (figure 6).

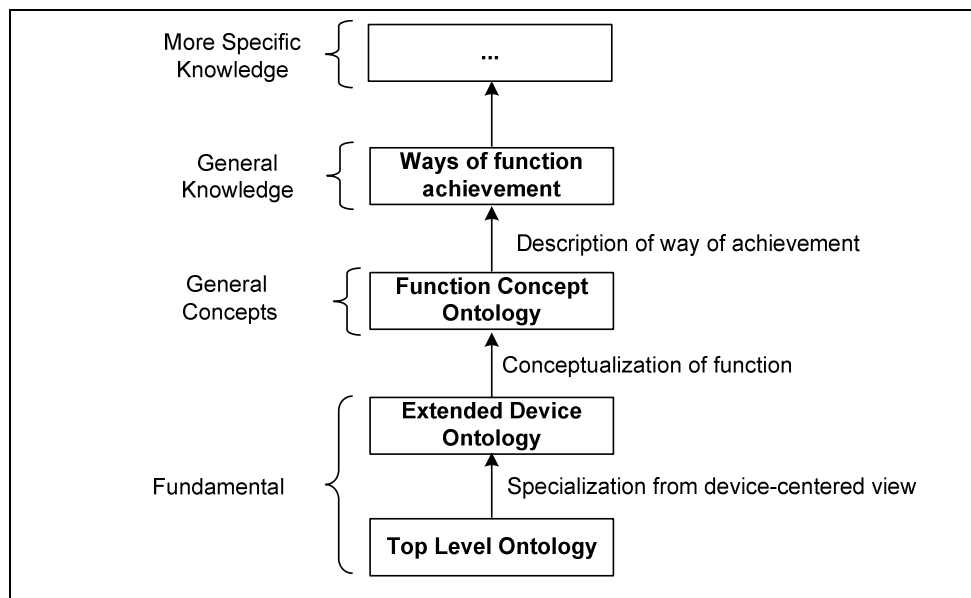


Figure 6. Part of the hierarchy of functional knowledge (from [Kitamura, Mizoguchi, 2004]).

The Top-level Ontology, which is currently under development, is intended to provide the fundamental vocabulary, which in turn is specialized to Extended Device Ontology. Extended

Device Ontology provides a common device-centric viewpoint for representing artifacts, it also defines the concept of function. The Functional Concept Ontology specifies functional concepts and depends on the Extended Device Ontology. Thus, already the architecture of the hierarchy shows that the notion of function is not a top-level concept but is dependent on (and restricted to) the notion of a device. The knowledge about a device is organized in three levels ([Kitamura et al., 2002] p. 146-8):

1. Behavioral layer
2. Base-function layer
3. Meta-function layer

The behavioral layer is provided by the Device Ontology and provides the objective description of the structure and behavior of a device. The functional levels (base-function layer and meta-function layer) provided by the Functional Concept Ontology are founded on the notion of behavior.

A base-function is defined as a teleological interpretation of B1 behavior, whereas B1 behavior is considered as a change of an attribute value of an operand from that at the input port of a device to that at the output port of the device [Sasajima et al., 1995]. We see that on the one hand a function is distinguished from the structure and behavior of a device, but on the other hand it is restricted to a particular behavior, namely B1 behavior, and is secondary to it.

A base-function is associated with the behavior realizing it by the mapping primitives called the Functional Toppings. Four kinds of functional toppings are distinguished: Obj-Focus, O-Focus, P-Focus, and Necessity. They represent respectively the object, its attributes and its ports, on which a function focuses. In this sense a function is the change of the value of a focused attribute in given input and output ports of a given object. The Necessity specifies whether a focused object is used in another component of a device.

Functional toppings show that the specification of a function is grounded both on the behavior and the structure and the environment of a device.

Base-functions are glued by several kinds of relation, among them *is-a* and *part-of*. The *is-a* relation is considered in terms of the intensional subsumption, e.g. the function `to remove` is defined as the function `to take` plus the condition `heat is unnecessary` ([Kitamura et al., 2002], p. 149) and thus it is a specialization of `to take`. The criteria for delimiting the sub-function are explicit and are given by the functional toppings.

The second kind of relation introduced into FCO is called *a method of function achievement* and is understood as a part-of relation between functions. One function, called a macro-function, is said to be achieved by its micro-functions. Micro-functions are considered to be parts of macro-functions, in this sense *a method of function achievement* corresponds to

the functional decomposition [Pahl, Beitz, 1988], the degree of complexity in [Hubka, Eder, 1998], and the part-whole relation in Multilevel Flow Modeling [Lind, 1994].

The third kind of relation involving functions in FCO is called the *is-achieved-by* relation (or the *way of function achievement*) and relates a base-function with the essential property of a structure and a behavior that achieves that function. Thus the is-achieved-by relation mediates, just as functional toppings the behavioral with the functional layer.

Apart from base-functions FCO introduces meta-functions representing a role of base-function, called an agent-function, for another base-function, called a target function ([Kitamura et al., 2002], p. 149). Meta-functions are organized on the third layer of functional knowledge.

The FCO framework has a wide scope of applications in the context of device modeling ranging from the explanation generation to the automatic identification of functionality of a device. However, we find several limitations of the framework when applied in a wider context.

Firstly, it does not support functions concerned with non physical entities. Since this approach is a domain approach for technical artifacts design, it does not consider functions not related to physical entities, such as cognitive or social functions.

Secondly, it does not support the functions of all types of physical objects but is restricted to technical devices only. Functional Concept Ontology specifying functional knowledge is not a top-level ontology but is dependent on the domain ontology of technical devices (see figure 6). That architecture itself imposes the domain restricted character of the notion of function in FCO.

Thirdly, even in the case of functions operating on physical objects it seems that not all function types are supported, in particular such that do not involve change. The notion of function in FCO is restricted to dynamic functions involving a change of physical attributes only. For example, if the function of a warehouse to store goods is considered, one would see that there is no B1 behavior involved in this function, since there is no change of the attributes of goods kept in the warehouse. In this sense a function of a warehouse cannot be represented as a teleological interpretation of behavior involving a change of an entity on which it operates. Thus we see that the framework is rather domain-oriented – it supports modeling only of dynamic functions involving changes of physical operands.

Fourthly, it seems that the distinction between base- and meta-functions is also dictated by the domain of application. The framework as it is presented by the authors enables the representation of functionality of devices; but it is not a framework for representing functions in general. Therefore of primary relevance are the functions assigned to devices or their components, whereas functions assigned to base-functions are considered on the meta-level.

FR and CFRL

In [Sembugamoorthy, Chandrasekaran, 1986] authors introduced the Functional Representation framework (FR) which was elaborated in a number of papers [Chandrasekaran et al., 1993; Chandrasekaran, 1994a; Chandrasekaran, 1994b] and was extended in [Iwasaki et al., 1995] to the Causal Functional Representational Language (CFRL). Both formalisms have been applied to a number of tasks, which is reported in [Chandrasekaran, 1994a], including device design [Chandrasekaran et al., 1993], design and redesign verification [Iwasaki, Chandrasekaran, 1992], diagnostic reasoning, computer program debugging [Allemang, 1991; Allemang, Chandrasekaran, 1991] and simulation [Pegah et al., 1993].

In FR a function is defined as a quintuple $\{Type_F, P_F, Dev_F, C_F, G_F\}$, where:

- $Type_F$ is one of the function types introduced in [Keuneke, 1991]: *ToMake*, *ToMaintain*, *ToPrevent*, *ToControl*.
- P_F is the functional goal.
- Dev_F is the device that the function is a function of.
- C_F are the conditions which specify when the function must be achieved;
- G_F is the set of Causal Process Descriptions (CPD) describing the causal mechanism achieving a function ([Iwasaki, Chandrasekaran, 1992], p. 6).

This definition evolved in CFRL where a function F is defined as a quadruple $\{E_F, D_F, C_F, G_F\}$ where:

- E_F is the name of the function which F elaborates.
- D_F is the description of the device of which F is a function
- C_F is the description of the context in which the device is to function.
- G_F is the description of the functional goal to be achieved ([Iwasaki et al., 1995], p.12).

The functional goal G_F of the latter definition is the combination of the functional goal P_F together with the set of Causal Process Descriptions G_F from the former definition. Thus, both definitions represent the function by references to the Causal Process Description (CPD). CPD is a directed graph with two distinguished node types, N_{init} and N_{fin} [Chandrasekaran, 1994b]. Each node of the graph represents a partial state of a device. N_{init} corresponds to the partial state of the device when the conditions of the function are satisfied. N_{fin} corresponds to the state, in which the goal of the function is achieved. The edges of the graph represent causal or temporal connections between the states.

CPD is an abstract description of a device's expected behavior in terms of a sequence of events. For example the CFRL definition of the function of an electrical power system in an

Earth orbiting satellite contains a very simple CPD (figure 7), composed of three nodes related by causal links. If the voltage exceeds 33.8, the relay is opened by the controller which causes a decrease of the battery's stored charge. In addition, qualifiers and annotations attached to the edges indicate the conditions under which the transition takes place as well as the type of the causal explanation of the transition. Here, the transition from $n1$ to $n2$ is causal and is achieved by the function of the controller component.

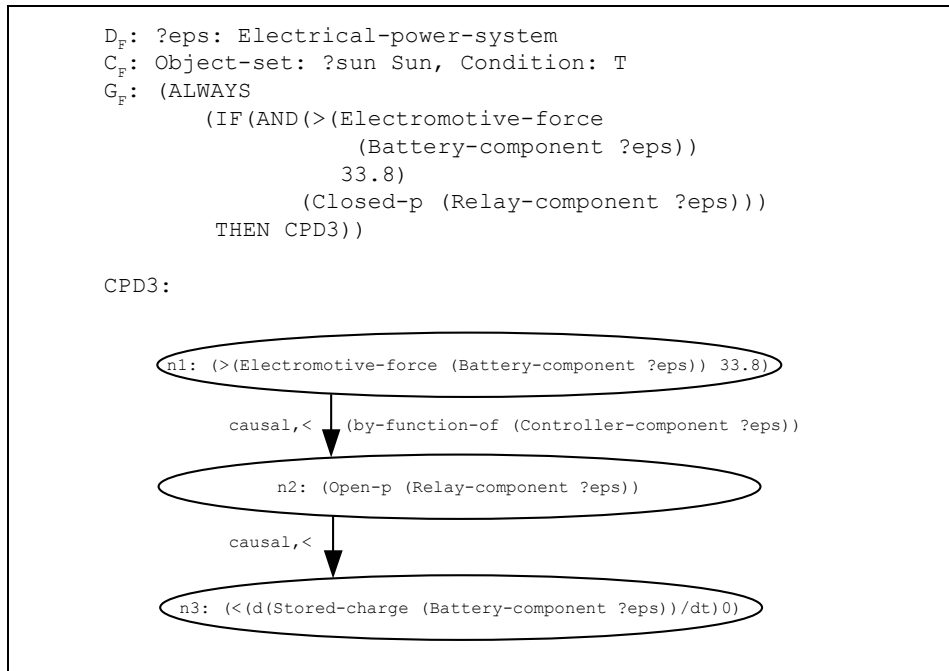


Figure 7. CFRL definition of function of an electrical power system ‘to prevent the battery from overcharging’ (from [Iwasaki et al., 1993]). D_F specifies the device - the electrical power system, C_F - the context of the device, i.e. the Sun and Conditions, G_F - the goal to be achieved, which is a conditional statement here saying that if the battery voltage exceeds 33.8 and the relay is closed then CPD3 should be executed. Finally CPD3 provides the chain of causal processes realizing the function.

The definitions of function both in FR and in CFRL involve the causal description of the behavior of a device, which is a mechanism of the function's realization. Umeda in ([Umeda, Tomiyama, 1997], p. 44) observed that in fact CFRL (and so FR) does not involve the explicit specification of *what* the system is intended to do, but instead it specifies only *how* it realizes its function, i.e. the intended function of the electrical power system to prevent the battery from overcharging is not included in the functional specification.

Moreover, since FR and CFRL relate function to behavior, it makes it problematic, just as in the case of FCO, to represent by their means functions not involving a behavior. Not all functions are explained in terms of behavior, e.g. the function of the external wall of the building to support roof is not considered in terms of its behavior, i.e. by the description of the state changes the wall is involved in. Instead, the wall's function is typically

explained by the structure of the wall. Functions of that kind are not a result of behavioral features, but rather follow directly from the structural properties. They are called “passive” in contrast to the behavioral, “active” functions [Keuneke, 1989]⁹.

Finally, FR and CFRL, just as FCO, define functions by means of devices. It restricts the definition of function to the definition of the function of a particular device and does not permit to model functions independently of the realizing them entities.

Bonnet’s Approach

Bonnet in [Bonnet, 1992] proposes to define functions as intended activities. He defines function in the following way:

“According to person P (designer or user) the function of device D in system S_T is *to do A*’ is interpreted as:

- P has a goal G
- P believes that:
 - b_1 : D will perform activity A in the future
 - b_2 : $A \Rightarrow_T G$, which reads: activity A causally entails the attainment of G according to theory T .”([Bonnet, 1992], p. 2)

According to Bonnet a function of device D is a subjective notion of some person P , who is the designer or the user of device D . Moreover a function of D is relativized to the system in the context of which D is viewed. It resembles Mode of Deployment (MoD) in [Chandrasekaran, Josephson, 1997], which is discussed in section 2.1.4 and correspond to the approach of Cummins discussed in section 2.3.1.

A function is represented by the form *to do A* where A is an activity contributing to the achievement of an intended goal. In this sense a function is identified with an activity (behavior). This approach is thus even more radical in identifying function with behavior than that of Kitamura, who considered a function to be *an abstraction* of behavior.

The reduction of a function to an activity, or to a certain belief about an activity, makes it impossible to represent passive functions, just as in the previously discussed approaches.

The inclusion of a person’s beliefs stresses the subjective and intentional character of a function. On the other hand it raises two questions:

⁹ Chandrasekaran in [Chandrasekaran, 1994b] does admit that passive function can have a behavioral descriptions but points out that they are normally not explained that way but by reference to their structure.

1. Should only the intentions of a designer or a user be considered? It seems that not only those two persons influence the function of an artifact, but also e.g. stakeholders dictating requirements or researchers exploring the artifacts could also be taken into account.
2. If a function is understood in terms of designer/user intentions it raises problems in understanding functions of non-artifacts, which lack a designer and often also a user. A similar problem concerns the ontology of Chandrasekaran and Josephson [Chandrasekaran, Josephson, 1997] discussed below (see section 2.1.4).

Function Behavior State

Umeda and colleagues [Umeda et al., 1990; Umeda, Tomiyama, 1995] developed a framework for representing functions called Function Behavior State modeling (FBS_{state} ¹⁰). FBS_{state} was implemented in a tool called FBS Modeler, which is reported to be successfully applied in the practice of device design [Umeda, Tomiyama, 1997].

In FBS_{state} a function is defined as “a description of behavior recognized by a human through abstraction in order to utilize it” ([Umeda, Tomiyama, 1995], p. 271). This resembles the definition of function adopted in FCO. However, the representation of function in FBS_{state} is different then in FCO. Here, a function is represented as a tuple (f_{symbol}, b) where f_{symbol} denotes a functional symbol in the form of the natural language expression “to do something” and b denotes behavior that realizes that function.

The representation of function in the form “to do something” is a common practice in Value Engineering, which however, does not permit to give a clear semantics of functions. That in FBS_{state} is given by behavior b . Behavior in FBS_{state} is defined by sequential changes of states over time and is always considered in some context (called aspect), which comprises entities, attributes, relations, physical phenomena and time of the current interest.

Although FBS_{state} is intended to represent functions independently of behavior the semantics of functions is given by the description of behavior. A function is defined as a relation between a functional symbol (a function) and a behavior realizing it. For example, the relation of functional decomposition of a given function f_0 as introduced in ([Umeda, Tomiyama, 1995], p. 275) involves not only the functional symbol of f_0 but also the corresponding behavior b_0 . Thus the function f_0 is decomposed into subfunctions f_1, f_2, \dots, f_n and

¹⁰ The Function Behavior State framework has the same abbreviation as the Function Behavior Structure framework discussed later. In order not to confuse the reader in the current work we abbreviate the former to FBS_{state} while the latter to $FBS_{structure}$.

embedding them behaviors b_1, b_2, \dots, b_n . In this sense functions in $\text{FBS}_{\text{state}}$ are inseparable from behaviors realizing them.

Since a function is represented as a relation between the functional symbol f and behavior b , it raises the question about the nature of the functional symbol: what does the functional symbol represent if not a function? On the other hand when considering the isolated functional symbol as a representation of function (which however violates the above definition of function) then a function is reduced to the natural language expression form and does not give any insight into its nature. It seems that $\text{FBS}_{\text{state}}$ relates intensionally considered functions represented by their functional symbols to behavior, rather than representing the structure of function as such.

Function Behavior State with Modifier

In [Takeda et al., 1996; Shimomura et al., 1995] $\text{FBS}_{\text{state}}$ is extended to $\text{FBS}_{\text{state}}/m$ ('m' stands for a modifier). In $\text{FBS}_{\text{state}}/m$ functions are not represented by a pair (f_{symbol}, b) but instead only by the functional symbol, extended by three additional notions: *function body*, *function modifier* and *objective entity*.

A function body is a symbol which carries the meaning of the function, typically it is a verb. For example, in the function to attach firmly backpack to a bike, the function body is to attach. An objective entity is an entity, which the function occurs on or to. Typically it is represented by a noun, here these are device, backpack and bike. A functional modifier is a symbol that restricts a function in order to match the functionality with designer's intentions. A typical functional modifier is an adverb, in our example these are precisely and firmly. A function body has no degree of satisfaction - it can be either satisfied or not. In contrast a functional modifier has degrees of satisfaction. For example, attach firmly may be achieved to different extents. Functional modifiers characterize how a function is achieved. Moreover, $\text{FBS}_{\text{state}}/m$ introduces several relations between functions, including the relation of function decomposition.

In $\text{FBS}_{\text{state}}/m$ the function is represented only by the functional symbol without reference to the behavior realizing. Thus, we see that the functional representation of $\text{FBS}_{\text{state}}/m$, in contrast to $\text{FBS}_{\text{state}}$ is realization-free, but on the other hand lacks any formal representation. $\text{FBS}_{\text{state}}/m$ provides an insight into the structure of function by the analysis of the natural language form representing a function.

Salustri's Approach

The notion of function is introduced in the context of behavior also by Salustri in [Salustri, 1998]. Salustri, however, in contrast to the formalisms reported above, does not define a function as a type or an abstraction of behavior, but instead he contrasts those two notions. Interestingly, contrary to FR, he defines behavior as *what* a device does and the function as *how* the behavior is achieved. He applies the analysis of why- and what-questions in order to delimit function from behavior. Salustri argues that, although the above understanding of function and behavior is at odds with the common intuitions in AI and software engineering literature, it is however consistent with the intuitions of many typical electro-mechanical designers. As an example he refers to the behavior of a mechanical part, which is commonly considered as its outward, measurable response to e.g. mechanical loads ([Salustri, 1998], p. 339). In Salustri's approach, as in FBS_{state} , functions are represented by means of verb object pairs. On the other hand he observes that behavior is represented analogously. This, in his opinion, is the reason for the frequent confusion of the two notions¹¹.

On the basis of the developed notion of function Bo and Salustri [Bo, Salustri, 1999] introduce a set of primitive functions understood in terms of basic interactions between mass, energy and information.

Conclusions

All the above-mentioned works show a strong correlation between the notions of function and behavior. In FR, CFRL, FCO, FBS_{state} , and Bonnet's approach the notion of function is grounded in the notion of behavior which makes it impossible to speak about functions independently of the behavior realizing it. Salustri, on the other hand, shows the contextuality of both and contrasts them on the basis of the analysis of what- and how-questions.

In our opinion the notion of function defined in the context of behavior reduces its scope to so-called dynamic functions, which involve actions and change the object on which they operate. Moreover, it does not permit to speak about functions independently of their behavioral realizations, which, however, is required in many situations, e.g. in the early phases of the design. Finally, functions defined in terms of behavior impose a particular behavioral realization, which make them realization-dependent and prevent from handling alternative realizations.

¹¹ We adopt the analysis of Salustri in chapter 5, where the relation of realization holding between functions is introduced.

In addition in FCO, FR, and CFRL functions are considered in the context of devices only, thus functions of non-devices, e.g. functions of organisms or organs, are per definition not handled. This cannot be considered a disadvantage of the device domain ontology but clearly it becomes one if one would like to consider it as a top-level solution.

2.1.3 Functions as Intended Roles

Another notion, beside behavior, in the context of which the notion of function is defined, is *role*. This viewpoint is represented, among the others, in Multilevel Flow Modeling (MFM) developed by Lind [Lind, 1990; Lind, 1994; Lind, 1999].

Multilevel Flow Modeling

Multilevel Flow Modeling (MFM) is a modeling methodology developed for the purpose of representation of structures, behaviors, goals and functions of complex plants. The representation of plants in MFM is based on decomposition principles established on the *means-ends* and *whole-part* relations (see figure 8). The distinction between means and ends permits to represent a system in terms of its components (structure, behavior), functions and goals. The structure and the behavior capture the causality of the system, whereas the function and the goal capture the intentional character of a system. Functions are thus not defined in the context of behavior or structure, but are purely intentional concepts that belong not to natural, but to social science.

According to the definition given in ([Lind, 1994]) functions represent the roles the designer intended a system to have in the achievement of the goals of the system(s) of which it is a part. Several types of flow functions have been distinguished: *mass* and *energy* functions specialized into *source*, *sink*, *storage*, *balance*, *transport* and *barrier*, and *information* and *action* functions specialized into *maintain*, *produce*, *destroy* and *suppress*. The classification of action functions is founded on the classification of basic actions by von Wright [VonWright, 1963].

Components and their behaviors realize functions, which in turn contribute to achievement of goals. In this sense components are *means* and the goals are the *ends* of a system. As Lind argues the ascription of function cannot be separated from the selection of goals. Beside the *achieve* relation ascribing functions to goals, and the *realization* relations between functions and the physical components realizing them, several other MFM relations are introduced, e.g. *condition*, the *goal-objective* relation or the *goal-subgoal* relation.

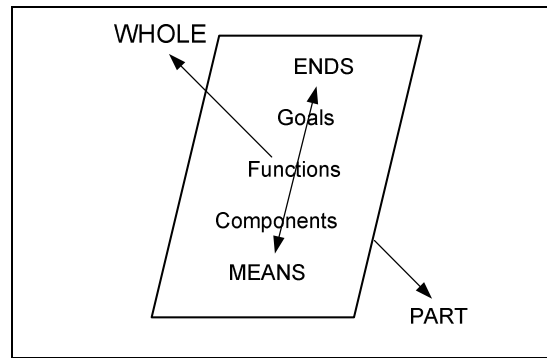


Figure 8. Types of plant decomposition used in MFM (from [Lind, 1994]).

Concerning the definition of function we first see that intentions of only the designer are taken into consideration, even though in the process of design not only designers are involved. For example in the process of software engineering requirements seldom come from a designer, but often from stakeholders or users. Therefore, as in the case of Bonnet's approach, one could perhaps think about other persons relevant for determining functions.

Secondly, in MFM the function of a system is always considered in the context of the whole, of which the system is a part. However, there are standalone systems which seem to have their functions independent of the whole they may be a part of. For example, the function of a car to transport people seems to be independent of any whole of which a car can be considered to be a part.

Thirdly, MFM stresses the difference between goals and functions. In the example given in [Lind, 1994] the components of circulation system realize the following functions: transport of water from supply to expansion tank, circulation of water, transport of energy from boiler to radiator. Those functions contribute to the following goals respectively: maintain water level within safe limits, maintain condition for energy transport, and keep room temperature within limits. In this sense a goal answers the question *why* a function is performed. The goal, however, does not determine the function, i.e. the water level within safe limits can be maintained also by other functions than transport of water from supply to expansion tank, say by evaporation. Thus the function in MFM not only says *what* is done, but also *how* the goal is achieved. In this sense a function in MFM involves a particular way of realization and resembles the function described by CPD in FR and CFRL. Moreover, it seems that the notions of goal and function are used convertibly in some cases, e.g. although *maintain* is a function type, *maintain water within safe limits* is considered as a goal, not as a function.

2.1.4 Functions as Effects

Function is identified with the effect a device has in a given environment by a number of researchers e.g. by Chandrasekaran and Josephson, Brown and Blessing or in the Function-Behavior-Structure approach. This view is close to the view of the function as the role of an object. Here, an effect of a device is often understood as the role an object has in a given environment.

Ontological Framework of Chandrasekaran and Josephson

In [Chandrasekaran, Josephson, 1997; Chandrasekaran, Josephson, 2000] the authors sought the smallest ontological framework sufficient for developing a notion of function. In those works the intention of authors was not to legislate how the term ‘function’ should be used, but to provide a descriptive theory of how functions are represented by people. This framework, in contrast to previous works of Chandrasekaran discussed above avoids representing functions by references to the behavior.

In the framework the object O is defined to have a role F in the world W iff O causes F to be satisfied in W . If F is intended or desired by agent A , who is a designer or a user of O , then it is said that O has or performs the function F in W for A .

An object is always considered in W in some context (view) called the mode of deployment (MoD). Thus, a function is always relativized to some mode of deployment. For example the function of a battery is to provide voltage is relativized to the MoD₁: battery is electrically connected to electrical terminals of a given object. The same battery has a function to support paper under the MoD₂: bottom surface of battery is on top of the object paper.

In accordance to the above definition a function is contextual - depends on MoD, intentional - depends on the intentions of a designer or a user, and is identified with the causal effect of a device. Since a function is not ascribed to devices only, as it was the case in FR, but is a role of an arbitrary object the framework is intended to be applied not only in the context of artifacts, but also e.g. in functional biology. For that purpose however the framework requires to interpret the evolution as the agentive designer, who intends the biological functions. This, however, may be seen as a too strong simplification (for discussion see section 6.3).

According to authors the approach, handles multiple function realizations, since the function does not involve the specification of the mechanism of realization. Moreover, it permits to handle passive functions, which do not involve behavior.

Extension of Brown and Blessing

Brown and Blessing in [Brown, Blessing, 2005] extend the above model of function with the notion of a goal. They observe that the key component of the definition of function is that a function is a desired role. They argue in turn that for each function, there must be some reason *why* it is desired. In other words there must be some *goal* that is intended by an agent, who desires the function. A goal is defined as a desired state of the world, and the intention - as a description of how to reach that goal. For all or a part of the intention there may be constructed a plan, which is a sequence of operations corresponding to the intentions. The plan should either reduce the complexity of the intention or reach the goal. In this sense intention and plan resemble CPD's describing the causal mechanism that realizes a function. Both, a plan in Brown's terms and CPD in FR, provide a mechanism of how the function is realized. Function description in Brown's meaning provides the answer for both questions concerning *what* is intended (a goal) and *how* (what is the plan) it is intended to be achieved.

Additionally, Brown distinguishes the function from the affordance, which is a possible action of an object, as well as the function intended by the designer, the function desired by a user and the one actually provided by the device. He observes that those three kinds of functions do not necessarily come together. It may happen that the user desires and uses a device differently than it was intended by a designer, or the desired and intended functions may be the same but due to the inappropriate construction the device does not deliver them. Finally, the designed functions and the actual ones resulting from them may not be what a user needs. This analysis is especially interesting in the context of malfunction.

Function-Behavior-Structure

Gero and colleagues [Gero, 1990; Qian, Gero, 1996] developed a framework called Function-Behavior-Structure (FBS_{structure}), which was aimed to represent functional knowledge supporting the multidisciplinary design by analogy. FBS_{structure}, just as a number of approaches discussed above, is based on the distinction between the notions of an artifact's structure, behavior and function.

The structural description includes only the physical, topological and geometrical properties of an object. Behavior is the description of an object's actions or processes in given circumstances. It is represented by qualitatively distinct states of an object connected by causal dependencies. In this sense it resembles CPD describing the mechanism of function realization. Function, in turn, is considered as the result of behavior, its effect or product. Thus, a function is still closely related to a behavior, but is not identified with it. Moreover, FBS_{structure} does not suffer from problems with representing functions which do not change

anything, as was the case in FCO. $FBS_{\text{structure}}$ does not only consider behaviors changing something (B1 behaviors in terms of [Sasajima et al., 1995]) but also static behaviors, which do not involve a change over time.

In recent works the $FBS_{\text{structure}}$ framework is extended, e.g. in [Gero, Kannengiesser, 2004] by the notion of situatedness and in [Rosenman, Gero, 1998] by the notion of purpose, which, similarly to MFM, is located on top of the function layer. Gero argues that artifacts have ascribed purposes not only when considered in techno-physical but also in socio-cultural environment, involving human concepts and intentions. Purposes represent the intentions behind an artifact and are the reasons for which an artifact exists. Thus a purpose is considered to be an intended function of an artifact.

In $FBS_{\text{structure}}$ in contrast to FCO, FBS_{state} or Bonnet's approach, functions are treated as the objective effects of an object, and not as subjective and intentional ones. Moreover, since all effects of an object's behavior are considered to be functions, then e.g. the functions of cars include such effects as space occupation, noise and pollution production. Those, however, do not seem to be considered commonly as functions of a car. This distinguishes the FBS approach from Chandrasekaran's ontology, where not all effects of an object are considered to be functions but only those, which are intended by a designer (thus only purposes in Gero's terminology). That, on the other hand, seems to be too restrictive, since there is a big group of functions, which are not intended by a designer or user but follow from the actual capabilities of objects.

2.1.5 Conclusions

Above we have presented and classified the selected AI approaches to functional representation in area of technical device modeling. The list of the works presented cannot be considered complete and a number of approaches have been omitted due to the rigor of space. To our knowledge, however, they subsume the general views on functional representations distinguished above.

Many of the approaches in this field come not only within theoretical investigation but provide tools supporting device engineering, like e.g. FBS Modeler [Umeda et al., 1996] or Schemabuilder [Bracewell, Sharpe, 1996].

The overview presented demonstrates the difficulties and problem areas that arise when representing functional knowledge and defining the notion of function. Let us now summarize them briefly:

- The common problem is the distinction of function from behavior and structure on the one hand and from purpose and goal on the other hand. The discussion involves the functional explanation types related to the above notions:
 1. Why an object exists?
 2. What an object does?
 3. Why an object is constructed in a given manner?
 4. Why an object does what it does?
 5. How an object does what it does?

In the ontology of Chandrasekaran and Josephson function is identified with the answer to the first question. In MFM it is related to the second question, goal to the third and the fourth and behavior to the fifth. In contrast, in FR, CFRL, and FBS_{state} function combines the answers to both the second and the fifth. Finally, in Salustri's approach function is considered to answer the fifth and a behavior the second question.

- Function understood in terms of the answer to the what-question (the second question) may have several flavors: in Gero's approach a function is considered as everything an object does, in Chandrasekaran and Josephson's as only that which an object does and is intended by a designer or user to do, whereas Brown differentiates designer from user function and the actual capabilities of the device.
- The strong correlation of function and behavior raises problems with the representation of passive functions, not related to dynamic behavior (the exception here is Gero's approach).
- The representation of functions in the approaches discussed is heavily biased towards the purpose and domain of the application. For instance, in FR function involves a mechanism of realization since it is important for diagnosis, where a causal process is analyzed in order to see why the intended function is not being achieved.
- In most of the approaches (apart from the ontological framework of Chandrasekaran and Josephson) functions are considered only in the context of (technical/mechanical) devices. However, the direct application of the notion of function, acceptable in device representation, to other domains seems to result in oversimplification (e.g. the ontological framework of Chandrasekaran and Josephson).
- Functions are often contrasted to structure and behavior, as subjective and intentional.
- Several types of relations between functions are introduced; most commonly these are the relations of subsumption and part-of.
- Functions result not only in intended effects but also in those unwanted and harmful. Those effects should be included into functional model but they should not be mixed with intended effects, as it is done e.g. in FBS_{structure} where next to intended effects also unwanted effects are considered to be valid functions.

- In [Chandrasekaran, Josephson, 1997] authors have pointed out that not only objects but also processes may have functions. For example, the process of boiling water may have the function to produce steam. However, the approaches discussed consider only functions of objects.

2.2 Software Engineering and Business Modeling

The notion of function is crucial not only in the context of the design of technical/mechanical devices but also of other types of artifacts. In the current section we focus on two of them, namely on software and business systems. We analyze the selected approaches in the areas of software engineering and business modeling, starting from structured methods, through object-oriented ones and ending with the heterogeneous methods.

2.2.1 Structured Methods

Structured methods date from the mid 1970s when they were invented as the solution for the problems met in the development of large-scale business systems, such as inadequate requirement elicitation, limited user involvement, ad hoc and ambiguous modeling and design techniques. Many structured methodologies and techniques have been developed, including Structured Systems Analysis and Design Methodology (SSADM) [Ashworth, Goodland, 1990], the Yourdon Systems Method [Yourdon, 1993], Gane and Sarson's approach [Gane, Sarson, 1979], MERISE [Quang, Chartier-Kastler, 1991], or the CASE*Method [Barker, 1990a; [Barker, 1990b; Barker, Longman, 1992].

Some of the structured methods involve an aspect of functional modeling, in particular the technique of functional decomposition, which is introduced among others in Gane and Sarson approach, the Yourdon Systems Method and the CASE*Method. In functional decomposition the system's most general functions are decomposed to the hierarchy of more detailed functions.

We will investigate the functional modeling aspect of the structured methods on the example of the CASE*Method (CM) developed by Barker, which underlies the Oracle Designer CASE tool. The CM introduces an explicit distinction between behavior (business process) and business function. The modeling of functions is the early phrase of a system's design, in which the analysis are concentrated on what an enterprise does, rather than on what and how the output software should work.

A function in CM represents *what* an enterprise does, independently of the mechanism of its realization. It is depicted with a box containing a natural language expression of the form “do something” (figure 9). Functions may be invoked by the events, called triggers and represented as labeled arrows, and may generate the events. In this sense the notion of function in CM resembles the input-output view, discussed in the previous section.

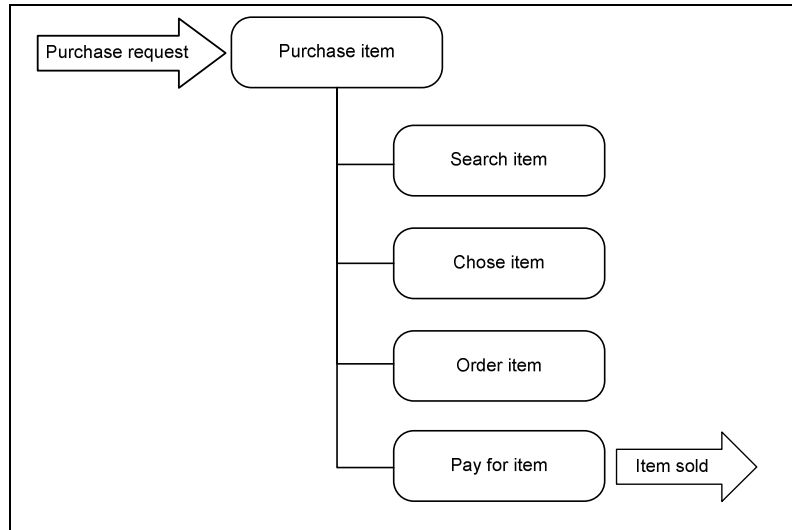


Figure 9. Hierarchy of functions in the CASE*Method.

In the function hierarchy diagrams functions are organized into hierarchy, in which all direct subfunctions of a decomposed function F are all the functions necessary for the realization of F (figure 9). In this sense the relation between the decomposed function and the set of its direct subfunctions is the relation of realization – a superfunction is realized by the set (sequence) of its direct subfunctions. Functions which are not decomposable are called *leaf* functions. In addition to the functional hierarchy, functions in CM are organized into Function Dependency diagrams, which represent causal dependencies between functions.

Processes in CM resemble functions, except that they do not provide the specification of what an enterprise does in order to achieve its goals, but instead they specify what a system should do. Moreover, processes include mechanisms of realizations. Thus, the distinction between function and process in CM is twofold:

1. Functions are mechanism-free, whereas processes involve mechanisms.
2. Functions depict an enterprise, whereas processes – a software.

The second issue we find irrelevant for our purposes, whereas the first reveals the dichotomy already discussed in the context of the AI approaches to device design: function answers the “*what* is done?” question while process, answers both the “*what* is done?” and “*how* it is done?” questions.

2.2.2 Object-Oriented Modeling and UML

Object-orientedness is nowadays a dominant paradigm in software engineering, although as some researches observe, there still is no definite proof that it is better than structured methods [Glass, 2002]. Object-orientedness started with Simula-67 developed in 1967 and got an impact in the 80's with the availability of Smalltalk and later by C, C++. The first object-oriented analysis methods were released in the late 1980s and early 1990s e.g. [Shlaer, Mellor, 1988; Coad, Yourdon, 1991].

The current de facto standard of object-oriented analysis is the Unified Modeling Language (UML) which initially was intended to unify the Booch [Booch, 1993] and the OMT [Rumbaugh et al., 1991] methods. At present UML is developed and maintained by the Object Management Group (OMG) [OMG, 2006]. Its current version is UML 2.0 [OMG, 2005].

UML 2.0 is founded on the explicit distinction of the static and the dynamic view of a system; it introduces thirteen diagrams grouped into two sets: structural modeling diagrams and behavioral modeling diagrams¹².

Structure diagrams define the static architecture of a model. They model the 'things' that make up a model and the dependencies and the relations between them. They handle both the physical and the abstract components (classes, objects, interfaces) of the system. The class diagram, which is the object-oriented successor of the entity relationships diagram introduced by Chen [Chen, 1976] is the core of the static view. Behavioral modeling diagrams represent the behavior of a system over time.

This architecture reminds of the distinction between structure and behavior adopted by several approaches to device design discussed in the previous section, like FBS_{structure} or MFM. But, in contrast to those, UML lacks the separate and independent representation layer for the functional modeling. However, some elements of the functional modeling are present in UML in the behavioral diagrams, in particular in use case diagrams.

The use case diagram is aimed to represent the overall functionality of a system, a subsystem or a class perceived (and available) by the outside users, called actors. Each use case represents a coherent unit of functionality. It represents what a system (or a subsystem, a class or an interface) does in interaction with an external actor. A use case is depicted by an oval labeled with a short active verb phrase (figure 10). A use case does not specify *how* the system realizes a function. This is specified by corresponding sequences of events. They depict

¹² Structural diagrams comprise package diagrams, class diagrams, object diagrams, composite structure diagrams, component diagrams and deployment diagrams. Behavioral diagrams comprise activity diagrams, state machine diagrams, communication diagrams, sequence diagrams, timing diagrams, interaction overview diagrams and use case diagrams.

“all the behavior use case entails – the mainline sequence, different variations on normal behavior, together with desired response” ([Rumbaugh et al., 1999] p. 64). One use case may have more than one sequence realizing it, each called a scenario. Scenarios are called primary when defining a main sequence of use case realization or secondary when defining an alternative sequence. In the later phases of the design the informal textual scenarios are replaced with the behavioral diagrams e.g. activity diagrams.

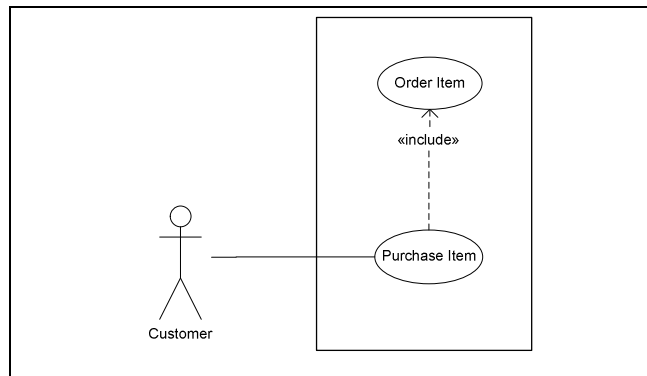


Figure 10. Use Case diagram.

Use case diagrams represent several relation types in which use cases are involved. These are: *association* of a use case to an actor, *generalization* of use cases, the *include* and the *extend* relations, indicating the insertion of an additional use case into a given use case. The types of relations introduced between use cases resemble the relations between functions introduced in device modeling paradigms:

- The extend and include relations could be understood as a part-of relation, where in the former case it is an optional part-of, meaning that a use case may optionally be a part of base use case, whereas in the latter a use case is a mandatory part of a base use case.
- Generalization resembles the is-a relation in FCO.

Considering the use case diagram as a formalism for functional modeling also raises several issues. Firstly, a use case comprises both the behavior and the function of a classifier. Although it is pointed out that the use case does specify *what* a system does and not *how* it is realized, the explicit difference between the notion of function and behavior is not provided. Those two notions are often used convertibly, i.e. the use case is understood on the one hand side as a coherent unit of functionality ([Rumbaugh et al., 1999], p. 488) and on the other hand as a descriptor of a potential behavior ([Rumbaugh et al., 1999], p. 489).

Secondly, use cases lack specification beyond natural language label. The only precise definition of the use case is given by its behavior in scenario, which however does not specify the use case but rather its realization. The use case resembles the function in FBS_{state} , where

functional symbol is a natural language expression, for which semantics is given in terms of realizing its behavior.

Thirdly, as Fernandes in [Fernandes, 2003] observes, use cases are defined in terms of interactions between one or more actors and the system. However, some systems may include a substantial percentage of their functionality that is not a reaction to an actor's input ([Fernandes, 2003], p. 20). Use cases do not permit to represent those functionalities¹³.

Fourthly, since use case diagrams are behavioral diagrams we can see that UML does not provide means to model functions beyond behavior. This results not only in the confusion of the terms of function and behavior but in fact reduces the former to the latter.

UML Business Patterns

The applications of UML exceed the area of software engineering. For example Eriksson and Penker demonstrates in [Eriksson, Penker, 2000] how to model by means of UML business systems. They have provided a set of UML patterns for the purpose of business modeling. Of our interest here are especially Goal Patterns, which enable goal and functional modeling. Goal Patterns support goal modeling, which is considered to be the critical issue in business modeling. There are three Goal Patterns developed:

1. Business Goal Allocation Pattern, which is used for assigning goals to business processes, resources and rules.
2. Business Goal Decomposition Pattern, which permits to break down goals into hierarchies of subgoals.
3. Business Goal Problem Pattern used for the identification of problems that can hinder the achievement of goals.

Of special interest in connection with functional modeling are the first two patterns. According to the Business Goal Allocation Pattern every business process should have a goal assigned (top part of figure 11). A goal is not considered as the output of a process, but it is introduced in terms of a desired state. For example, the process of `selling and delivery` has the output `a delivered product`, whereas the assigned goal is `a high rate of return` (bottom part of figure 11). A goal may be assigned not only to a process, but also to other elements of the model. For example, the output of the business process can itself have a

¹³ This argument is however debatable. In principle actors represent external users of a classifier, however "actors of lower-level subsystems may be other classes within the overall system" ([Rumbaugh, et al., 1999], p. 489).

goal assigned, e.g. a delivered product, which, as the output of the delivery process, could have the goal that a client is satisfied (bottom part of figure 11).

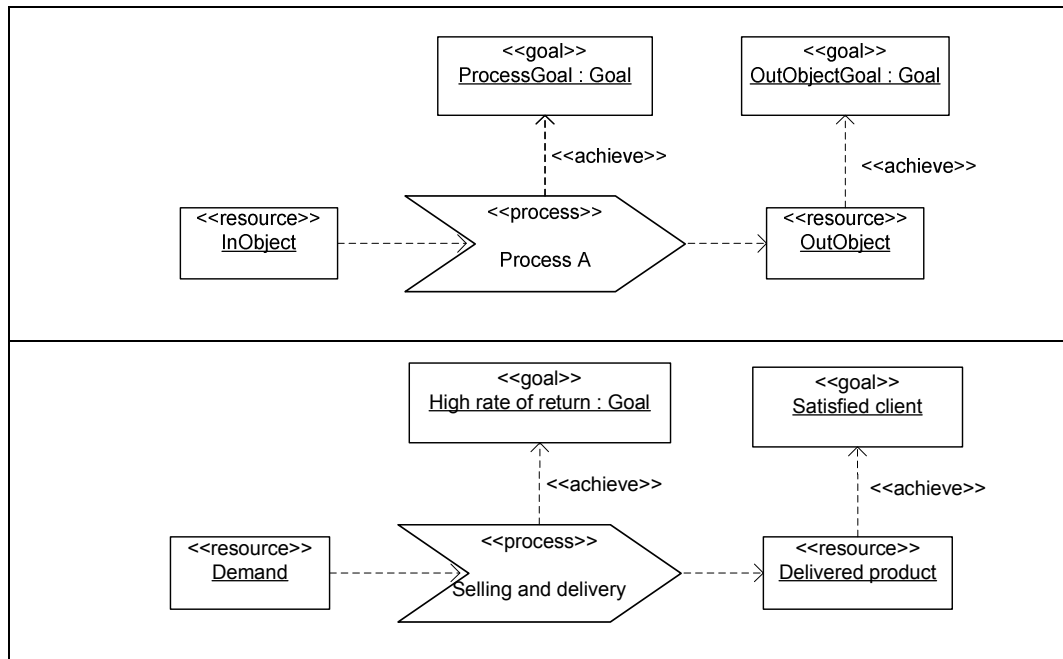


Figure 11. Business Goal Allocation Pattern (from [Eriksson, Penker, 2000], p. 278), together with an example (extended version of [Eriksson, Penker, 2000], p. 277).

The notion of a goal as defined in the pattern does not permit to distinguish between the states intended to be achieved by each individual instance of the processes/outputs and the general goals not intended to be achieved by individual processes/outputs. For instance, both types of goals are undifferentiated in the example of the sales process, provided by the authors ([Eriksson, Penker, 2000], p. 280). The goal of the sales process (to meet the annual sales budget) is not a goal of an individual process but of a whole set of processes, whereas in the same diagram the goal of the outcome of that process (satisfied customer) is a goal of *each* individual process.

Moreover, the difference between an output and a goal of the process is not clear. One could raise a question why a goal cannot be understood as the additional output of a process. By definition, output objects are the objects produced of the process (its results), this however holds also for goals. In the above example a satisfied customer, next to a delivered product, could be considered as an additional output of the process of selling and delivery. It seems of no help here to remark that the goal is an *intended* result, since outputs are intended results as well.

Overall business goals can be decomposed to subgoals by means of the Business Goal Decomposition Pattern, which is of particular help in the identification of business functions.

The decomposition of a goal is done by examining *how* a goal is achieved, whereas the super goal provides the reasons for its subgoals and answers the *why*-question.

In the example provided ([Eriksson, Penker, 2000], p. 286) the overall goal of Internet Business Inc. is to attract many customers. This goal is decomposed into three subgoals referring to three different customer categories:

- Many internet visitors
- Many registered customers
- Many subscribing customers

Those goals are further decomposed, e.g. the third is decomposed to the following sub-goals:

- Communicate bonus service for subscribers
- Active pricing
- Provide good bonus service

The second decomposition does not rely on the categorization of subscribing customers, just as it was done in the case of the decomposition of the most overall goal. Instead, it refers to the different ways of achieving the goal. Therefore it seems that in the Business Goal Decomposition Pattern two hierarchies are confused: subsumption of goals and the way-of-realization (see FCO, section 2.1.2).

Some of the goals are represented as states in which processes result, e.g. many internet visitors, whereas others are rather the processes that lead to achieving them, e.g. provide good bonus service. Therefore, it seems that the notion of a goal covers in Goals Pattern both the result and the process leading to it.

Concluding, we can say that the business pattern shows an important close relation between the notions of a function and the notion of a goal, although the overall pattern is not free of problems as discussed above. In this sense it reminds the AI approaches reported in the previous section, which interpret function in terms of an effect.

2.2.3 Object-Process Methodology

Object-Process Methodology (OPM), developed by Dori [Dori, 2002], is a meta-model for conceptual modeling and system engineering, which integrates function, structure and behavior of a system. Intended application area of OPM exceeds software engineering and covers both technical and natural systems modeling. OPM consists of the Object Process Diagrams (OPD), which are the visual formalism that describes the structure and/or behavior

of the system or its part ([Dori, 2002], p. 106) and the Object Process Language (OPL), which is a semi-natural language for specifying OPD.

OPM, in contrast to UML or the CASE*Method, provides rich and precise formalism for representing functional knowledge. The main components of the system in OPM are processes and objects. Dori writes that whereas objects are what a system or product is, “processes are what a system does” ([Dori, 2002], foreword). Functions in OPM are distinguished from processes and are defined in the following way:

“Function is an attribute of object that describes the rationale behind its existence, the intent for which it was built, the purpose for which it exists, the goal it serves, or the set of phenomena or behaviors it exhibits. “([Dori, 2002], p. 251)

This definition, according to Dori, emphasizes the *what* and the *why* aspects of function. In contrast the dynamics (behavior) is concerned with the *how* question. Dori argues that function and dynamics are often-confused synonyms, while in fact they are distinct concepts. A function is about what a system does and why it does it, while dynamics (or a system’s behavior) is about how the system acts or operates to attain its function.

Functions are ascribed to objects; each object having (caring) a significant function is called in OPM a system. Not all functions are equally significant for a system. The significance of a function depends on its contribution to the desired purpose.

Functions in OPM are represented by function names phrased in OPL as command sentences, which are imperative verbal phrases of the form “do something”. Function names are depicted inside dashed boxes called function boxes. A function box encloses at least one object and one process. (figure 12). Functions in OPM are context-dependent and subjective; they depend on the viewpoint of the beneficiary. Different structure-behavior combinations can achieve the same function. For example the function of `telling the time of day`, may be achieved by the sundial, or the mechanical clock architecture.

The only kind of relation between functions introduced in OPM is the functional decomposition. Functional decomposition is the principle of building function hierarchies. Intuitively it is the relation of enablement, where each lower function enables the upper one and provides the answer how the upper function is achieved. It is represented by means of the part-of relation between the objects in the functional boxes (figure 12). For example the function `generate circular motion` is considered to be a subfunction of the function `enable transition`, since the former is the function carried by the internal combustion engine, which is a part of the a car whose function is `enable translation` (figure 12).

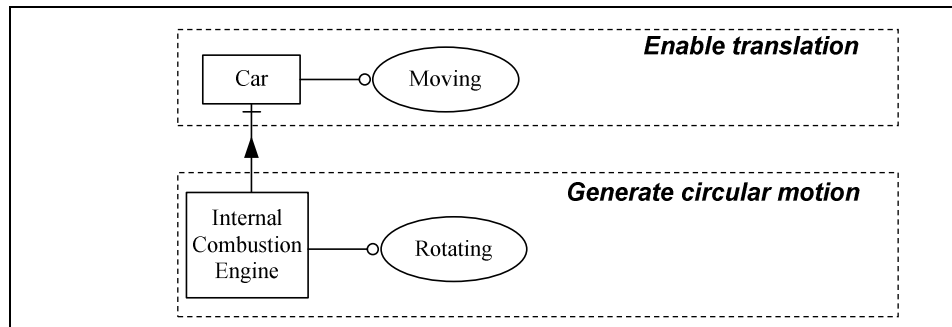


Figure 12. OPM function representation and function decomposition (from [Dori, 2002]).

The function hierarchy based on functional decomposition is in fact the hierarchy of function realizations not of functions as such, where a function of object *O* is decomposed to the functions of objects being parts of *O*. As long as functions are not assigned to objects, say in the early stages of design, OPM does not permit to decompose them, in contrast to, for example, the CASE*Method.

Function hierarchy in OPM thus resembles rather functional dependence between a part and a whole as in [Vieu, Aurnague, 2005] than the part-of relation between functions. It should be mentioned, however, that the function hierarchy can not be identified with every part-of relation. For instance, although a car can have as its part a cd player, it makes no sense to decompose enabling translation into playing cds. Moreover, relying on part-of relation in function hierarchy does not permit to decompose functions achieved by the external objects which are not parts of the system realizing the superordinate function.

The problem with OPM arises when it comes to functions of processes. In [Chandrasekaran, Josephson, 1997] the authors have pointed out that not only objects but also processes may have functions. This is especially important in the context of services, which are often processes and clearly have functions. This, however, cannot be handled in OPM since function is defined there as an attribute of an object, and objects are disjoint with processes. Moreover, OPM suffers from problems with the notion of service itself. In OPM services are considered as products (which are artifact systems) being processes ([Dori, 2002], p. 266). However, it remains contradictory with the definition of a system, saying that a system is an *object* caring a significant function ([Dori, 2002], p. 253). Since objects and processes in OPM are disjoint, and since products are defined as artificial systems (and thus as peculiar objects), they can not be processes.

2.2.4 Conclusions

In the above section we have pursued the notion of function in the areas of software engineering and business modeling, starting from structured methods, through object-oriented methods and ending with heterogeneous methods.

The results of this pursuit have shown that, just as in AI approaches to functional modeling, on the one hand there the distinction between function and behavior is stressed (e.g. OPM) and on the other those notions are mixed or are left inseparable (e.g. UML). All the approaches discussed come with means for representing interdependences between functions, in particular the is-a and part-of relations. However, the understanding of those significantly varies from formalism to formalism. Moreover, those relations are often realization-biased, e.g. functional decomposing in OPM, or the underlying principles are not explicit, e.g. the decomposition of goals in Eriksson and Penker's UML profile.

2.3 Functions in Philosophy

The notion of function has recently been broadly discussed in philosophy, although the first remarks come already from Aristotle, who among four causes distinguished the final cause, and discussed it in the context of functional explanation as “the end (telos), that for the sake of which a thing is done” [Cohen, 2002]. Roughly speaking, nowadays in philosophy two main ways of understanding functions can be distinguished: one formulated by Cummins in the 1970s [Cummins, 1975], who discussed the notion of the function in the context of the notion of disposition, and the second, called an etiological approach, founded in late 1980s and early 1990s by, among others, Millikan [Millikan, 1989a; Millikan, 1989b], Neander [Neander, 1991a; Neander, 1991b], Griffiths [Griffiths, 1993] and Godfrey-Smith [Godfrey-Smith, 1993].

In contrast to most of the approaches discussed so far (apart from OPM and the ontology of Chandrasekaran and Josephson), philosophical approaches not only account for the functions of artifacts but also deal with the functions of biological and social systems. Below we present a short overview of two most dominant philosophical approaches. Moreover, we will introduce the problem of the ontological status of function, discussed in the context of social reality by Searle in [Searle, 1995].

2.3.1 Functions as Dispositions

The idea of tying the notion of function to the notion of disposition is already present in Wright's article [Wright, 1973]. He defines the expression "the function of x is z " as follows: (i) x is there because it does z and (ii) z is a consequence (or result) of x 's being there. To cover the cases of malfunction Wright relaxes the condition (i) and interprets it as follows: "all that seems to be required is that x be able to do z under appropriate conditions(..)"([Wright, 1973], p. 158)¹⁴. In this sense x has a function z if, among others, x is able (has a disposition) to do z .

Function is explicitly related to disposition (also called capability) by [Cummins, 1975], where he defines the *has-function* in the following way:

" x functions as a ϕ in s (or: the function of x in s is to ϕ) relative to an analytical account A of s 's capacity to ψ just in case x is capable of ϕ -ing in s and A appropriately and adequately accounts for s 's capacity to ψ by, in part, appealing to the capacity of x to ϕ in s ."([Cummins, 1975], p. 768)

Interestingly, according to this definition the function of x is relativized not only to the capacity of some system s to which x is related (just as in Chandrasekaran and Josephson's ontological framework function is relativized to MoD), but it is relativized also to the epistemological aspect, namely to the analytical account of this capacity. Thus, it is appropriate to say that the function of a heart is to pump blood only when a heart is considered in the context of a particular explanation of the circulatory system's capacity to transport food, oxygen and other substances.

Buller in [Buller, 1998] surveys several objections against Cummins' theory that Millikan raised in a number of papers. First in [Millikan 1989a] she argued that Cummins' theory is too liberal in assigning functions. According to her, for a given entity one can find an unlimited number of systems or subsystems in which the entity is involved. This, in turn, makes the number of Cummins' functions unlimited as well, e.g. if the Earth's water-cycle is seen as a system, then clouds by producing rain make vegetation grow, and in this sense should have ascribed (in Cummins' sense) the function of making the vegetation grow in this system. However, as Millikan observes, to make vegetation grow is clearly not a purpose of clouds. Thus, she observes that Cummins' function has nothing to do with the purpose of the thing in question, whereas in particular in biology having a function is a matter of having a purpose. This objection seems to touch more the problem of finding the rules of delimiting the

¹⁴ Cited after [Kreos, 2001].

containing system than the problem of function ascription itself. It seems perfectly appropriate to say that clouds in the Earth's water cycle have the function of making the vegetation grow, although it sounds odd to say that they have a purpose in doing so.

The lack of references to purpose also disables Cummins from handling *accidental effects* [Millikan, 2002]. In Cummins' understanding of a function all effects that contribute to the capacity of a containing system are functions, also those that are contributing to it by accident. In this sense Cummins' theory seems to confuse the function *of* with the function *as*. Wright, who also relates to capacity in defining functions, distinguishes accidental benefits from functions by (i) condition, which demands that a function is a reason of the object's existence.

Thirdly, Cummins' theory has problems with ascribing functions to malfunctioning objects. Cummins' claims that "if the function of x in s is to ϕ , then x has a disposition to ϕ in s " ([Cummins, 1975], p. 758). Thus, an object that lacks a disposition to ϕ in s does not have a function to ϕ in s either. For instance, the Cummins' definition implies that a heart that lacks a disposition to pump blood lacks that function as well. Millikan, however, argues that a malformed heart is still considered to have the function of pumping blood. Thus she says that function should not be defined by reference to what an object is *capable* of doing, but what it is *supposed* to do.

Finally, it is argued that Cummins' theory of function ascription does not explain the presence (or existence) of the object having the ascribed function. For example, making the vegetation grow does not explain the presence of clouds.

2.3.2 Etiological Theories

As an alternative to Cummins' function the ethological theory of functions was introduced. The primary area of the theory's application was biology, however, there are also attempts to interpret the artifact's functions in terms of etiological theories (e.g. [Vermaas, Houkes, 2003]). In general, the etiological theory originates from Wright's (i) condition cited above and reveals the function of an item by reference to its kind's (evolutionary) history, not to its current capacities. The explanatory motivations behind the etiological theory are also different than in Cummins' approach. Cummins answers the question about how the capacity of a given object contributes to the capacity of a system into which the object is involved, whereas the explanatory theories explain why a given object (or a kind of objects) exists. In this sense the question about the function in the context of the etiological theory is the teleological question about the purpose (or reason) of existence.

Etiological theories have been quite popular and come in several flavors, here we refer to two of the first developed, namely that of Neander and of Millikan. Neander defines the proper function in the following way:

“It is the/a proper function of an item (*X*) of an organism (*O*) to do that which items of *X*’s type did to contribute to the inclusive fitness of *O*’s ancestors, and which caused the genotype, of which *X* is the phenotypic expression, to be selected by natural selection.” ([Neander 1991b], p. 174)

The proper function of a trait of an organism *O* is such an activity which contributed to the fitness of *O*’s ancestors and for which the trait was selected by natural selection. For instance, the function of a bird’s wings is to enable to fly since flying contributed to the fitness of birds and it is the reason for which wings were selected by natural selection. Neander considers a proper function of a trait as “whatever it was selected for”.

In contrast to Cummins’ function, the proper function is not relative to the context (neither ontological nor epistemological) in which a given object is considered, but it refers only to its history, and hence it may be considered to be objective.

Proper function does not suffer from being too liberal as Cummins’ function did, because it does not treat every disposition contributing to some system as a function but only one which has been selected for that purpose. Neither has it any difficulties in dealing with accidental effects. Only those effects of an item which were selected are considered to be its functions. Finally, it handles properly with the normative character of functions and malfunctions. An item that lacks a disposition, and therefore lacks Cummins function, may still have a proper function since proper function does not refer to an item’s current dispositions but to its history. A malfunction with respect to function *F* may be predicated when an item has a proper function to *F* but lacks a disposition to *F*.

Etiological Theories of Artifacts

The primary domain of the application of etiological theories is biology, whereas Cummins functions seem to be better suited for explaining how artifacts function. However, Millikan also tried to adopt her theory to artifacts. Although, the etiological theories presented above are non-intentionalist, Millikan also introduces the intentionalist variant of her etiological theory¹⁵.

¹⁵ We use the notion of intentionalist etiological theory as it is introduced by Vermaas and Houkes in [Vermaas, Houkes, 2003]. They call an etiological theory of functions intentionalist iff it ascribes

In [Millikan, 1984] it is proposed to treat as (derived) proper functions of artifacts the functions intended for them by their makers. In this sense the history involving the intentions of artifact's designers delimits the artifact's proper function. This corresponds with the Kitcher's approach, according to which "there is a direct link between function and intention: the function of X is what X is designed to do, and the design stems from an explicit intention that X do just that" ([Kitcher, 1993], p. 260).

The etiological intentionalist theory resembles the ontological framework of Chandrasekaran and Josephson, in which the function of an object is also revealed by the intentions of a designer. In contrast to Chandrasekaran and Josephson, the intentionalist etiological theory is applied only to artifacts, whereas natural objects are explained by non-intentionalist etiological theories.

Discussion

For an argument against interpreting functions in terms of the designer's intentions one may refer to the example given by [Keil, 2003]. As a psychologist, Keil is interested in the problem of the so-called essence of human concepts. It is sometimes postulated (e.g. in [Bloom, 1996]) that people consider the intentions of a designer as essential for the categorization of artifacts. Thus, some writers of psychological essentialism seem to agree with the intentionalist theory of functions.

Keil discusses the following counter-example: he considers a hypothetical Adam, who wanders into a surgical suite and spots an array of surgical instruments lying on the table. He picks up one labeled "re-seater", takes it home and, being a skilled mechanist, duplicates it for the purpose of selling it on the black market. Although his intention is to copy a surgical tool, it turns out that the object he copied was a plumber's tool left there accidentally by the plumber, who removed some defect in the surgical suite. Keil argues that we do not think that Adam's tool is the surgical tool, although Adam's intention was to manufacture one. It seems that similar conclusions can be drawn for functions. Only on behalf of Adam's intention would we not ascribe any surgical function to the tool he manufactured. This particular problem can be solved by the condition saying that a function not only must be intended by a designer but moreover an item must be capable of realizing it. This, however, analogously as the Cummins' approach would have problems of handling malfunctions.

It is not only the intentionalist version of the etiological theory that raises problems, but the idea of relating the function of an object to its history, or the history of object's species,

functions to items *I* on the basis of causal histories *ch(I)* that necessarily involve intentional behavior of agents ([Vermaas, Houkes, 2003], p. 270).

itself is problematic. An object that has identical disposition as a given object *o* but has a different history cannot have, according to etiological theory, the same function as *o*, which however seems counterintuitive. Since having a function is reduced to having appropriate history, the etiological theory implies that an *accidental double* that is molecularly identical with an object having a proper function does not have this proper function since an accidental double does not have the right history [Millikan 1989b].

Moreover, whilst the Cummins' theory is argued to be too liberal in assigning functions the etiological theory is found to be too strict. For example, Davies [Davies, 2000a] finds that functions of organisms arise also as a consequence of non-selective forces such as e.g. drift.

Assigning functions in accordance with the etiological approach is very demanding in its requirement of knowing the purposes behind natural selection. [Vieu, Aurnague, 2005] report that although the etiological approach was originally intended to explain the phenomena of biological functions, it has been criticized in biology "for impracticability to refer to the evolutionary history of some organisms, especially fossils, while biologists still use a functional talk in these cases" (p. 491).

In addition, Preston in [Preston, 1998] criticized the attempt of reducing all function ascriptions to proper functions. She argued that apart from the proper functions, which are normative and permanent, there is a vast group of temporary and non-normative functions, called system functions, especially in the area of artifacts. System functions are either unintended functions or the functions ascribed to the items used in a novel way, e.g. soft drink bottles used as bird feeders. Those functions are not proper since they are ascribed independently of the causal history of the items, but instead as results of an object having some dispositions. System functions are not normative and are understood in terms of Cummins' functions.

2.3.3 Ontological Status of Functions

Proper function is considered to be an objects' objective feature, which is the result of a particular (objective) causal history. In contrast, the Cummins' function is highly relative; it depends not only on the system in which a given item is contained but also on the way in which the system is explained. This radical difference raises the problem of the *ontological status* of functions. The question is then the following: of what ontological kind is function?

The ontological status of functions was discussed by Searle in [Searle, 1995]. He argues, just as Cummins, that functions are not the objective features of reality - they "are never intrinsic to physics of any phenomena but are assigned from outside by conscious observers and users. *Functions, in short, are never intrinsic but are always observer relative*"(original

emphasis, [Searle, 1995], p. 14). In this sense Searle's approach is analogical to the approach of Bonnet reported in the previous section, where the function was relativized to some person's belief. Bonnet is concerned with the beliefs of a designer or user, Searle generalizes it to an observer. Thus, also a neutral observer, which is neither using nor designing an object may assign a function to an object. Functions then, according to Searle, are ontologically subjective - the functional features exist then relatively to some observer or user ([Searle, 1995], p. 10). In contrast to the above approach functions are sometimes considered to be non-intentional entities, especially when considered in terms of (an objective) behavior or capacity.

2.3.4 Conclusions

The discussion about functions in philosophy is mainly concerned with the problem of function ascription. Philosophers do not seem interested in the structure of functions, which is so broadly discussed in AI and conceptual modeling. This is of no surprise, since philosophy is not concerned with the practical problems of building functional models, which in turn is the issue in computer science.

Finally, function ascription in philosophy is analyzed in a wider context, covering not only functions of artifacts but also functions of natural objects. In this sense the approaches developed there seem to be richer and more general than those discussed in the previous sections.

2.4 Requirements for an Ontology of Functions

In the current section on the basis of the works discussed we will summarize the issues which we find to be of importance for a top-level ontology of functions.

The list below will be used as a guide in the coming chapters for the construction of the top-level ontology of functions. Many of the issues listed below are handled by at least one of the approaches mentioned. However, to our knowledge none of the approaches is aimed to handle all of them, which, in our opinion, is the task for top-level ontology.

In order to provide a domain independent top-level ontology of functions (OF), incorporated into a wider ontological framework, in our opinion four questions require an answer:

1. How to represent and determine functions independently of their realizations?
2. Under what conditions an entity is a realization of a function?
3. What does it mean that an entity has a function?

4. Of what ontological kind is function?

Those questions indicate the main areas, which OF should cover, namely the structure of function, the realization of function, function ascription, and the incorporation of the ontology of functions into a wider top-level framework. The first question concerns the structure of function and is of particular relevance in functional modeling, where it is required to represent functions independently of particular realizations. Secondly, we find it important to provide the ontological foundations for the evaluation of function realization. Thirdly, the functional description is often a part of the knowledge about entities, thus it is important to provide conditions for an item to assign a function to it. Finally, the ontology of functions is intended to be incorporated into a wider framework of the top-level ontology which will provide strong ontological foundations and enable cohesive representation of both functional and non-functional knowledge. In order to permit it, an ontological status of function should be investigated. Those four problem areas are broken down into the detailed requirements the ontology of functions should meet, presented in table 2.

Reference number	Requirement Description
R.1.	The representation of a function should be independent from the function's realization. In particular, the following should be provided:
R.1.1.	The function description, which is both precise and easily comprehensible for human users. Especially in complex models, comprising a high number of functions, it is important to represent functions in a form easily comprehensible for human users. On the other hand, the representation must be precise enough to enable identification of functions.
R.1.2.	The function representation should be compatible with most common understanding of functions, in particular with the input-output approach.
R.1.3.	Function and realization should be handled separately. In particular:
R.1.3.1.	The function representation should be realization-free, since functions are often modeled independently of non-functional aspects, especially in the first phases of the design.
R.1.3.2.	The description of function realization should be function-free (a non-functional description). Entities realizing functions may be described in purely non-functional way, i.e. as processes.
R.1.4.	Function should be differentiated from behavior and processes in general. The definition and representation of functions should not be given in behavioral terms. This enables one to deal with non-behavioral, passive functions.

R.1.5.	Functions should not be defined in the context of devices only, but the functions of non-devices (non-artifacts) should be handled as well.
R.1.6.	Relations between functions should be defined independently of a particular function realization. For instance, functional decomposition should be independent of the partonomy of structures realizing the function.
R.1.7.	Side effects and accidental benefits of functions should not be identified with function goals but should be included in the framework.
R.2.	Conditions for the realization of function which permit to evaluate entities against their functions should be provided.
R.2.1.	Apart from processes, other (static) entities should also be considered as candidates for realizations of functions.
R.3.	Conditions for function ascription, stating the circumstances under which an entity has a function, should be provided. In particular the following should be investigated:
R.3.1.	Possible modes of function ascription.
R.3.2.	Types of entities that may have functions ascribed. For example, not only objects (persistants, presentials) carry functions but also processes, e.g. services.
R.3.3.	The role of agents' beliefs and intentions in function ascription should be examined.
R.3.4.	The normativity of functions and malfunctions should be handled. Due to the normative character of functions also malfunctions are ascribed to entities.
R.4.	An ontology of functions should be incorporated into a full-fledged top-level ontology.
R.4.1.	The features of functions should be recognized. In order to find out of what ontological kind function is, first its properties should be investigated.
R.4.2.	The ontological status of function should be determined and the definition of function should be provided. The determination of the ontological status of function permits to incorporate it into the broader taxonomy of a top-level ontology.
R.4.3.	Taxonomy of functions should be developed. Classifications of functions enable one to specialize the concept of function and provide a backbone of the ontology of functions.

Table 2. List of the requirements for a top-level ontology of functions.

3 Structure of Functions

3.1 Introduction

After presenting in the previous chapter the requirements which we believe the ontology of functions should fulfill, in the present chapter we introduce the general structure of functions. The structure of functions is doubly relevant: first it provides the means necessary for representing functions independently of their realizations (ref. R.1), and secondly it sets frames for a definition of function (ref. R.4.2).

In OF we propose to represent the structure of function $Fu(F)$ as a quadruple $STR(F)=(LABEL(F), REQ(F), GOAL(F), FITEM(F))$, where:

- $LABEL(F)$ denotes a set of *labels* of function F .
- $REQ(F)$ denotes *requirements* of function F .
- $GOAL(F)$ denotes a *goal* of F .
- $FITEM(F)$ denotes a *functional item* of F .

In the present chapter we are going to discuss in detail each of the above function components. Moreover, an additional notion of *final state* of F , $FSTATE(F)$ will be introduced, which provides a means for the introduction of *multiple-goal* functions.

3.2 Label

Functions are represented in natural language form in a number of approaches to functional device modeling, e.g. FCO, FBS_{state} , FBS_{state}/m , but also in the fields of business modeling and system modeling, e.g. the CASE*Method or FBS_{state} . In the latter approach function is expressed as a tuple (f_{symbol}, b) , where the functional symbol f_{symbol} is a natural language expression of the form “to do something”, whereas in OPM functions are described by *function statements*, which are imperative sentences of the form “do something”.

The representation of functions in natural language is useful especially for the purpose of interaction with human users. However, we find it insufficient to reduce the representation of a function only to a natural language form as it is done in the CASE*Method or OPM, since this results in an imprecise function representation which does not permit further analyses of

functions. Therefore, we adopt the natural language form as an intuitive and easily comprehensible *label* of a function, but only for the purpose of supporting a human user. The remaining components of a function are introduced in order to permit a precise function representation.

Definition 1 (Labeling). The *labeling* of a function f is a set of natural language expressions which describe the function: $LABEL(f) = \{l_1, \dots, l_n\}$. Every member l_i ($1 \leq i \leq n$) of the labeling is called a *label* of the function.

Beside the verb phrase “to do something” adopted in FBS_{state} , OF further admits forms like “doing”, a substantive adverbial form, e.g. `research`, `login`, and a conditional form “if...then”. The latter has the advantage of an explicit reference not only to the goal state but to the function requirements as well. Finally, we do not restrict the number of function labels. A function may have several natural language descriptions assigned. For example, the labels `goods transport` or `to transport goods` may form a labeling of one and the same function.

As reported in section 2.1.2, FBS_{state}/m not only uses the phrases of natural language for function representation, but also it provides the decomposition of the phrases into three elements: a verb called *function body*, a noun called *objective entity* and an adverbial phrase called *function modifier*. We find it problematic to treat adverbial phrases in general as function modifiers, having some degree of satisfaction. For example, in the function `to deliver mail undamaged` the adverbial phrase `undamaged` should not be considered as a modifier but rather as a part of the function body. We believe that it does not say *how* a function should be achieved but *what* a function should achieve.

Secondly, modifiers in examples given by the authors in [Takeda et al., 1996] do not seem to refer to functions but rather to the objects on which functions operate, or even to external entities. For example, `bicycle rear rack` is considered to have a function `to carry/fasten backpack to a bike`. To that function the authors assign four modifiers: `easy of use`, `a sporty-appealing form`, `for most bikes`, and `reasonable price range`. However, it seems that only the first of them refers to the function. Neither `a sporty appealing form`, `for most bikes` nor `a reasonable price` refers to the function, but rather all of them are requirements for the device itself. Thus, they express non-functional requirements and have nothing to do with the function of a rack.

It seems intuitive that for each function there is an entity on which the function operates. In the function `to carry/fasten a backpack to a bicycle`, these entities are

bicycle and backpack. However, the notion of objective entity does not permit to distinguish entities which perform the function, here a *rack*, from those entities on which the function operates - a *bicycle* and a *backpack*.

In FCO a more detailed picture of the entities involved in a function is given. There are distinguished *agent*, *operand* and *conduit* of a function [Kitamura et al., 2004]. The division between an agent and an operand permits to distinguish an entity which performs the function from that on which the function operates – an operand.

Unfortunately, FCO has also some limitations in revealing the structure of the label of a function. An operand is defined as a physical object which is changed by the function. However, this does not hold for all types of functions, as Kitamura and colleagues [Kitamura et al., 2002] recognize themselves. There is a big group of functions not changing anything, and for those functions the notion of an operand is not properly defined. Moreover, not all functions operate on physical objects.

Although the grammatical analysis of function labels yields insight into a function, it is however difficult to build a formal representation of functions on that basis alone. We argue that the natural language form should be included in the functional structure, but it is insufficient to represent functions only by their labels, as these are too ambiguous and not precise enough in determining functions.

Therefore, in the current study we make use of some ideas presented in $FBS_{state/m}$, namely we adopt the distinction between the verb - the *function body* and the noun - the *operand* of the function. We make some improvements, trying to avoid the pitfalls reported above. An operand is considered herein neither in the FCO sense, i.e. being limited to physical entities changed by the functions, nor in the general sense as the objective entity in $FBS_{state/m}$. In our understanding, an operand is every entity involved in the function realization which is not a realizer¹⁶ of that function. In addition, we put no ontological constraints on the nature of an operand; thus it is not limited to physical objects.

3.3 Goal

Functions are commonly considered as teleological entities which is reflected in most of the approaches discussed in section 2.1 by the inclusion of a goal to the representation of functions. For example, [Sasajima et al., 1995] define function as a teleological interpretation

¹⁶ The notion of realizer is introduced in section 5.5, here suffice it to mention that intuitively a realizer is identified with an entity realizing a function. For example, the realizer of the function *to drive a car* is a person driving a car, namely a *driver*.

of behavior, and Chandrasekaran and Josephson identify the function of a device with the effect “that the object under discussion has on its environment”[Chandrasekaran, Josephson, 1997].

In OF we also consider functions to have a teleological character and represent this by including a goal into the structure of a function. This, however, yields the following questions: (1) Of what ontological kind are goals? (2) Does the goal completely determine the function or are additional determinants needed?

The second question is to be investigated in the next section. Regarding the first, it should be mentioned that there are differences in the understanding of the notion of a goal across the literature. In approaches like FR, the goal of a function is considered as a causal process of a function’s realization, whereas in McDowell’s approach a goal is seen as a goal state that must be reached and maintained, or the control relation that must be maintained [McDowell et al., 1996]. Hence, we see that FR identifies the goal of a function with the process of function realization, which, as reported in above, causes a functional representation to be realization dependent.

We interpret a goal, following [McDowell et al., 1996] for instance, as the result state of a function. For example, `pumping blood` is the function resulting in the state of `blood` being pumped, and the function of some logistics company of `transporting goods` to the specified destination results in the state in which goods are located in the specified destination.

The first observation concerning the nature of a goal is that it is a relational entity. For example, in Sowa’s ontology the relation *has purpose* is introduced as a triadic relation between an *agent* that has an intention, an *act* performed by that agent and an *intended situation*, which is the reason for which an agent performs an act ([Sowa, 2000], p. 272). Since we are not interested in the goals of the actions of agents but in goals of functions we paraphrase the above and define a goal as a relational entity mediating between an *agent*, a *function*, and some *chunk of reality*¹⁷:

¹⁷ The notion of *chunk of reality* by analogy to the notion of *state of reality* does not impose that a given ontological entity is indeed a part of *reality*. According to our pragmatic realistic approach, the entities of the ontology are not considered as parts of the reality but as elements of the model, i.e. ontology, which is used for the description of reality. Our approach is both realistic and pragmatic, since the only argument for our claim that the constructs used in the ontology do have their counterparts on the side of reality is pragmatic in nature – we assume that those counterparts exists since we find it useful to model reality by means of the model elements corresponding to them.

Definition 2 (Goal). A goal of a function f is an intentional entity established for some reason by an agent referring to a chunk of reality, which is (to be) affected by the function f .

A goal is a *relational* entity, since it is *of* something (of a function in our case) and *for* somebody (an agent). The $GoalOf(x,y)$ relation means that a goal x is assigned to some function y , and $GoalFor(x,y)$ means that x is a goal for an agent y ; in other words, agent y establishes the goal x . The entity x can be assigned to the function as its goal only if x is affected by function y , which is denoted by $Affect(y,x)$. Thus, definition 2 can be represented formally,

$$Goal(x,y,z) \rightarrow GoalOf(x,y) \wedge GoalFor(x,z). \quad (1)$$

In the next sections we will have a detailed look at the above definition.

3.3.1 Affected by the Function

Each function is associated with some state of the world, in which it is expected to result. In terms of GFO that state can be considered as a complex whole. For example, the function of painting a wall results in a configuration consisting of a wall, paint and the relation of being covered that holds between them. In accordance with the axiomatization of GFO in FOL provided in [Heller et al., 2005] this could be represented as follows:

$$\exists xy(Ph(x) \wedge x :: wall \wedge Ph(y) \wedge y :: paint \wedge covered-by(x, y)).^{18}$$

The state of the world, which is a result of the function f is called a *final state* of function f and is discussed in section 3.3.7. Nevertheless, we do not consider the whole final state to be the goal of the function. The goal of painting a wall is to establish a particular relation between the wall and the paint. That relation is the goal of painting the wall, in contrast to the overall situation consisting of a wall, some paint, and the relation between them, which may be the

¹⁸ There exists an individual physical object which is an instance of a wall and there exists an individual which is an instance of paint and the former is covered with the latter. The GFO predicate $Ph(x)$ denotes a physical object x and $x::y$ – x being an instance of y . All notions and symbols of GFO used in the current text are defined in appendix A.

goal of the function to preserve the existence of the wall and the paint AND to cover the wall with the paint.

Therefore we restrict the goal of the function only to those entities which are *affected by* the function. Intuitively, we call affected all those elements of the state of the world which are influenced by every realization of the function¹⁹. To grasp the intuitions behind it, consider our example of the function to paint a wall. Before the realization of the function happens, the wall is not covered with paint and after the realization it is covered, thus what the function affects is the relation *covered-by*(x,y) between a wall x and a paint y . Restricting the goal only to affected entities seems to meet the intuitions that the goal of the function to paint a wall is not to maintain the existence of the wall or the paint, but it concerns only the individual relation between a paint and a wall.

3.3.2 Agent

Although the notion of an agent is not central for the theory of functions, functions and agents are correlated, since functions are *agent-dependent* entities. Therefore, let us briefly clarify our understanding of the notion of agent.

In the agent based systems community a number of theories of agents have been developed (for an overview see [Wooldridge, Jennings, 1995]). Among the commonly accepted features of agents is proactivity. Agents are proactive in the sense that they are able to take the initiative in performing goal-directed actions. However, performing a goal-directed action requires that a goal *is established and recognized*. In this sense agents are understood herein as those entities which are able to establish goals. This narrow interpretation of an agent is adopted for our framework.

In addition in OF three particular relations are assigned to agents, namely the relation of having belief, denoted by *Believe*(x,y), the relation of having desire denoted by *Desire*(x,y) and the relation of having intention, denoted by *Intent*(x,y). In this sense the adopted in OF notion of an agent corresponds to the postulates of a well known paradigm of agency – BDI (belief, desire, intentions) introduced by [Bratman, 1987].

¹⁹ Affectedness is taken in OF as a primitive notion. Among other things, it touches on the Frame Problem, which, however, we find to be outside the scope of the present work.

3.3.3 Established by an Agent

A goal is an *intentional* entity; for some reasons an agent may distinguish some part of reality to be a goal. For example, the state of the world W : the goods are located in Berlin may be recognized as a goal for the manager M in a logistics enterprise. Here we would say that W is a goal for M , $GoalFor(W, M)$. A goal is always established intentionally by some agent, and there is no goal without an agent who recognizes a given chunk of reality as a goal. The effected entities not being a part of the function's goal are called side effects and are discussed in section 3.7.

Note that the *goal-for* relation should not be seen as restricted to the relation of *desire*, as for example in [Chandrasekaran, Josephson, 1997] where the function is considered as a desired or intended effect of a device. As will be demonstrated later, an agent may pick up a given chunk of reality as a goal due to other factors than just desire. Therefore, we use a general relation of *goal establishment*, whose specific types, including *desire*, result from different types of reasons an agent has when establishing a goal.

The inclusion of an agent into the structure of a function has the following consequences on the nature of functions:

1. functions are subjective,
2. functions are not part of the material stratum²⁰,
3. The normativity of functions comes from agents.

Since on the one hand a goal is a function's determinant and on the other hand it involves an agent who establishes it, there are no functions independent of agents. In order to describe any function we have to refer to an agent who establishes its goal. This means in turn that a chunk of reality, which is established by one agent A as a goal is not by itself recognized as such by another agent B , and therefore agent B could not recognize the function resulting in that goal. In this sense, we say that functions are *subjective*. This corresponds to the claim of Searle. In [Searle, 1995] he says that functions are assigned to entities by an external observer, and therefore they are observer relative. Although, this claim refers to the problem of function ascription, we think that it is not only function ascription that is determined by an agent, but also that a function itself is agent-dependent. Agents create functions by establishing goals, instead of finding functions as elements of an objective reality.

Concerning the second point above, since functions are agent dependent they may not be present at a level of reality (or a level of the description of reality) at which agents are not present. Because agents do not belong to the material stratum but to the social or cognitive

²⁰ For details see 'Stratum' in appendix A.

strata, functions cannot be present at the material stratum either. This works well for the example of the business function `to deliver goods` introduced above, which is situated at the social stratum, as it is established by some human in the role of a manager.

However, the elimination of functions from the material stratum raises issues, e.g. the handling of biological functions. Biological functions, in particular their teleological character, like the heart's function of `pumping blood`, are common but also controversial aspects of biology (see [Cummins, 2002] for a critique of teleology and [Gould, Lewontin, 1979] for a critique of the adaptationist programme in biology based upon teleological explanation). If we consider the function of the heart `to pump blood` then there seems to be no agent involved. Moreover, since biology is concerned with the material stratum, in principle biological functions seem to be functions at the material level. If biology is understood as a description of the world that does not refer to any subjectivity but is supposed to be objective, then, according to the assumptions made above, such a description cannot contain functions. In this sense the current framework seems to be inappropriate for biological functions.

However, we think that biology is not only concerned with explanation of the world in purely descriptive terms but, like other sciences, it is also about developing theories. Theories, in turn, being artifacts developed by scientists belong to the cognitive and social strata and involve some subjectivity. Thus, in OF we represent goals of biological functions as goals *established by* particular agents, namely by scientists. In this sense we argue that there are no biological functions at the level of the biological (material) stratum, but that functions are constructed by biologists in their theories, which are intentionally created artifacts belonging to the cognitive and social strata²¹.

Regarding the third point we see that since functions come from agents then the normative character of functions also has its origins in agents establishing functions.

3.3.4 Kinds of Establishing Goals

Definition 2 says that a goal is established by an agent. This means that an agent recognizes some state of reality (or more precisely some chunk of reality, see below) as a goal. The most intuitive case refers to an agent which desires some state of reality and establishes this as his goal. In many functional modeling paradigms discussed in chapter 2 goals are identified with

²¹ Functions belong to the social stratum since they are established by agents, and the notion of an agent belongs to the social stratum. However, functions do not require a society of agents - one separate agent can establish a goal and then recognize a function. In this sense functions belong primarily to the mental stratum.

desired states of reality, e.g. [Chandrasekaran, Josephson, 1997]. In this sense the relation of having a goal is considered as a relation of desiring a goal (preferably by a particular agent, e.g. the designer of the device). However, the establishment of a goal is not only driven by desire. Preliminarily, we can distinguish the following kinds of goal establishment:

- *A goal is desired by an agent.* This corresponds to the most intuitive meaning of the phrase of having a goal - an agent x desires some state of the world y , *Desire*(x,y), and therefore calls it a goal. We assume that having an intention implies a desire.
- *A goal is believed by an agent to have a utility.* An agent does not have to desire a state of reality to call it the goal of somebody or something. In this case the agent establishing the goal does not gain the profits of it. For example, the state in which blood is being pumped is useful for the human organism and as such is distinguished by some agent (say some physician or biologist) from other states as a goal. The goal of `blood being pumped` is not desired by a physician but nevertheless he considers it as the goal of the heart's function of `pumping blood`. Neither is the state of circulating blood recognized as a goal of the heart's function, due to some agent who desires it, i.e. the person, whose organism is under consideration. X may be established by some agent to be a goal, not because it is desired but because an agent believes that it is useful in some context y , *Useful*(x,y)²².
- *A goal provides a good explanation of a given phenomenon.* A particular type of a goal's utility is its epistemic usefulness. Some state of the world x may be crucial for some theory acknowledged by an agent – it may provide an explanation of some phenomenon y , *Explain*(x,y), and thus it may be recognized as the goal of a function. For example, the fact that hemoglobin transports oxygen from the lungs to other parts of the body may be considered as an answer to the question “why does blood contain hemoglobin?” relevant in the context of some biological theory. That kind of explanation is called functional explanation and is broadly discussed e.g. in the context of biology.

In conclusion we can say that a chunk of reality is a goal for some agent, either when he desires it or believes that the state is useful in some context, or that it provides an explanation of some phenomena, formally,

²² A goal is believed by an agent to be useful in some context; this, however, does not imply that it is really useful in that context.

$$\begin{aligned}
\text{GoalFor}(x,y) \leftrightarrow & \text{Agent}(y) \wedge \\
& (\text{Desire}(y,x) \vee \exists wvr (\text{Believe}(y,w) \wedge \text{BelCont}(w,r,x,v) \wedge \\
& (r :: \text{Useful} \vee r :: \text{Explain}))). \tag{2}
\end{aligned}$$

The relation $\text{Believe}(x,y)$ has the meaning that an agent x believes in y . The content of the belief b is depicted by the predicate $\text{BelCont}(b, R, a_1 \dots a_n)$ where R is an n -place relation and a_1, \dots, a_n are arguments of R . In the above definition the relation r is either the *useful* or the *explanation* relation. Thus the content of the belief is either “ x is useful for v ” or “ x explains v ”.

3.3.5 Priority of Goals and Functions

Not all goals are equally important and thus not every function has an equal priority. For example, the heart’s function of `pumping blood` is more important than the function of `producing beat sounds`. In fact due to the lack of good reasons the second could be considered not to be a function of the heart. Common factors determining the priority of a goal are the following:

- the reasons underlying the goal,
- the reliability and number of agents who establishes the goal.

The reliability of an agent is the value of the social trust, which an agent has and it depends on the role of the agent in a given society.

The priority of functions is of particular relevance in the context of function conflicts. Consider the situation where the requirements and the triggers²³ of two conflicting functions are fulfilled. Which function should be realized in such a case? Intuitively, one could say that the first to be realized is the function which is more important – in our terms one that has a higher priority. The order in which functions are realized is given by the order of goal priorities; the higher the priority of a goal the earlier the position of the function in the queue for realization.

²³ Triggers are discussed in section 3.4.

3.3.6 Arbitrary Chunk of Reality

Often a goal is called a goal state or a situation which could suggest considering goals only as situations. For example, the goal of the function `to bring peace to the world` is the situation of the world peace.

For our part, however, goals do not refer only to situations. Situations in GFO are considered as presential comprehensible wholes which are complex entities having a high degree of independence in comparison to other types of entities. However, as the example of the previous section has shown, a goal can consist of one binary relation only, which cannot be considered as a whole situation. The goal of the function `to paint a wall` is not a complex entity (an entity composed of two or more entities), but it comprises only a binary relation that holds between a `wall` and `paint`. Note that in the GFO framework we do not consider relations as mere collections of their arguments, but as concrete entities that glue their arguments together. In this sense the relation of being covered by paint is not a pair (*wall*, *paint*) but rather it is an entity per se gluing them together.

An agent, when establishing a goal of a function, is not restricted to any particular type of entity, but may choose an entity of an arbitrary ontological kind to be a goal. Let us consider a particular type of goal establishment, namely desiring. The object of my desire may be a whole situoid, e.g. `holidays in the mountains` but I may also desire a tiny part of reality, like `the color of the wall in my room is green`. The former is a situoid, whereas the goal of the function `to paint a wall green` is a presential, individual value of the individual property of an individual wall. We find no common feature of those two entities, apart from the fact that they are both goals for some agent and may become goals of certain functions. A goal, therefore, can be an arbitrary ontological category, which we call an *arbitrary chunk of reality*²⁴. Table 3 gives some examples of various ontological kinds that may play the role of the goal of a function.

The final remark regarding the nature of a goal to be made here concerns the issue of universal goals. We have put no constraints on the ontological status of goals, and hence we permit both individual and universal goals. An individual goal refers to a chunk of reality with "fixed"/particular constituents, whereas a universal goal refers to a chunk of reality where

²⁴ To be precise one should say that actually a goal is not an arbitrary chunk of reality, since, as stated in definition 2, it is an intentional and relational entity which refers to a chunk of reality once that the goal is established. However since in the current work we refer mainly to the latter, for the sake of the simplicity of language we call the latter the goal. This seems to be well justified even in natural language where entities of the world are called goals, e.g. one can point to a painted wall and say "this was my goal".

some constituents are merely constrained by universals but not individually identified. In other words the individual goal is composed of individuals only, and refers to exactly one individual chunk of reality, whereas the universal goal is also composed of universals and may refer to more than one individual chunk of reality. For example, the goal at time T the wall W is painted with paint P is an individual goal, referring to an individual presential relator between W and P located at time boundary T. In contrast, the goal a wall is painted is a universal goal, constrained by the universal relation *cover-by* between the universals wall and paint, which refer to the set of all relators gluing some paint with some wall.

Function	Goal	GFO category
to pump blood	blood is being pumped	process (situid)
to deliver goods to Rome	goods are located in Rome	presential relator
to paint the wall red	redness of the wall	property value
to build a house	a house	presential

Table 3. Various GFO categories may play a role of function goals.

3.3.7 Final State

The goal of a function can refer to an arbitrary chunk of reality and, as shown in the previous section, often it is not a comprehensible whole. In the case of the function to paint the wall the goal is a binary relator, i.e. the instance of the covering relation. As a relator this goal requires two relatas - the wall and paint. If we consider the relator of covering together with the wall and paint, this yields a new entity, namely the wall being covered with paint. An entity of this kind in GFO is called a *fact*, and is considered to be a comprehensible whole.

In GFO facts are most simple composite entities considered as wholes. They are constructed of one relator together with its relata or one property bearer with its property. Facts come in a number of kinds depending on the kind of the entities involved in them, e.g. the *presential fact* consists of presential relata and relator, the *processual fact* – of a processual relator and relata etc. Facts can be aggregated in more complex entities, which are considered as wholes as well. An aggregate of presential facts which exist at the same time-boundary is called a *configuration*. A configuration is itself a presential. For example, a wall being painted and polished is a configuration consisting of two facts – wall being painted and wall

being polished. In turn, aggregates of processual facts are called *configuroids* and are considered to be occurrents.

Most complex aggregates considered as wholes are called *situations* and *situoids*. A situation is a configuration which can be comprehended as a whole and satisfies certain conditions of unity, which are imposed by relations and categories associated with the situation. The wall together with *all* its properties and its environment is a situation. A situoid is a processual counterpart of a situation, i.e. it is such an occurrent whose boundaries are situations. Facts, configurations, configuroids, situation and situoids have their universal counterparts, e.g. a universal fact has at least one relatum being a universal, e.g. the fact `John speaks to the clerk` contains the universal `clerk`. All the above notions are underpinned by a derived notion of *complex whole*, denoted by $Whole(x)$ ²⁵.

If a goal is not a comprehensible whole then it may be placed in the context of the comprehensible whole which it is a part of. For that purpose we introduce the notion of *final state*, which can be optionally included into the structure of functions.

Definition 3(Final State). A *final state* x of a function f , denoted by $FSt(x,f)$, is a minimal comprehensible whole that contains as its part a goal of the function f . Formally,

$$FSt(x,y) \rightarrow Whole(x) \wedge Fu(y) \wedge \exists u(GoalOf(u,y) \wedge Part(u,x)). \quad (3)$$

A final state is a minimal comprehensive whole; hence a final state cannot have a final state as its part. Formally,

$$FSt(x,y) \rightarrow \neg \exists z(FSt(z,y) \wedge Part(z,x)). \quad (4)$$

For the goal being the `covered-by` relator a final state is the fact of wall being covered with paint, but it is not a configuration or situation comprising that fact.

3.3.8 Complexity of Functions

Analysis of the final states permits to distinguish basic function from complex functions, later called non-basic as well.

²⁵ $Whole(x) \leftrightarrow Fact(x) \vee Config(x) \vee Configu(x) \vee Sit(x) \vee Situ(x)$

Definition 4 (Basic Function). A function f is called a *basic function*, and denoted by $Fu_{Basic}(f)$, iff the final state of f is a single fact.

$$Fu_{Basic}(x) \leftrightarrow \forall yz(FSt(y,x) \wedge FSt(z,x) \rightarrow Fact(x) \wedge Fact(y) \wedge y = z). \quad (5)$$

Definition 5 (Complex Function). A function f is called a *complex function*, and denoted by $Fu_{Comp}(f)$, iff it is not basic.

$$Fu_{Comp}(x) \leftrightarrow Fu(x) \wedge \neg Fu_{Basic}(x). \quad (6)$$

Basic functions are the most elementary functions, which in functional hierarchy always appear as the bottommost and non-decomposable. The goal of a basic function is called a basic goal and it consists of a single property value, or a single relator, e.g. the goal of the function to deliver goods to Rome comprises only one basic entity, namely the *located-in* relator.

Every non-basic function can be decomposed to basic functions by the decomposition of its goals. For example, the complex function to deliver undamaged goods to Rome, with the final state which is a configuration consisting of two facts - goods are in Rome and goods are not damaged is an aggregate of two basic functions: to deliver goods to Rome and to protect goods.

The final states of non-basic functions may compose one *coherent entity* or its parts may remain unrelated²⁶. The functions of the former kind we call *coherent functions*, and contrast them with the *multiple-goal* functions.

Definition 6 (Coherent Function). A function f is called a coherent function, and denoted by $Fu_{Coh}(x)$, iff all final states of f compose a coherent entity. Formally,

$$Fu_{Coh}(x) \leftrightarrow \exists y(Coh(y) \wedge \forall z(FSt(z,x) \rightarrow CPart(z,y))). \quad (7)$$

From the above definition and the definitions of coherent entity and fact, it also follows that all basic functions are also coherent entities.

²⁶ By a *coherent entity*, denoted by $Coh(x)$, we understand such a complex entity (a whole) that all its constituent parts are interrelated: $Coh(x) \leftrightarrow \forall yz(CPart(y,x) \wedge CPart(z,x) \wedge z \neq y \wedge x = y + z \rightarrow Rel(y,z))$. The relation *Rel* is a root relation of the GFO relations hierarchy.

Definition 7 (Multiple Goal Function). Every function which is not a coherent function is called a multiple-goal function, and denoted by $Fu_{MultGoal}(x)$. Formally,

$$Fu_{MultGoal}(x) \leftrightarrow Fu(x) \wedge \neg Fu_{Coh}(x). \quad (8)$$

For example, the final state of the function `to deliver undamaged goods to Rome` is a configuration of two facts involving the same goods. In contrast the function `to deliver goods to Rome by plane` has two basic goals, whose corresponding final states do not compose one coherent entity. The first final state, `the goods are in Rome`, is a presential composed of the presential `goods`, the presential `Rome` and the relation `located-in` that holds between them at some time boundary. The later final state is the process of transportation involving `goods` and `plane` considered as persistants participating in the process. Therefore, the final states of that function do not form one coherent entity but instead two distinct entities: (i) a configuration C supporting the fact that the mail is in Rome and (ii) a configuroid (or a complex process) C' of transporting goods by plane. Note that C and C' are independent of each other and have different temporal extensions.

3.3.9 Restrictions on Functions

Often a goal of a function is distinguished from the restriction of its realization. The former could be considered as *what* should be done, whereas the latter *how* it should be done. For example, the function `of transporting goods by car` could be understood as a function `to transport goods` restricted by the constraint: `a car should be used for transportation`. Intuitively, two general types of restrictions can be distinguished. One, concerning the realization of the function, as in the example above, and the other concerning the output of the function. To illustrate the latter consider the function `to deliver goods` restricted by the condition that goods should not be damaged. Hence, the goal `goods are in B` is restricted to the goal `goods are in B AND goods are NOT damaged`.

As the reader could already observe in the previous section, we do not include the distinction between the goal and the restriction in OF. According to accepted assumptions a function is not about how something is to be done but about what is to be done. We assume that everything which is intended to be affected by a function is the goal of the function, thus we do not distinguish the output from the goal as [Eriksson, Penker, 2000] do. Therefore both types of restrictions are considered to be merely goals of functions. That is, the function `to transport goods to B by car` is interpreted as a multiple goal function having two

goals - the goods are located in B and the goods are being transported by a car. If any of its goals is not fulfilled, then the function is not realized.

A particular type of function restriction is the time constraint. Take for example a logistics company in which, due to regular delays, the function to deliver goods in 24 hours was introduced. This is a non-basic function, with the goals G : goods are in destination location and G' : goods are in the destination location not later than in 24 hours. The second goal concerns the temporal location of the presential configuration. The basic goal referring to the temporal location we call a *time frame* of the function.

An additional phenomenon concerning restrictions is that they are often considered to be secondary to the goal or less important than the goal. For example, one may claim that it is more important that goods are delivered than the fact that it is done by car. This, however, can be adequately represented in OF by means of priorities which can be attributed to goals.

3.4 Requirements

The interpretation of function as a teleological entity raises the questions if a goal alone determines the function or are there additional determinants needed?

In our opinion the goal alone does not determine the function, since the same state may be a goal of two different functions, and the relation between a function and a goal is not one-to-one. For example, the function of transporting goods to Rome and the function of preventing goods from leaving Rome have the same goal, which is a state, in which the goods are in Rome, but clearly those are two different functions. Those functions differ not with respect to the goals they result in, but in their initial states. In the first case goods are expected not to be in Rome, whereas in the second they are expected to be there. The information about an initial state may be present in the function structure in two ways:

1. It could be stated explicitly in a goal, i.e. the goal of the former function may have the conditional form: If the goods are in Rome (initial state), then they should not leave Rome (goal state). Here, the goal is not considered as a state of the world, which will appear, when the function is realized, but instead it is a conditional statement comprising the conditions and the goal state. This interpretation of a goal, which, however, we do not follow, is adopted e.g. in CFRL, where a functional goal comprises a whole causal process of function realization, including both its initial state and final state ([Iwasaki et al., 1995] p. 13).

2. It is introduced as an additional determinant of a function, beside a goal. It is often implicitly indicated by the verb in the function's label. For example, in the label `to prevent goods from being sent` the verb `prevent` implicitly indicates that the goods are initially in Rome.

Since we consider a goal as an intentional entity referring to a chunk of the reality in which a function results, and not as a conditional statement, we see that it is insufficient to represent a function only by reference to a goal. Therefore, we introduce *requirements* as an additional component of the function structure.

Definition 8 (Function Requirements). *Requirements* of the function f are an intentional entity referring to a chunk of reality which should be present if the function f is to be realized.

Later in the text we refer to *requirements* also in terms of an *initial state*²⁷ or *conditions* of the function, which is denoted by $Req(x,y)$, having the reading that x is the requirement of function y . Each function requires some conditions to be fulfilled in order to enable its realization. Those conditions are common for all individual realizations of the function²⁸. For example, every realization of the function `to hammer nails` requires that there are available nails and some physical object, to which nails should be hammered. Those conditions do not impose what is the type of the object, into which nails should be hammered, although some conditions of it can be given, for example that its hardness must be lower than the hardness of nails. The conditions remain realization-independent: they are equally valid for hammering nails into a wooden table as into a brick wall. We say that all of those necessary conditions compose the requirements of a function.

Requirements, by analogy to the goal, are said to form a chunk of reality, since it seems that there are no restrictions on the ontological character of the entity or entities that form them. Fairly good candidates for requirements in the context of GFO seem to be configuroids and configurations. For example, in case of the function of hammering nails, the requirements form a configuration containing two facts: `nails are present` and `a physical object, for nails to be hammered in, is present`.

²⁷ The initial state can be considered as a minimal comprehensible whole of which requirements are part, analogically to the final state in case of a goal. However, this distinction is not made explicit in the present work since it is not as relevant as the earlier-introduced distinction between a goal and a final state.

²⁸ The notion of an individual realization of a function is discussed in section 5.2, whereas the distinction between individual and universal functions is drawn in section 4.2.1.

However, we do not want to exclude the functions that have simpler requirements, i.e. those consisting only of one simple entity.

Often the label of a function refers itself to the requirements of the function. For instance, the label “*to hammer nails*” refers to nails, which are the function operand, and which must be present in order to realize the function. An operand, however, is not always present in the requirements, for example in case of the label: “*to prevent flood*”, where the operand `flood` is not part of the requirements.

The role an entity plays in the context of a function determines the type of requirements that concern it. The following types of requirements are distinguished²⁹:

- The operand requirements denoted by $Req_{Op}(x,y)$, having the meaning that x is an operand requirement of function y .
- The environmental requirements denoted by $Req_{Env}(x,y)$, having the meaning that x is an environment requirement of function y .
- Functional item requirements denoted by $Req_F(x,y)$, having the meaning that x is a functional item’s requirement of function y ³⁰.

The operand’s requirements are those requirements that concern an operand, for example in the function of `hammering nail`, the operand `nail` is expected to have a particular shape and to be reasonably tough in comparison to the object into which it is to be hammered to. The functional item requirements are those requirements that concern an entity playing the role of a functional item, which is an entity executing a function. In our example a functional item is required to have enough force to hammer nails.

Finally, the environmental requirements are all those requirements, which are neither operand nor functional item requirements. In our example environmental requirements are e.g. those given by the background theory like the laws of physics.

Requirements are necessary but not sufficient conditions for the realization of a function. Apart from them, for a function to be realized what is needed is a *trigger*, denoted by the predicate $Trig(x,y)$ having the reading that x is a trigger of function y . The notion of a trigger is present, among others, in the CASE*Method [Barker, Longman, 1992]. In OF a trigger, together with the requirements, provide a necessary and sufficient condition for the function realization; in this sense a trigger can be understood as a direct cause of the realization of a function. For instance, the realization of the function of `transporting goods` may be triggered by the event `the invoice is signed` or by the event the

²⁹ $Req(x,y) \rightarrow (Req_{Op}(x,y) \vee Req_F(x,y) \vee Req_{Env}(x,y))$.

³⁰ The notion of functional item is defined and discussed in section 3.6.

phone order. In contrast to requirements a trigger is not considered to be a part of the function structure. Therefore, a function remains the same regardless of having different triggers in different realizations. For instance, the function of transporting goods triggered by the event the invoice is signed, and the function of transporting goods triggered by the phone order, remain in our framework the same function, to which different triggers are applied. This enables us to represent functions independently of their particular applications.

3.5 Temporal Extensions of Functions

After introducing the notions of requirements, goal and the basic function we will now provide the classification of basic functions reflecting the relation of functions to time. The issue is of relevance since firstly it permits to separate the notion of function from the notion of process, and secondly it enables the identification of the three kinds of function realization.

Concerning the first point we see that functions are commonly related to processes and treated as particular types of processes. We discuss the interdependence of functions and processes in detail in section 7.3.1, as well as in the context of function realization in section 5.8. Herein we mention only two approaches that follow this line, namely Loebe's approach [Loebe, 2003] and the CASE*Method.

In [Loebe, 2003] a function is treated as the universal of a processual role. A processual role is a layer of the process containing a persistant. Since a process layer is considered to be a part of the process which is a process itself, therefore a function is considered to be a particular type of a process universal, namely one whose instances (1) contain a persistant and (2) are layers of some process. For instance from the universal process of painting a wall containing among others a painter, a wall and paint, one may cut out a layer containing a painter and its behavior, which can be understood in terms of a painter's role in that process. This process layer according to Loebe is to be understood as the function of a painter.

Similar ontological assumptions yield the solution adopted in the CASE*Method [Barker, Longman, 1992], where the difference between functions and processes is that the latter contain the mechanism of realization and the former are mechanism-free processes³¹. Here, functions could also be considered as particular layers of bigger processes, namely ones which are free of the realization mechanisms.

To illustrate the above let us consider the process *P* of transporting goods with a car. We could cut off the layer of that process, which does not contain a car but

³¹ See section 2.2.1 for a detailed discussion)

only goods being transported and call this layer a function F : to transport goods. In this sense function F is a layer of a process P and thus is a process itself. Although we follow a very similar strategy in defining the realization of functions (see chapter 5), yet we argue that functions are not processes.

Understanding functions as processes requires functions to be time-extended. This meets the intuitions behind functions such as to transport goods, or to pump blood, where in both cases the function is recognized as extended in time. However, we think that functions do not have to be time extended-entities. To demonstrate this let us have a closer look at the relation of function to time. When discussing this relation we recognize two important factors:

1. time-location of the requirements
2. time-location of the goal

For the sake of simplicity we do not take into account multiple-goal functions but we concentrate only on basic functions. When considering points (1) and (2) in terms of GFO, we see that the requirements and the goal of a function can have one of two types of time locations: each can be either a time extended process, projected on the framing chronoid or a presential, fully present at a time boundary. From the number of all possible combinations of pairs of the temporal extensions of the requirements and the goal we excluded as unreasonable those where a goal started before the start of requirements and where it ended before the end of requirements. This gives us ten reasonable combinations of time locations of the requirements and the goal, of which three will be analyzed in detail three (figure 13).

In the first two cases both the requirements and the goal are presentials. In the first case the requirements are located before the goal. For example, in the function to transport goods, the requirements are the presential configuration of goods located in A at some time boundary tb_1 , whereas the goal (final state) comprises goods being located in location B at some time boundary tb_2 . Typically, we assume that transportation is not instantaneous, thus tb_1 and tb_2 not only are not coincident but moreover $tb_1 < tb_2$. We call functions of that kind *sequential* functions as the requirements and the goal are in the temporal sequence.

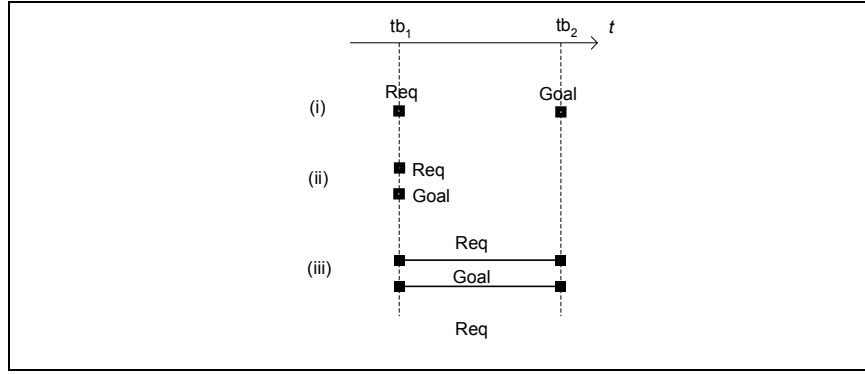


Figure 13. Three combinations of time locations of the requirements and the goal of a function. The arrow indicates the time line. Points represent presentials located at time boundaries, lines joining the points represent processes. Requirements are labeled ‘Req’, while goals - ‘Goal’.

Definition 9 (Sequential Function). A basic function f is called a *sequential* function, denoted by $Fu_{Seq}(x)$, iff the requirements and the goal of f are presentials and the requirements are present before the goal.

$$\begin{aligned}
 Fu_{Seq}(x) &\leftrightarrow Fu_{Basic}(x) \wedge \\
 &\forall yz (Req(y,x) \wedge GoalOf(z,x) \rightarrow Pres(y) \wedge Pres(z) \wedge \\
 &\quad \forall st (At(y,s) \wedge At(z,t) \rightarrow s < t)). \quad (9)
 \end{aligned}$$

In the second case presented in figure 13 the goal and the requirements are also presentials but are not sequentially ordered in time. For the presential requirements and the goal it is not necessary that the goal occurs *after* the requirements, but instead they may be present on the same time boundary. Although this case seems to be odd at the first glance, we will demonstrate that it is well justified.

Consider the function F to camouflage a tank at a battlefield. The requirements and the goal (and the final state) of the function could be defined as follows, $REQ(F)$: a tank is at a battlefield, $FST(F)$: a tank is camouflaged at a battlefield, $GOAL(F)$: the camouflaged-at relation. The realization of this function may be provided by an appropriate type of covering. Due to its chemical structure it may absorb the heat generated by the tank’s engine, and due to an appropriate color and pattern it makes a tank hard to be spotted by an observer. It seems natural to say then that the covering of a tank realizes the function to camouflage a tank.

Now, suppose we consider an individual tank as a presential at a given time boundary t . We find that at that time boundary a tank is covered by covering c . Thus in GFO terms, what we find at the time boundary t is a situation that fulfils the requirements of the function (tank is at a battlefield) and the relation of being camouflaged, camouflaged-

$in(tank, battlefield)$. Both the requirements and the goal are presentials and parts of the same situation, thus, they are located on the same time boundary.

This example shows that the realization of a function is not necessarily a time extended process, and thus the notion of function should not be identified with a process. Functions of that kind we call *instantaneous* functions and they refer to passive functions discussed in [Keuneke, 1989; Chandrasekaran, 1994b].

Definition 10 (Instantaneous Function). A basic function f is called an *instantaneous* function and is denoted by $Fu_{Instant}(f)$ iff the requirements and the goal of f are presentials located on the same or on the coinciding time boundaries. Formally,

$$Fu_{Instant}(x) \leftrightarrow Fu_{Basic}(x) \wedge \forall yz (Req(y,x) \wedge GoalOf(z,x) \rightarrow Pres(y) \wedge Pres(z) \wedge \forall st (At(y,s) \wedge At(z,t) \rightarrow s = t \vee Coinc(s,t))). \quad (10)$$

The requirements and the goal of an instantaneous function can be generalized to time extended entities, which brings us to the third case presented in figure 13. In this case both the requirements and the goal are processes having equal temporal locations. Functions of that kind we call *continuous* functions.

Definition 11 (Continuous Function). A basic function f is called a *continuous* function and is denoted by $Fu_{Contin}(x)$ iff the requirements and the goal of f are processes having the common start and ending.

$$Fu_{Contin}(x) \leftrightarrow Fu_{Basic}(x) \wedge \forall yz (Req(y,x) \wedge GoalOf(z,x) \rightarrow Proc(y) \wedge Proc(z) \wedge ProcStarts(y,z) \wedge ProcEnds(y,z)). \quad (11)$$

To illustrate this kind of function the camouflage function may be modified to the following: to camouflage a tank in the battlefield overnight. Here, the requirements are the process of the tank being in the battlefield overnight, and the goal is the (time extended) state of the tank being camouflaged overnight. Both processes have the same temporal extensions, namely from sunset till dawn.

This particular continuous function can be reconstructed by a number of instantaneous functions. The tank is camouflaged overnight if it is camouflaged at every moment of the night. This, however, is not the case with all continuous functions. Take for example the

function to pump blood. The requirement of this function is a presence of blood in the circulatory system and the goal is the process of blood being pumped. Note that the presence of blood in the circulatory system should be considered here not as a presential but as a process extended in time which co-occurs with the process of blood being pumped.

Intuitively, this function says that whenever blood is provided, it should be in the state of being pumped. It cannot be reduced to the number of presential functions as was done in the case of the camouflage function above. There the goal was the static relation of `camouflaged-in`, which could have been considered on the presential level as well. Here, the goal is the movement of blood, which is a process and cannot be interpreted on the presential level. If the process of blood being pumped is projected on the time boundary of its framing chronoid then there would be found a presential blood participating in that process. However, on the level of presentials there will not be found the movement of blood, as it is a process. Thus, we see that not all cases of time extended functions can be reduced to presential functions.

From the above three function kinds some other kinds can be constructed. For instance the requirements of the function could be a presential and the goal of the function – a process, which starts after the requirements. Consider as an example the function of the house construction, the requirements of this function is a presential configuration of required materials. The goal of the function is the time extended presence of a house.

This function can be decomposed into two functions – a sequential and a continuous function. First we observe that the function of constructing the house is a sequential function in which the goal is a presential house. However, a house is not only intended to be ready when the construction process ends, but moreover it should persist throughout a given period of time. This can be represented by the continuous function of maintaining the house. In this sense it is a combination of the sequential and continuous function.

3.6 Functional Item

So far we have represented functions by means of the requirements, which are an input of the function, the goal which is an output of the function and the function's basic kind. This representation resembles the input-output view on functions. However, it is in our opinion not sufficient to represent functions only as the input-output pairs, since this loses the important feature of functions, which is their dependency. Treating a function as a dependent entity supports the intuitions that a function is always *of* something. Thus the goods transportation understood only in terms of an input and an output is not a function but

rather a teleologically interpreted process. Moreover, the lack in the function structure of the entity to which a function is assigned results in ambiguities and does not permit to determine the function precisely, because it is possible that two different functions have exactly the same requirements and the same goal. Consider for example two functions F and F' :

$LABEL(F) = \text{"to deliver an item to A"}$

$REQ(F) = Ph(x) \wedge x :: item \wedge \neg located(x,A)$

$GOAL(F) = located(x,A)$

$LABEL(F') = \text{"to be delivered to A"}$

$REQ(F') = Ph(x) \wedge x :: item \wedge \neg located(x,A)$

$GOAL(F') = located(x,A)$

Both functions have the same requirements: item is not located in A, and the same goal: item is located in A, thus F and F' in light of previous considerations should be considered as the same function. However, the difference between F and F' is obvious. The first is the function of somebody (something) who is supposed to deliver an item, while the second is the function of an item that ought to be delivered. Therefore, we see that the function specification must not only indicate the input and the output but also the entity, which realizes the function. That entity, let it be x , we will call a *functional item* of a function f , and denote by $Fl(x,f)$.

In the literature on functional modeling a device can be found to be a typical counterpart of our functional item. Function is therefore defined as a function *of* a device. For reasons presented in section 2.1.5 we find this solution problematic. To give a brief summary of its drawbacks: firstly, it assigns functions only to devices, but clearly not only devices realize functions, and secondly and most importantly, it makes functions realization-dependent. For example, if a device is a part of a function specification then, the function of transporting people realized by a car must be considered to be different from the function of transporting people realized by a plane, since a car and a plane are two different devices³². But this makes functions realization-dependent and does not permit to model functions independently of their realizations.

³² If we consider different brands of e.g. cars to be different devices, then it makes functions even more realization vulnerable: Fiat's functions of transporting people and Renault's function of transporting people should be considered as two different functions then.

Thus, there arises quite a dilemma: on the one hand the functional representation should assign a function to its realizer, but on the other hand it should be independent of it. To solve the dilemma we will refer to the concept of *role*.

3.6.1 Role

We adopt here the pattern of roles developed by Loebe [Loebe, 2003; Loebe, 2005] and incorporated into GFO, which is coherent with other approaches, e.g. with the Eriksson and Penker's Actor-Role pattern ([Eriksson, Penker, 2000] pp. 191-197) (see figure 14). According to Loebe a role is an entity played by some *role-filler* which is said to have a role in some *role-context*.

$$Role(x) \leftrightarrow \exists yz (HasRole(y,x) \wedge RoleIn(x,z)). \quad (12)$$

For instance a `person` may have a role of `student` in the context of a `university`. The role-context is the main criterion of classifying roles. The distinguished types of roles are *relational roles*, whose context is a relation, *processual roles*, whose context is a process and which describes participation in a process; and *social roles* which corresponds to the involvement of a social object within some society.

Although roles should not be confused with properties (and property values), we find those entities similar. Firstly, they both describe the entity they refer to, called a role player or a property bearer, respectively. Secondly, they are both dependent on that entity. Finally, they can both be understood as aspects of that entity³³. In this sense a role is an aspect of an entity against some (external to the entity) context or, to be more precise, a role selects some of the aspects (properties) of an entity with respect to some context. For example, the role of a `driver` selects such aspects of a `person` as `driving skills` or `driving experience`.

Roles can be classified on the basis of the nature of properties they select. Thus a `driver` role selects the properties we call structural, i.e. belonging to the structure of the role-filler and as such can be called a structural role. Moreover, some roles are a mixture of the structural and role aspects, where the structural aspects come from the role player and the role aspects from the context of a role [Loebe, 2005]. For example, the role of `father` contains both the structural aspect of a `male` and a role aspect of a `parent`. In contrast, `parent` is a *pure*

³³ Similarly e.g. in UML both roles (called there associations ends) and attributes are generalized under the notion of property [OMG, 2004b].

role, which does not contain any structural aspect of its role-player but is defined purely by its context.

Finally, we observe that it is common for the comparison of entities to be based on and restricted to the scope given by their roles, hence one can compare two persons with respect to their properties such as driving skills or driving experience, which are captured by their driver roles.

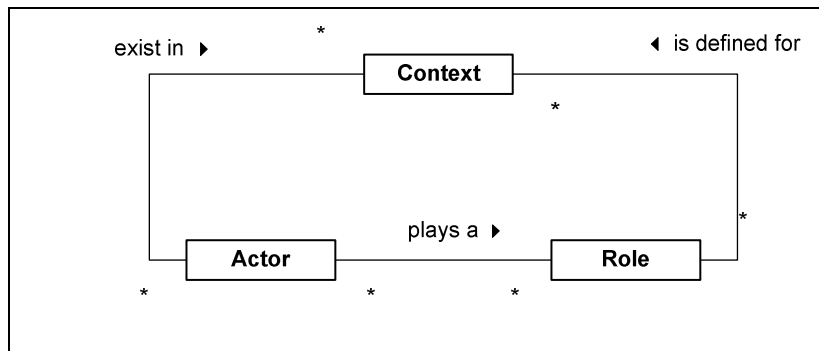


Figure 14. Part of the business actor-role pattern developed by Eriksson and Penker ([Eriksson, Penker, 2000] p. 193). A role is considered as an entity mediating an actor and a context.

3.6.2 Functional Item as a Role

As observed above we do not want to define the functional item in terms of particular entities (devices) which realize the function in order not to fall into realization-dependent function specification. Instead we will use for that purpose a notion of role: we understand the functional item as the role an entity plays in the context of the realization of function.

For example, a `heart` in the context of the process of `pumping blood` can be considered to be a `blood pump`. A `car` in the context of the process of `transporting goods` can be considered as a `goods transporter`, etc. In this sense both a `blood pump` and a `goods transporter` are roles of those entities in the context of the process which realize functions.

However, not every role is suitable to be a functional item. The functional item should not constrain the realization of the function more than the goal of the function does. If this condition is not fulfilled, then the functional item again would be a source of the realization-dependence in the function structure. For instance, if we consider the function of `pumping blood`, the functional item should not be the role `heart pump` but rather a more general role `pump`, which does not exclude `mechanical hearts` from being its role players. As a solution to this problem we think that the roles naming the entities realizing the function

should be constrained only by the goal of the function. Therefore we define the functional item as follows:

Definition 12 (Functional Item). The *functional item* of a function f indicates the role of entities executing a realization of f , such that all restrictions on realizations imposed by the functional item are dictated also by some goal of f .

The functional item is a purely teleological role in the sense that it abstracts from every aspect of the entity realizing the function, which is not related to the (appropriately defined) goal. It abstracts from everything apart from the goal, and thus does not impose any structural features of the filler as long as the goal does not do it. As such it is a common umbrella for all roles which are constrained not only by the goal but also by the role filler. For instance, `transporter` as a functional item of the function `to transport goods` is a common umbrella for `car-transporter`, `plane-transporter` and all the others realizers of this function. Each of them extends the functional item by structural aspects. For instance, `plane-transporter`, beside the teleological aspect of executing the realization of transportation of goods, contains also the structural aspects of a plane, like `cargo space`, `maximum flight distance`, and others.

A functional item is either a simple entity, which is a role of a single entity, like in case of `teacher`, being a role of `person`, or it can be a more complex entity composed of roles of several entities. A functional item composed of more than one role is called a *complex functional item*:

Definition 13 (Complex Functional Item). A functional item x of a function f is called a complex functional item of f , and denoted by $FI_{Comp}(x, f)$ iff it has as its role proper part more than one role.

$$FI_{Comp}(x, y) \leftrightarrow \exists v w (RolePPart(v, x) \wedge RolePPart(w, x) \wedge v \neq w) \quad (13)$$

For example, a functional item `car-transporter` of the function `to transport goods by car` involves beside a role of a `car` also a `driver`, which is a role of an agent. Often the complex functional item is not just a simple aggregate of its elements but it

³⁴ The notion of proper role part is defined as follows: $RolePPart(x, y) \leftrightarrow Role(x) \wedge Role(y) \wedge PPart(x, y)$. The predicate $PPart(x, y)$ denotes a proper part which is a non-reflexive variant of the part-of relation.

may have the interior structure. For example, in case of `car-transporter`, there is involved a relation of `driving` that holds between a `car` and a `driver`. In this sense complex roles are complex wholes whose relata are roles only³⁵.

3.6.3 Discussion

We have introduced functional item as a pointer to the entities intended to realize the function. However, in order to avoid defining functions in the context of the entities that realize them, we assign functions to their realizers by the mediating role called *functional item*.

Now one could object to our example that `transporter` could in fact be considered as a type of device, whose subtypes are `plane` and `car`. The introduction of `transporter` as a general device concept solves the problem of realization-dependency on the one hand, and on the other it does not force us to resign from the number of approaches that assign functions directly to devices.

There are, however, at least two reasons, for which such a solution is problematic. Firstly, if we organize into one hierarchy the categories of `device`, `transporter`, `car` and `plane` and still want to remain realization-independent in defining functions, we have to provide criteria that permit to define functions by means of some of the concepts of this hierarchy, like `transporter` and prevent from doing so by means of others, here by `car` or `plane`. However, it seems that no such mechanism may be provided. If the hierarchy represents devices then, as long as defeasible subsumption is not considered, there is no basis on which a device, i.e. `transporter`, can be a functional item whereas its subconcepts, `car` and `plane`, cannot.

This solution has also a second drawback, namely such that placing `transporter` in a subsumption hierarchy above the concepts of `car` and `plane` results in the erroneous taxonomy. `Transporter`, as we have seen, is a role, thus by definition for each role x there is some y , which is its role player, and some z , which is a context of the role x . Since `car` and `plane` would be subconcepts of `transporter`, the above should by inheritance hold for them as well. However, it can clearly be seen that it does not. An entity to be a `car` does not require any role player and any context. Thus, we see that the non-role concept cannot be a subconcept of the role concept. Analogous constraint on subsumption taxonomy, although based on different arguments, is given in OntoClean³⁶.

³⁵ $FI_{Comp}(x,y) \rightarrow FI(x,y) \wedge Whole(x)$.

³⁶ OntoClean is a methodology for supporting construction of concept taxonomies [Guarino, Welty, 2004]. It is founded on the ontological meta-properties of rigidity, identity, dependency and unity which

In conclusion we can say that (1) the approaches that define function by reference to a device are fated for representing functions in realization-dependent manner; (2) this problem may not be avoided by considering the role-concept of functional item as a type of device, since it gives no means to differentiate functional items from mere device concepts, and moreover it results in erroneous taxonomy, where role concepts subsume non-role concepts.

3.7 Side Effects

Not all effects of the functions must be established by the agents as goals. In such cases functions are said to have *side effects*. We recognize two kinds of side effects: one which belongs directly to functions and one which belongs to a particular realization of functions. The former we define as follows:

Definition 14 (Function Side Effect). A *function side effect* of a function f , denoted by $SideEf(x, f)$, is a chunk of reality x affected by f , which is not a part of the goal of f .

$$SideEf(x, y) \leftrightarrow Affect(y, x) \wedge \neg \exists z (GoalOf(z, y) \wedge Part(x, z)). \quad (14)$$

Side effects of functions could be taken as those consequences of the goal, which are not intended by an agent. A side effect, then, is everything, which is not a goal but belongs to the very nature of the goal or, in other words, everything which is inseparable from the goal or dependent existentially on the goal. For example, the function to disband the university U , which has a goal that there is no university U , has a side effect that John stops being the professor at U . Here, being a professor is the role of John at university U , which is dependent on John and on U . When U ceases to exist so does the role of John. Thus the function affects the role. However, it does not have to be an intended goal of disbanding the university U to deprive John of his position. In this sense a side effect of the function is an unintended consequence of the goal.

The dependency relation between the goal and the side effect may be different in its nature and in its strength, however for the purpose of the current topic it is not an issue to investigate its nature, and we treat it roughly as an *existential dependency* relation.

provides constraints for taxonomy structures [Welty, Guarino, 2001]. The rigidity based constraint prohibits subsumption of non-roles under roles [Guarino, Welty, 2000].

Whereas function side effects are independent of the particular way of function realization, the second type of side effects is the result of a particular function realization. Consider for example `air pollution`, which is a side effect of the process of transporting people by car, which realizes the function `to transport people`. Here, however, the side effect depends not on the function itself but on the particular realization. Hopefully, if we switch to ecological cars, that particular side effect will be avoided. Side effects of that kind are called *realization side effects*. The definition of the realization side effect is introduced later in section 5.7.2 since it refers to the notions of individual realization, fulfillment and others, which are handled in chapter 5.

3.8 Summary

In the current section we have introduced the structure of functions $STR(x)$ which enables to represent functions independently of their realizations. The structure of the function has four elements, $STR(x) = (LABEL(x), REQ(x), GOAL(x), FITEM(x))$. Every element of $STR(x)$ is called a component of function x , and is denoted by $C(x)$, $C(x) \in STR(x)$.

$LABEL(x)$ is a set of natural language expressions describing function x , the remaining components are defined on the basis of introduced binary relations as follows:

$$REQ(x) = \{y: Req(y,x)\}. \quad (15)$$

$$GOAL(x) = \{y: GoalOf(y,x)\}. \quad (16)$$

$$FITEM(x) = \{y: Fl(y,x)\}. \quad (17)$$

The goal and the requirements provide the ontologically refined view on the input and output approach to functions. The particular type of the goal is the time frame, which provides a temporal restriction on the function. The time frame is such a goal which is a time entity,

$$TFRAM(x) = \{y: GoalOf(y,x) \wedge Te(y)\}. \quad (18)$$

The functional item indicates the entities which are intended to realize the function, without imposing any constraints on them not imposed by a goal.

Since the label $LABEL(x)$ plays only an informative role for human users, we do not use it for determining functions. Functions are determined by the subset of the function structure,

defined as $DETE(x) = STR(x) \setminus \{LABEL(x)\}^{37}$. Each element y of $DETE(x)$ is called a *determinant* of the function x and is denoted by $D(y,x)$. The equality of functions is founded on the equality of their determinants.

$$x =_{Fu} y \leftrightarrow DETE(x) = DETE(y). \quad (19)$$

The equality of functions can be restricted to only some of their determinants, and thus two functions can be equal with respect to the goal, the requirements or the final item:

$$x =_{Gl} y \leftrightarrow GOAL(x) = GOAL(y). \quad (20)$$

$$x =_{Req} y \leftrightarrow REQ(x) = REQ(y). \quad (21)$$

$$x =_{Fi} y \leftrightarrow FITEM(x) = FITEM(y). \quad (22)$$

³⁷ $D(y,x) \leftrightarrow Req(y,x) \vee GoalOf(y,x) \vee Fl(y,x)$.

4 Relations between Functions

4.1 Introduction

So far we have discussed the function considered in isolation, however for the purpose of functional modeling functions are glued by relations into functional models. The primary question then is what kinds of relations connect functions? In this chapter we investigate the following relations:

- *Instantiation*
- *Is-a*
- *Part-of*
- *Realization*
- *Enablement*
- *Support*
- *Prevent*

Instantiation, is-a and part-of are typical ontological relations, commonly discussed and applied. In the context of the functional modeling they are present in several approaches discussed in chapter 2, however there is no consensus on their meaning, and they are seldom defined in a formal way. The relation of realization is not a common ontological notion, but it belongs to the domain of functional modeling, although its interpretation varies across formalisms. Moreover, some additional relations, peculiar for functional representation can be found. These are: enablement, support, and prevent. They are present in some of the AI approaches to functional modeling, e.g. in FCO or MFM.

The aim of the current chapter is to give a formal and a general specification of the relations mentioned, which permits them to be handled across diversified domains.

4.2 Instantiation

Firstly, let us consider the relation of instantiation. It is a common mechanism used in programming, conceptual modeling and in ontologies. Take as examples the UML notion of instantiation, and the RDF property `rdf:type`, utilized in OWL. The distinction between

instantiation and subsumption is often problematic. The problems have their roots already in the linguistic ambiguity, since both relations pass *is-a* test. Informally, it can be said that “an instance *is a* universal”, like in example *John is a person*, as well as that “a subclass *is a* superclass”, i.e. *a human is a mammal*. Thus, both relations are sometimes called *is-a* relations and therefore confused (see [Brachman, 1983] for discussion). Here, in accordance with GFO and many other approaches we consider these two relations being different.

The instantiation relation varies across formalisms. In UML it is the relation holding between the elements of models on different layers of MOF metadata architecture [OMG, 2002], and thus not only holding between objects of layer#1 and classes on layer#2, but also between classes and meta-classes (layer#3); meta-classes and meta-meta-classes (layer#4). In contrast in OWL Lite and OWL DL the instantiation denoted by `rdf:type` holds only between individuals and classes. In OWL Full classes can be considered as individuals, thus analogically to MOF the instantiation between classes is permitted there. In both UML and OWL instantiation is not distinguished from the membership relation. In contrast, in GFO instantiation is “the intensional counterpart of the membership relation as it (instantiation) does not satisfy the *principle of extensionality*” (original emphasis, [Heller et al., 2005]).

In GFO instantiation, denoted by “`::`”, is a binary relation, whose second argument is a universal and the first, called *instance*, is an individual or a universal. Individuals are entities that do not have, and are not permitted to have, instances. Not all entities lacking instances are individuals in GFO framework. For instance, classes, which do not have instances, but elements, are not considered to be individuals. In contrast to universals individuals are located in time and space. However, one should have in mind that it is not a necessary characteristic of individuals since some of them, which in GFO are called *general* or *abstract* individuals, are not located in time and space. Universals on the other hand are defined as entities that have, or may have, instances. Universals all of whose instances are individuals are called *primitive universals*.

Just as in UML and OWL Full, meta-universals, whose instances are universals, are not excluded in GFO. Meta-universals are commonly met, especially in context of biological taxonomies. In the example “Hedgehog is an instance of *species*, Tony is an instance of *hedgehog*”, *species* is a meta-universal, *Hedgehog* is a *primitive universal* and *Tony* is an individual ([Heller et al., 2005], p. 17). In GFO, in contrast to UML, and similarly to OWL Full the instantiation hierarchy is not restricted to only four layers, and therefore higher than meta-meta-universals are permitted³⁸.

³⁸ The discussion on the comparison of meta-architecture of MDA and the meta-architecture of GOL can be found in [Herre, Loebe, 2005].

4.2.1 Individual, Universal Functions and Instantiation

Since the notion of instantiation is closely related to the notions of universal and individual, let us first introduce our understanding of universal and individual functions. We account for both those notions by the references to the function structure.

Definition 15 (Individual Function). A function f is called an *individual function*, and denoted by $IndFu(f)$, iff all its determinants are individually determined.

$$IndFu(x) \leftrightarrow \forall y(D(y,x) \rightarrow Ind(y)). \quad (23)$$

Definition 16 (Universal Function). A function f is called a *universal function*, and denoted by $UniFu(f)$, iff at least one of its determinants is a universal.

$$UniFu(x) \leftrightarrow \exists y(Uni(y) \wedge D(y,x)). \quad (24)$$

The intuition behind the above definitions is that an individual function is such that cannot be instantiated in any dimension, which means that none of its determinants can be instantiated. In this sense the individual function is a function determined by individuals, whereas universal function is determined by at least one universal. Consider for example the function F : to paint a wall and the function F' : to paint by the person A the wall W with the paint P within the time period T . The first is clearly a universal since at least one its determinant, and here in fact all of them, is a universal. In contrast, the determinants of the latter are individually determined. The requirements is the individual configuration of paint P and the wall W located at the left time boundary of the period T . The goal is the painted-with relator gluing the wall W and paint P located at the right time boundary of the period T . Finally, the painter is the functional item which is a role restricted to the individual role filler, namely the person A .

The interesting case of a universal function is such a function whose all determinants but the functional item are individually determined. A function of that kind we call a *primitive universal function*.

Definition 17 (Primitive Universal Function). A function f is called a *primitive universal function*, and denoted by $UniFu_{Prim}(f)$, iff all its determinants apart from the functional item are individual entities.

$$UniFu_{Prim}(x) \leftrightarrow \forall y (GoalOf(y, x) \vee Req(y, x) \rightarrow Ind(y)) \wedge \forall z (Fl(z, x) \rightarrow Uni(z)). \quad (25)$$

Consider the function F'' : to paint the wall W with paint P within the time period T . Here, both the goal and the requirements are the same as in the function F' and are individuals, whereas the functional item is a universal role with undetermined role filler—a painter of wall W with paint P .

In contrast to primitive functions there are functions similar to F , all of whose determinants are universals not containing as their parts individuals. Functions of that kind are called *absolute universal functions* and are defined as follows:

Definition 18 (Absolute Universal Function). A function f is called an *absolute universal function*, denoted by $UniFu_{Ab}(f)$, iff all determinants of f are absolute universal determinants of f . A determinant x of a function f is called an absolute universal determinant of f , denoted by $UniD_{Ab}(x, f)$ iff x is a universal that does not contain any individual as its part³⁹.

³⁹ The ontological analysis of the part-whole relation is a topic in itself, and exceeds the scope of this work. Here in principle we refer to GFO as a framework defining the part-whole relation. It is only worth mentioning that we treat the part-whole relation in a more general sense than it is done typically. Typically, the part-of relation is considered as relation between individuals (see [Guizzardi, 2005] for overview). We, however, do not exclude the part-whole relation between universals, or between universals and individuals. For example, consider the situation S : a car is in a garage. S is clearly a universal since it may be instantiated by an individual situation s_I : my car is now in the garage. On the other hand we may say that a car, which also is a universal (instantiated by an individual My car) is a part of situation S . In this case a universal is a part of a universal. The part-whole relation between universals refers to the problem of principles of concept structures discussed in experimental psychology and in cognitive science [Laurence, Margolis, 1999]. In frames of GFO it is handled by the relation of *categorical part* [Herre, Loebe, 2005].

In the current work we extend the notion of the categorical part to the relation that may hold between a universal and an individual. Thus, we permit an individual that is a categorical part of a universal. For example, an individual My Car is a part of a universal situation My car is in a garage. However, one should not understand that the individual (really existing) physical car is a part of the category, which would sound odd. All ontological entities discussed in the present work, including individuals, are not considered here as entities in the world, but rather as elements of the model which describe the world. In this sense My Car is considered as an individual, which *refers to* some physical individual object in the world not as that object directly.

The relation denoted by $Part(x, y)$ is used in the present work as a general notion of the part-whole relation, which holds either between individuals, universals, or a universal being a whole and its

$$UniD_{Ab}(x,y) \leftrightarrow D(x,y) \wedge \forall z (Part(z,x) \rightarrow Uni(z)). \quad (26)$$

$$UniFu_{Ab}(x) \leftrightarrow \forall y (D(y,x) \rightarrow UniD_{Ab}(y,x)). \quad (27)$$

Now, after distinguishing individual from universal functions we can define the relation of instantiation between functions. Just as the analysis of the function determinants permitted to distinguish universal from individual functions, the instantiation between function determinants is helpful in defining the instantiation of functions.

Let us consider the goal first. The goal a wall is painted is a universal relation of being painted between the universal wall and the universal paint. In turn, the goal the wall W is painted with paint P at time T is an individual presential relator at time boundary T gluing the individual wall W with the individual paint P . It can be easily seen that this relator is an instance of the relation of being painted. In this sense we can say that the latter goal is the instance of the former. The instantiation of goals of two functions is denoted by $x ::_{Gl} y$, and defined as follows:

Definition 19 (Goal Instantiation).

$$x ::_{Gl} y \leftrightarrow \exists vw (GoalOf(v,x) \wedge GoalOf(w,y) \wedge v :: w). \quad (28)$$

Similarly, the instantiation may hold between the requirements of two functions. For instance the requirements of function F' form an individual configuration which is an instance of a universal requirement of F . In most cases, also in the case of function F' , the individual requirements entail an individual goal, but this is not the rule. Consider the function of a construction crew - to construct a house out of given components. Here, requirements are a particular, individual configuration of the given components, whereas the goal is a universal house. The configuration of components does not determine individually the house which will be constructed.

The instantiation of requirements of two functions is denoted by $x ::_{Req} y$, and defined analogously to the goal instantiation.

individual part. The only case which we excluded is such, where a universal is a part of an individual, thus : $Part(x,y) \rightarrow (Uni(x) \wedge Uni(y)) \vee (Ind(x) \wedge Ind(y)) \vee (Ind(x) \wedge Uni(y))$. More on part of relation in GFO can be found in appendix A.

Definition 20 (Requirements Instantiation).

$$x ::_{Req} y \leftrightarrow \exists vw (Req(v, x) \wedge Req(w, y) \wedge v :: w). \quad (29)$$

Considering the functional item we assume that its role-filler usually is not determined by the goal. For example, in the function `to transport goods` the goal is a universal and the functional item is a role of an arbitrary entity which transports goods, i.e. a `transporter`. However, there are also functions in which the goal imposes the filler of the functional item, as for example in the function `to transport goods by car LVB 2040`. Here, the filler of the functional item is individually determined persistent `car LVB 2040`.

On the basis of the relation of instantiation which holds between an individual and a universal role filler of two functional items we introduce the relation of *functional item instantiation* between functions.

Definition 21 (Functional Item Instantiation).

$$x ::_{FI} y \leftrightarrow \exists vwst (FI(v, x) \wedge FI(w, y) \wedge HasRole(s, v) \wedge HasRole(t, w) \wedge s :: t). \quad (30)$$

According to the above definition the function `to transport goods by car` is instantiated with respect to the functional item by the function `to transport goods by car LVB 2040`, since the functional item of the latter has individually determined role player, i.e. `car LVB 2040` which is an instance of the role filler of the functional item of the former, i.e. `car`.

Concluding we can see that a functional item analogously as a goal and requirements can be a universal or an individual. In the former case it may be a universal role with or without individually determined role filler. For instance `transporter` is a universal with non-determined filler, whereas `car LVB 2040 transporter` is a universal comprising `transporter` roles of an individual role filler: `car LVB 2040`. Finally, in the case of individual functions where the goal, the requirements and the role filler of the functional item are individuals the functional item is an individual as well.

Now, after defining the relations of the determinant instantiation we can define the relation of function instantiation.

Definition 22 (Function Instantiation). An individual function f *instantiates* a universal function f' , denoted by $f ::_{Fu} f'$, iff f instantiates f' with respect to all determinants.

$$x ::_{Fu} y \leftrightarrow IndFu(x) \wedge UniFu(y) \wedge x ::_{Fl} y \wedge x ::_{Req} y \wedge x ::_{Gl} y. \quad (31)$$

4.3 Taxonomic Relations

4.3.1 Introduction

The *is-a* relation is the main taxonomic relation, often considered as a backbone of an ontology. It is common across the conceptual modeling and the ontology representation formalisms. For example, in UML the generalization is the relation between classifiers, whereas in OWL the RDFS property `rdfs:subClassOf` is the basic taxonomic constructor that holds between OWL classes.

The semantics of the notion of the *is-a* relation is, however, not clear-cut (for discussion see [Brachman, 1983]). Most often it is taken as a logical implication, for example in OWL, where if *A* is a subclass of *B* then every instance of *A* is an instance of *B*.

In this understanding it may be called an *extensional subsumption* and is contrasted to its *intensional* counterpart, called also structural subsumption, which was introduced by Woods in [Woods, 1991]. In accordance with the intensional subsumption one concept subsumes another not by virtue of a model-theoretic criterion but by virtue of their structures. Concepts in Woods' approach are considered as atomic or composite descriptions. An atomic description consists merely of an atomic concept label. A composite description is of the form: $c_1, \dots, c_k / m_1, \dots, m_n$, where c_i are primary conceptual descriptions, and m_i are the relation-value pairs ($r_i:v_i$) called modifiers ([Woods, 1991], p.50). For example $[person] / ([like]:[golf])$ is a composite description describing a person who likes golf. Structural subsumption is defined on the basis of subsumption of descriptions which is understood as the subsumption of primary conceptual structures and the subsumption of modifiers. One conceptual description $c_1, \dots, c_k / m_1, \dots, m_n$, subsumes the other $c'_1, \dots, c'_k / m'_1, \dots, m'_n$, if each primary conceptual description c_i subsumes some c'_j and each modifier m_i subsumes some m'_j .

For most cases both the extensional and intensional criterion brings the same results. However, those notions are not equivalent. Woods writes:

“For example, one might judge the concepts [polygon with three sides] and [polygon with three angels] to be intensionally distinct, even though they will necessarily have the same extensions in all possible worlds, because that fact does not follow directly from the structure

of the descriptions but has to be deduced from the logic of the domain. If the two expressions were intensionally the same, one would argue, then a proof should not be necessary - it would suffice to examine the meanings to see they are the same.“ ([Woods, 1991], p. 72-73)

Moreover, Woods observes that the intensional subsumption entails the extensional subsumption but not vice versa. In this sense the intensional subsumption is the extensional specialization of the extensional subsumption⁴⁰.

Both the extensional and intensional subsumption have problems when confronted with the prototype theory and with exceptions. The prototype theory have its roots in the research of Rosch and Mervis [Rosch, Mervis, 1975] in the field of cognitive science and experimental psychology, who have found that people's concepts often lack a definition of the necessary and sufficient conditions, but instead they often are structured prototypically. Prototypically structured concepts are depicted by a representative, having the typical features. However, not all representatives of the category share all of the typical features. Representatives lacking them are considered to be *atypical* or *peripheral*. For example a prototype of the bird has wings, has feathers and can fly. All those hold for e.g. eagles, doves, etc. On the other hand ostriches or penguins cannot fly but still are considered to be birds. In this sense the principles of both extensional and intensional subsumption fail, since none of them permit to classify ostriches and penguins as birds.

For that purpose the *defeasible* subsumption was introduced. In the defeasible subsumption not everything that is true for a super-class must be true for a subclass, but it may be cancelled. For example, the property of a bird: *ability to fly* is canceled in case of ostrich and penguin. That mechanism is used also in the object-oriented paradigm where the property of a class may be overwritten by its subclass.

However, the cancellation of properties in subsumption is not free of problems. Brachman in [Brachman, 1985] reported the problems that the cancellation of properties raises for subsumption. He observes that if it is allowed with no restriction to cancel in a subconcept an attribute or an attribute value of the superconcept, then it results in the mishmash in the subsumption hierarchy. For example, if we except non-flying penguins and ostriches as subconcepts of *bird*, then nothing stops us from treating also airplanes as birds, in spite of the fact that they are not living beings, though having wings and being able to fly.

The is-a relation is also required in function modeling and it has been introduced in several approaches, e.g. FCO, although, as far as we know, the semantics of functional subsumption seems not to be analyzed in details in the literature.

⁴⁰ Every instance of the intensional subsumption is an instance of the extensional subsumption as well.

4.3.2 Subsumption, Specialization and Individualization

It is not our intention to contribute to the general discussion about the is-a relation outlined above; instead we intend to provide a formal understanding of taxonomies of functions. In contrast to the relation of instantiation, which is sometimes also labeled by “is a”, the is-a relation holds not between an individual and a universal function but between universal functions only. Three kinds of is-a relation between functions are introduced: *subsumption*, *specialization* and *individualization*.

As in the case of instantiation, all three are founded on the analysis of the relation of function determinants. Let us introduce subsumption first:

Definition 23 (Function Subsumption). A universal function f is *subsumed* by a universal function f' , denoted by $f \subseteq_{Fu} f'$, iff every determinant of f is subsumed by the corresponding determinant of f' .

$$x \subseteq_{Fu} y \leftrightarrow \text{UniFu}(x) \wedge \text{UniFu}(y) \wedge \forall \phi \forall uv (\phi \in \text{DETE} \wedge \phi(v,x) \wedge \phi(u,y) \rightarrow \text{Subsume}(u,v))^{41}. \quad (32)$$

Subsumption of determinants denoted by $\text{Subsume}(x,y)$ is understood here in an extensional sense and is reflexive,

$$\text{Subsume}(x,x). \quad (33)$$

The non-reflexive variant of subsumption of determinants is called here *specialization*, and is denoted by $\text{Specialize}(x,y)$.

$$\text{Specialize}(x,y) \leftrightarrow \text{Subsume}(y,x) \wedge \neg \text{Subsume}(x,y). \quad (34)$$

On the basis of the specialization of determinants the specialization of functions is introduced.

Definition 24 (Function Specialization). A universal function f specializes a universal function f' , denoted by $f \subset_{Fu} f'$, iff all determinants of f are subsumed by the appropriate determinants of f' and at least one determinant of f specializes the appropriate determinant of f' .

⁴¹ The formulas 32, 35, 36 are second order formulas but can be reconstructed in FOL.

$$\begin{aligned}
x \subset_{Fu} y &\leftrightarrow \text{UniFu}(x) \wedge \text{UniFu}(y) \wedge \\
&\forall \phi \forall uv (\phi \in \text{DETE} \wedge \phi(v,x) \wedge \phi(u,y) \rightarrow \text{Subsume}(u,v)) \wedge \\
&\exists \gamma \exists wz (\gamma \in \text{DETE} \wedge \gamma(w,x) \wedge \gamma(z,y) \wedge \text{Specialize}(w,z)).
\end{aligned} \tag{35}$$

If determinants of two functions cross-specialize, then the subsumption between those functions does not hold in any of the directions. For example, if the requirements of the function F specializes the given requirements of the function G and the goal of G specializes the goal of F , then neither F subsumes G nor G subsumes F .

Different kinds of functional specialization can be distinguished by references to the kind of determinant which is specialized. These are *goal specialization*, *requirements specialization*, and *functional item specialization*. Goal specialization usually entails requirements specialization, since requirements are to some extent dependent on the goal. For example, the function `to deliver mail` is a goal specialization of the function `to deliver item`, since the goal of the former: `mail is delivered` is a specialization of the latter: `item is delivered`. Concurrently the specialization holds between the requirements of those functions, since the first function requires a `mail` and the second an `item` to be present, and `mail` specializes `item`. In turn, the goal and requirement specializations entail the functional item specialization.

Yet apart from the subsumption and specialization of functions there is another taxonomic relation between universal functions, namely *individualization*. Determinants of universal functions can not only be related by subsumption and specialization but, as it was shown in the previous section, by instantiation. For example, the goal and the requirements of the absolute universal function `to renovate a house` are instantiated by the goal and the requirements of the primitive universal function `to renovate the White House within time period T`. The latter is not an instance of the former since it is not an individual – its functional item (the `White House renovator`) does not have individually determined role fillers. In this sense one can say that the second function is *a kind of* the former function. However, this is-a is neither founded on the subsumption nor on the specialization of function determinants but on their instantiation. To cover cases of that kind we introduce the next taxonomic relation called functional individualization.

Definition 25 (Function Individualization). A universal function f is an *individualization* of a universal function f' , denoted by $f \Rightarrow_{Fu} f'$, iff at least one of individual determinants of f instantiates a corresponding universal determinant of f' and the remaining determinants of f are equal to the corresponding determinants of f' .

$$x \Rightarrow_{Fu} y \leftrightarrow \text{UniFu}(x) \wedge \text{UniFu}(y) \wedge \\ \exists U (U \subseteq \text{DETE} \wedge \forall \phi (\phi \in U \rightarrow x ::_{\phi} y) \wedge \forall \gamma (\gamma \in \text{DETE} \setminus U \rightarrow x =_{\gamma} y)). \quad (36)$$

In contrast to instantiation, individualization holds between universals and not between a universal and an individual function. For instance, both functions `to renovate a house` and `to renovate the White House` are of a universal character. Moreover, analogously to subsumption and specialization the extension of the latter function is the subset of the extension of the former. However, in contrast to subsumption and specialization, individualization is founded on the relation of instantiation between determinants. Here, it is founded on the instantiation of the universal `house` by the individual `the White House`.

Intuitively, the difference between specialization and individualization is that the first restricts a universal function by the restriction of the kind of (at least) one of its determinants, whereas the second restricts a universal function by the instantiation of its determinant. However, the instantiation of one determinant does not imply the instantiation of the whole function.

Concluding we can see that all three kinds of is-a relations – subsumption, specialization and individualization are taxonomic relations for structuring universal functions. All three have an intensional character - they are identified by the analysis of functions structures, not only by the analysis of functions extensions. In addition, all of them fulfill the extensional condition. Cancellation is not permitted and thus defeasible taxonomical links are not supported in OF.

4.4 Part-Whole Relation

4.4.1 Introduction

It is common in the literature on functional modeling to consider the part-of relation between functions. For example, in MFM [Lind, 1994] introduces the *whole-part* relations for all levels of functional description – between goals, between functions, between behaviors and between structures. Of our interests here are not the relations holding between behaviors and structures but only those between functions, and between goals. The whole-part relation between goals means that one goal is a super ordinate goal for other goals. [Lind, 1994] illustrates it with the following example: the goal *G* that the central heating system operates properly may be decomposed to three goals *G_i*, maintain water level within

safe limits; G_2 , maintain condition for energy transport; and G_3 , keep room temperature within limits. According to Lind G_1 , G_2 and G_3 are parts of G . Analogously the function of transporting energy from radiator to the boiler have three parts: transport of water from supply to expansion tank, circulation of water, transport of energy from boiler to radiator.

In FCO the notion of decomposition of functions is also introduced by means of the part-of relation: “the *part-of* relation between functions represents how a function is achieved by finer grained function [Kitamura et al., 2004], p. 116). Analogously in the OPM function hierarchy, lower level function answers how the superordinate function is achieved.

4.4.2 Function-Part

In OF we distinguish two kinds of part-of relation applied to functions:

1. *Function-part*: Function A is a part of function B if the determinants of A , in particular the requirements and the goal are the parts of the appropriate determinants of B
2. *Sequence-part*: Function A is a part of function B if it is an element of the sequence realizing B .

The function-part is based on the part-of relation that holds between the requirements and the goals of the functions and corresponds to Lind’s whole-part relation between goals. The sequence-part seems to meet the intuitions similar to the whole-part relation between functions. Both kinds of part-of relations are now define formally.

Definition 26 (Function-Part Function). A function f is a *function-part* of a function f' , denoted by $Part_{FU}(f, f')$, iff the requirements of f are the part of the requirements of f' and the goal of f is a part of the goal of f' .

$$Part_{FU}(x, y) \leftrightarrow \exists vw (GoalOf(v, x) \wedge GoalOf(w, y) \wedge Part(v, w)) \wedge \exists st (Req(s, x) \wedge Req(t, y) \wedge Part(s, t)). \quad (37)$$

Consider two functions F_1 : to deliver a car from A to B and F_2 : to deliver an engine from A to B. The former has the requirements: car is in A and the goal: car is in B, whereas the latter has the requirements: engine is in A and the goal: engine is in B. We may say that F_2 is a part of F_1 , since the configurations containing the engine being in A or in B are parts of the configurations of the car being in A or

B, which follows from the fact that an engine is a part of a car. The function-part relation should not be confused with the subsumption of functions based on the subsumption of requirements and goals. For example, the goal G_1 of the function to transport a car is a subgoal of the goal G_3 of the function to transport a vehicle, since car involved in G_1 subsumes vehicle involved in G_3 .⁴²

The particular pattern of function-part is the decomposition of a non-basic goal to the basic ones. Every non-basic goal may be decomposed to two or more basic goals, and every function resulting in any of basic goals is a function-part of the decomposed function. For example, the function F : to produce software satisfying the needs of customer may be decomposed to two basic functions I : to produce software, with the goal: software present, and function H : to satisfy a customer, with a goal: a customer satisfied. The complex goal of function F is then composed of those two chunks which are the goals of functions G and H , hence G and H are the function-parts of F .

4.4.3 Sequence-Part

To illustrate the second type of the part-of relation between functions consider the function F of a purchase system: to purchase an item. The requirements $REQ(F)$ is: exists a customer and an item, and a goal $GOAL(F)$ is: a customer posses an item. In a typical e-commerce system this function may be decomposed into the *sequence* of the following functions:

1. To view/search item
2. To choose item
3. To buy item
4. To deliver item to a customer

It seems intuitive to say that each of the above functions is a part of the function to purchase an item in a sense that they answer how F is achieved. Such a part-of relation we call a sequence-part and introduce it by means of the notion of *sequence*, that is the list l of functions realizing a function f , denoted by $Seq(f, l)$.

⁴² An analogous relationship holds for the requirements of those functions.

Definition 27 (Sequence).

$$\begin{aligned}
Seq(y, L) = L \text{ is a List} \wedge \\
& \exists v(Req(v, Head(L)) \wedge Req(v, y)) \wedge \\
& \text{for every prefix } x \text{ of } L \text{ and } z \text{ s.t. } L = x + z \text{ holds } Enable(Last(x), First(z)) \\
& \wedge \exists w(GoalOf(w, Last(L)) \wedge GoalOf(w, y))^{43}.
\end{aligned} \tag{38}$$

Each element of the sequence of function y we call a *sequence-part* of y and write $Part_{Seq}(x, y)$.

Definition 28 (Sequence Part).

$$Part_{Seq}(x, y) \leftrightarrow \exists L(x \in L \wedge Seq(y, L)). \tag{39}$$

In this sense we say that each of the functions of the e-commerce system is a part of the function `to purchase an item`.

The sequence-part differs from the function-part in a number of properties. Firstly, a function is a sequence-part of another function always with respect to some realization of the second, whereas a function is a function-part of another function independently of the realizations of the second. For example, the function F could be decomposed to some other sequence in a different purchase system.

Secondly, the requirements and the goal of the sequence-part are not necessarily parts of the requirements and the goal of the whole function, whereas the requirements and the goal of the function-part function by definition are parts of the requirements and the goal of the whole function, respectively. In fact, the goal, or the requirements, of any sequence-part function taken in isolation may have nothing to do with the goal or the requirements of the decomposed function. For example the goal of the function `to view item`, is not related to the goal of the function `to purchase an item`.

4.5 Additional Relations between Functions

There may be found a bunch of relations between functions that relates the goal of one function with the requirements of another function. Those relations correspond to meta-

⁴³ Every two successive elements of the sequence are related by the *enablement* relation, which is defined in the next section.

functions introduced in FCO [Kitamura, Mizoguchi, 1999]. A meta-function is introduced as a role of a base function, called an *agent* function, for another base function, called a *target* function. The meta-functions are distinguished from base-functions. The latter are defined as teleological interpretations of behavior, whereas the former are the roles that some base-functions play in the context of other base-functions. Since in OF we do not define functions in the context of behavior, we are not forced to distinguish base-functions from meta-functions. Moreover, we find that not a (agent) function has a role in the context of some other (target) function but to be more precise it is the goal of an agent function having a role in the context of the requirements of the target function.

The role that a goal of one function has in the context of the requirements of the other function we represent by the relations: *support*, *enable*, *prevent*. Moreover, there are cases where the goal of one function influences the other only in a particular realization of the first. For example, the function `to rotate` is improved by the function `to cool` not in general but only in the context of an `engine`. Thus, two realization-dependent relations between functions, *trigger* and *improve*, are introduced later in section 5.7.

4.5.1 Support, Enable and Prevent

We say that the goal of a function f is relevant for a function f' if it influences the requirements of f' . The requirements of f' may be influenced by the goal of f in various ways:

- the goal of f is a part of the requirements of f' , or
- the requirements of f' are a part of a goal of f , or
- the goal of f excludes requirements of f' .

We introduce three relations corresponding to the above three cases, these are: *support*, *enable*, and *prevent*.

Definition 29 (Support). A function f supports a function f' , denoted by $\text{Support}(f, f')$, iff a goal of f is a proper part of the requirements of f' .

$$\text{Support}(x, y) \leftrightarrow \exists v z (\text{GoalOf}(v, x) \wedge \text{Req}(z, y) \wedge \text{PPart}(v, z)). \quad (40)$$

For example, the function `to provide a nail` supports the function `to drive a nail`, since it results in the goal: `a nail is present`, whereas `hammering nails` requires, among the other things, a nail to be present. Here the goal of the former function is a proper part of requirements of the latter.

One function may not only partially satisfy the requirements of another but it may support all the requirements of it. In such a case we say that one function not only supports but enables another.

Definition 30 (Enable). A function f *enables* a function f' , denoted by $Enable(f, f')$, iff the requirements of f' are the part of the goal of f .

$$Enable(x, y) \leftrightarrow \forall v (Req(v, y) \rightarrow \exists z (GoalOf(z, x) \wedge Part(v, z))). \quad (41)$$

The enable relation covers both the case when a goal of f equals the requirements of f' and the case where the requirements of f' constitute a proper part of the goal of f .

In both the support and the enable relations the goal of one function has a *positive* influence on the requirements of the other function. Analogously, a goal may have a negative influence. We say that a goal of the function f has a negative influence on the requirements of function f' when it *excludes* the requirements or part of the requirements of f' . In those cases we say that f prevents f' .

Definition 31 (Prevent). A function f *prevents* a function f' , denoted by $Prevent(f, f')$, iff the goal of f excludes a part of the requirements of f' .

$$Prevent(x, y) \leftrightarrow \exists vwq (GoalOf(v, x) \wedge Req(w, y) \wedge Part(q, w) \wedge Exclude(v, q)). \quad (42)$$

The predicate $Exclude(x, y)$ denotes that a chunk of reality x , here the goal of f , excludes a chunk of reality y , here the requirements of f' , which intuitively means that from the presence of x follows the absence of y .

4.6 Summary

In this section we identified several relations by which functions may be related in a functional model. In the first place we have analyzed the classical ontological relations such as is-a, instantiation and part-of relation. We have found that in context of functions those relations have a particular character and may have different flavors.

The instantiation of functions was based on the distinction of the individual and universal functions and the analysis of instantiation links between the determinants of functions.

The is-a relation between functions is not defined in extensional terms but instead it is based on the structural similarities between functions. Three kinds of is-a are introduced namely the subsumption, its non-reflexive variant – specialization, and individualization.

The part-of relation between functions comes in two flavors – function-part and sequence-part. The first says that one function has a goal and requirements which are parts of the other function’s goal and requirements, whereas the second says that a function is an element of the sequence of functions realizing some other function. The first kind of the functional part-of relation is realization independent, thus in contrast to e.g. OPM function hierarchy it enables the realization-free functional decomposition. The second kind grasps the intuitions behind the realization-dependent function decomposition.

The distinction of various kinds of is-a and the two kinds of part of relation permits to handle properly the functional decomposition, which is an issue of particular importance in functional modeling, since it enables to construct generic and complex functions out of more specific and simple ones. However, in our opinion the functional decomposition is often used informally and is a mixture of several types of relations, e.g. in the goal decomposing pattern of Eriksson and Penker [Eriksson, Penker, 2000] the subsumption and goal-part are implicitly mixed on different levels of decompositions (for details see section 2.2.2). We believe that functional decomposition can be composed of various relations, which however should be explicitly named and formally defined. All of the relations introduced in the current chapter can be used for functional decomposition. Figure 15 presents an exemplary decomposition of the function `to transport goods` based on:

- Subsumption and Specialization (in all their flavors)
- Individualization
- Sequence Part
- Function Part

It can be seen that this approach to functional decomposition not only increases the expressivity of the decomposition link but moreover explicitly distinguishes various principles of decomposition underlying it.

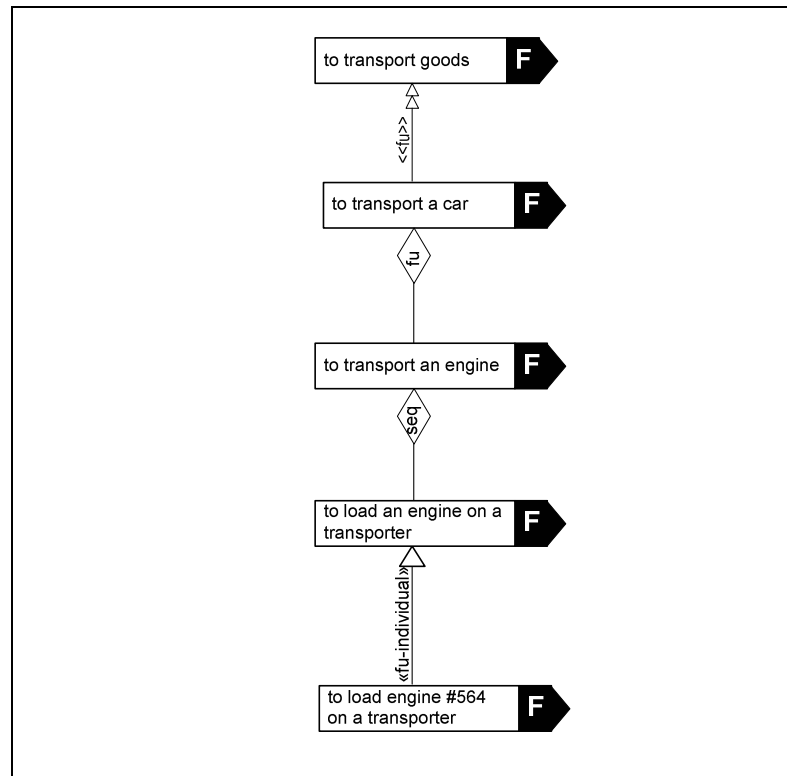


Figure 15. Function decomposition founded on four distinct relations: specialization, function-part, sequence-part and function individualization. The figure uses the UML profile for OF presented in chapter 8. Black-headed arrows represent functions named by labels. The double-headed arrow labeled with <<fu>> represents specialization; the line with a diamond labeled <<fu>> - the function-part; the line with a diamond labeled <<seq>> - the sequence-part; and finally the arrow labeled with <<fu-individual>> - the functional individualization.

Finally, the current chapter has defined three function-specific relations, which permit to model further interdependencies between functions. These are: Support, Enable and Prevent.

5 Realization

5.1 Introduction

Having introduced the common ontological relations of subsumption, instantiation, and the part-of, as well as a number of relations specific to functional modeling, we now turn to the notion of *realization*. In contrast to the relations discussed so far, realization is peculiar since it does not belong purely to the functional model but involves also non-functional entities.

The realization of a function provides the answer to the question *how* the goal of the function is to be achieved. In the literature the notion of realization is often distinguished from the notion of function, which in turn answers the question about *what* is to be achieved. We, however, hold that the notions of functions and realizations are not disjoint because in OF functions are also permitted to realize other functions.

In requirement R.1 for OF we postulated that the specification of an item's function should be separated from its behavior or its structure, which are often considered as the way the function is realized. On the other hand, it is recognized that both notions of function and realization are relative and context-dependent. For example, Salustri [Salustri, 1998] demonstrates that the distinction between function and behavior is contextual. Below we present Salustri's example but, since we do not share his intuitions in defining function, we interpret the example according to the intuitive difference between the function and the realization based on the *what*- and *how*-questions. Salustri considers the following four statements ([Salustri, 1998], p. 340):

- S_1 . The refrigerator keeps food cold.
- S_2 . The refrigerator keeps things cold.
- S_3 . The refrigerator preserves food.
- S_4 . The refrigerator lowers ambient temperature in an enclosed space.

Each of the above statements considered in isolation can be understood as a function of a refrigerator, since each describes *what* a refrigerator does. For example, statement S_1 could be considered as the description of the refrigerator's function to keep food cold⁴⁴. If one asked the question *how* this function was realized, one could find the answer in statement S_4 :

⁴⁴ Req: food present, Goal: food is_cold_during a period; FI: food cooler.

food is kept cold by lowering ambient temperature in an enclosed space. In this sense statement S_1 refers to the function, whereas statement S_4 refers to the realization of this function. This picture, however, gets complicated if we consider statement S_3 . Just as S_1 it can be taken as a description of the refrigerator's function, namely - to preserve food⁴⁵. In this case, when asked how it is achieved that food is preserved, one could use S_1 - food is preserved by keeping it cold. We see that although S_1 refers to a function (has a proper functional structure and has its realization described in S_4) it also refers to the realization of the function described in S_3 .

Two conclusions could be drawn from the above: (1) the realization is not a particular type of non-relational entity but it is a role of an entity in context of some function – the realization is then a binary relation; (2) realizations are not restricted to non-functional entities, but a function may also have a role of being a realization for some other function.

As a key for understanding the realization relation we use the *what* & *how* questions test. For two entities e and e' the test checks whether e (function) provides the answer to the question “*what is e' doing?*”, whereas e' (realization) provides the answer to the question “*how is e achieved?*”. The realization then provides additional information to a function, whereas the function provides the justification of the realization. When applying what & how questions to the pairs of functions related by the functional relations introduced so far the following can be observed:

1. If $x ::_{Fu} y$ then x realizes y .
2. If $x \subseteq_{Fu} y$ or $x \Rightarrow_{Fu} y$ then x realizes y .
3. If $Part_{Fu}(x, y)$ then y realizes x .
4. If $Seq(y, L)$, then L realizes function y , but each $x \in L$ taken in separation does not realize y .

To illustrate the first case let us consider the function to purchase book B by customer C at given time T . This function is an instance of the function to purchase an item. In this sense the former gives a (partial) answer to the question how the latter function is achieved - a purchase of an item is realized by customer C buying book B at time T .

To illustrate the second case consider the function to purchase a book, which answers how the function to purchase an item is realized. Here, the goal of the latter function is a subgoal of the former.

⁴⁵ Req: food edible at t_1 ; Goal: food edible at t_1 ; FI: preserver.

The third case can be illustrated by functions F : to transport an engine and F' : to transport a car. F is a function-part of F' , concurrently F' answers the question how F is realized: An engine is transported via the transportation of a car.

Concerning the fourth case we see that a sequence of functions realizing the function to purchase an item provides a mode of realization of that function and therefore answers the question how this function is realized.

The above cases show the diversity and vagueness of the realization relation, when based only on the intuitive what & how questions test. The picture gets even more complicated when one recognizes that the realization holds not only between functions, but a function may be realized by a *non-functional* entity as well, e.g. by a process, or a structure. In order to provide one general definition of functional realization, beyond the ambiguous distinction based on the what & how questions, and one that would be a common umbrella for the four cases above, we will first define what it means that an individual non-functional entity realizes a function. Secondly, the notion of individual realization will be generalized to the notion of universal realization. Finally, on those basis we are going to define the relation of realization that holds between functions.

5.2 Individual Actual Realization

Intuitively, the *individual realization* of a function f is an individual entity which is the achievement of the goal of f in the circumstances satisfying the requirements of f . Take for example the primitive universal function F : to transport goods G from Leipzig to Berlin in time period T^{46} and the individual process of transportation of goods G by plane from Leipzig to Berlin in T^{47} . In brief, we can say that the process starts with the requirements of F being satisfied and ends with achieving the goal of F and in this sense can be called the realization of function F . Note, that the primitive universal function F could be realized by some other individual process, e.g. by the process of transporting goods G from Leipzig to Berlin by car in T . However, it is not possible that both those processes realize that primitive universal function actually. For

⁴⁶ Figure 16 illustrates some of the notions introduced in this section and their interrelations on the example of the function to transport goods G from Leipzig to Berlin.

⁴⁷ Processes are often labeled by the functions they realize, which is the cause of confusion of both notions. For example both the function and the process can be labeled with the expression transport of goods. However, one should keep in mind that those notions are different. For a detailed discussion see section 7.3.1.

a non-primitive universal function there may be many actual realizations, whereas for a primitive universal function there is at most one individual actual realization, but there may be many possible realizations. We distinguish therefore *actual* realizations from *dispositional* realizations.

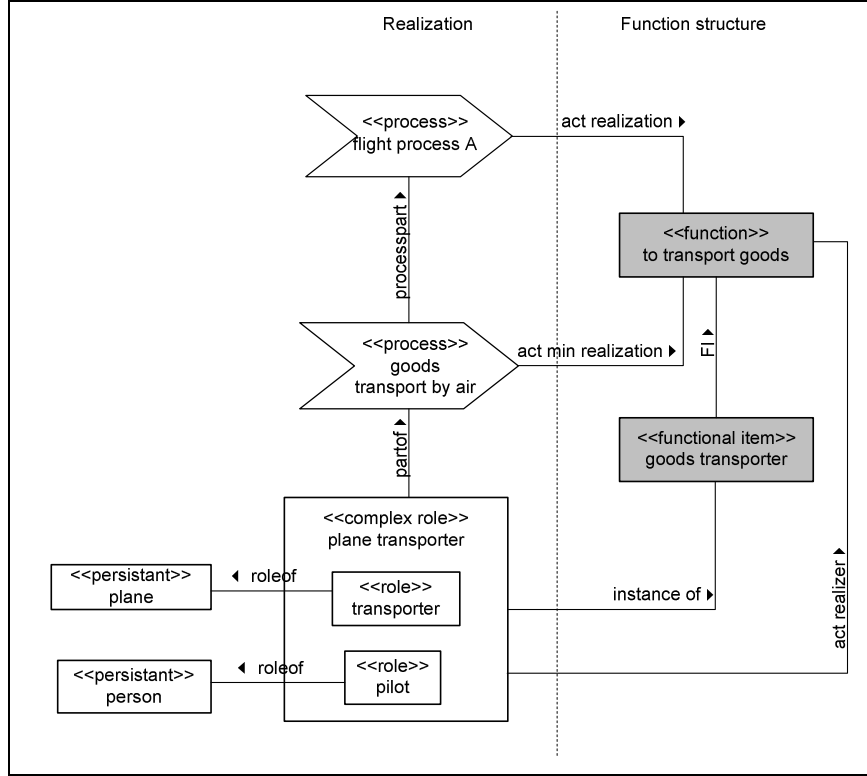


Figure 16. The semi-UML diagram representing entities involved in function realization. The right hand side of the figure presents the function determinants marked gray, the left hand side the entities involved into realization of the function. All classes are labeled with the OF or GFO terms.

Moreover, we distinguish three general kinds of function realization depending on the temporal locations of the requirements and the goal, which we discussed in section 3.5. On the basis of those three function kinds we now introduce three kinds of realization: *processual culminative*, *processual non-culminative* and *situational realization*.

5.2.1 Actual Culminative Realization

The first kind of an actual realization discussed is the culminative realization, corresponding to the sequential function.

Definition 32 (Actual Culminative Realization). An individual process x is called an *actual culminative realization* of a sequential function f , and denoted by $RI_{ActCulm}(x, f)$, iff x is a

process of causal transformation from the presential p fulfilling the requirements of f to the presential p' fulfilling the goal of f .

$$RI_{ActCulm}(x,y) \leftrightarrow Proc(x) \wedge Fu_{Seq}(y) \wedge \forall st(Req(s,y) \wedge GoalOf(t,y) \rightarrow \exists vw(v @ s \wedge w @ t \wedge Trans(x,v,w))). \quad (43)$$

The above definition makes use of the relation of fulfillment, denoted by $x @ y$, and having the intuitive reading that an individual (preferably a complex whole) x fulfills an entity y , in other words - y is fulfilled *at* x . If y is an individual then it is said to be fulfilled in a given complex whole, when it is a part of it. If y is a universal then it is fulfilled in x , when an instance of y is a part of x . For example, the universal goal goods are located in Berlin is fulfilled by every situation, which contains as its part the situation of individual goods being located in Berlin. Formally,

$$x @ y \leftrightarrow Whole(x) \wedge (Ind(y) \vee Uni(y)) \wedge (Ind(y) \rightarrow Part(y,x)) \wedge (Uni(y) \rightarrow \exists z(z :: y \wedge Part(z,x))). \quad (44)$$

The culminative realization is defined in terms of the causal transformation from one presential into the other, denoted by $trans(x,y,z)$. Here x refers to an individual process and y and z to presentials such that x is a *causal transformation* from y to z . We understand causal transformation by means of the notion of *causally cohesive process* developed in [Michalek, 2006]. The causally cohesive process, denoted $Cause_{coh}(x)$ is a process of a particular causal structure, namely every pair of coinciding (inner) time-boundaries contain presentials connected by the basic causal relation. To give only a rough understanding of the causal relation between the presentials, which in fact is out of scope of the current work, we should mention that in [Michalek, 2005; Michalek, 2006] it is understood in terms of regularity and manipulation conditions. The former is considered as a statistical dependency – the presence of the cause raises the probability of the presence of the effect, the latter states that the effect is manipulable by the cause.

By reference to the notion of causal cohesive process we introduce the notion of causal transformation from one presential into the other.

Definition 33 (Causal Transformation). A causal transformation from a presential p to a presential p' is a causally cohesive process, whose projection on its left boundary contains the presential p and the projection on its right boundary - the presential p' . Formally,

$$Trans(x,y,z) \leftrightarrow Cause_{coh}(x) \wedge ProcLBd(y,x) \wedge ProcRBd(z,x)^{48}. \quad (45)$$

From the above we see that not every process that starts with a presential p and ends with a presential p' is a causal transformation from p to p' but only the one which is a causally cohesive process. For example, if we consider the well-known example of the process of the movement of a spot of light on the wall from point A to point B , then we would not consider it as a causal transformation. The pairs of presentials on the coinciding inner boundaries of that process lack the causal connection, i.e. there is no causal link between two locations of the spot of light. The causal connection holds between the position of the source of light and the position of the spot of light, but the former is not a presential being a projection of the process of the movement of the spot of light on its time boundary. Rather it is a projection of the process of the movement of the source of light.

Now, having discussed the notions of the fulfillment and the causal transformation we can illustrate the culminative actual realization by an example. Let us consider the above-mentioned process of flight which could be checked against being the realization of the function of goods transportation. This process has on its left boundary a presential situation of goods being in Leipzig which fulfills the requirements of the function to transport goods from Leipzig to Berlin. On its right boundary the process contains the presential fulfilling the goal of the function, namely the situation that the goods are in Berlin. Moreover, since it is a causally cohesive process, it provides a causal transformation from the state where the goods are in Leipzig to the state that they are in Berlin. In this sense the process of flight is an actual culminative realization of the transportation function.

The ternary relation of transformation can be decomposed to two binary relations: *transformation-from*: $TransFrom(x,y) \leftrightarrow \exists z Trans(x,y,z)$, and *transformation-to*: $TransTo(x,y) \leftrightarrow \exists z Trans(x,z,y)$. From formula 45 and the above splitting of the notion of transformation we obtain the following formulae:

⁴⁸ Process left and right boundaries are presentials located respectively on the left and the right time boundary of the process' framing chronoid. In GFO they are defined formally, $ProcLBd(x,y) \leftrightarrow \exists c (Prt(y,c) \wedge Prb(y,L(c),x))$ (left process boundary), $ProcRBd(x,y) \leftrightarrow \exists c (Prt(y,c) \wedge Prb(y,R(c),x))$ (right process boundary).

$$RI_{ActCulm}(x,y) \rightarrow \forall z(Req(z,y) \rightarrow \exists v(v @ z \wedge TransFrom(x,v))). \quad (46)$$

$$RI_{ActCulm}(x,y) \rightarrow \forall z(GoalOf(z,y) \rightarrow \exists v(v @ z \wedge TransTo(x,v))). \quad (47)$$

In accordance with formula 46 the fulfilled requirements are the necessary condition for an actual culminative realization to occur; and in accordance with formula 47 an actual culminative realization is a sufficient condition for the goal to be fulfilled. For example, the goods must be available in Leipzig in order to enable the process of transporting them from Leipzig. If the process of transporting goods takes place, then at the end of the process the goods are in Berlin. Note that an actual realization assumes a positive result. It cannot happen that the process of transporting goods occurs but the goods are not transported as a result of the process. If they were not transported, the process would not be called an actual realization.

5.2.2 Actual Non-culminative Realization

The second kind of realization corresponds to the continuous function and is defined as follows.

Definition 34 (Actual Non-culminative Realization). An individual x is called an *actual non-culminative realization* of a continuous function f , and is denoted by $RI_{ActNonCulm}(x, f)$, iff x is a process containing as its layers a process p fulfilling the requirements of f and a process p' fulfilling the goal of f and all processes p'' which adhesively cause p' .

$$\begin{aligned} RI_{ActNonCulm}(x,y) \leftrightarrow & Proc(x) \wedge Fu_{Contin}(y) \wedge \\ & \forall st(Req(s,y) \wedge GoalOf(t,y) \rightarrow \\ & \exists vw (LayerPart(v,x) \wedge LayerPart(w,x) \wedge v @ s \wedge w @ t \wedge \\ & \forall q (Cause_{adh}(q,w) \rightarrow LayerPart(q,x)))) \end{aligned} \quad (48)$$

The notion of non-culminative realization is based on the notions of the process layer part and the causally adhesive processes [Michalek, 2006]. The process layer part denoted by $LayerPart(x,y)$ is a process x being a processual part of y , which is framed by the same chronoid as y . The causally adhesive processes, denoted by $Cause_{adh}(x,y)$ are the processes x and y such that at every pair of coinciding time-boundaries t and t' , such that t is the boundary of x and t' is a boundary of y , there exist causally connected presentials p and p' such that p is the projection of the process x and p' is the projection of process y .

Let us explain the definition of actual non-culminative realization by the example of a continuous function `to pump blood`. Intuitively a situoid comprising a heart, its behavior and the blood it pumps could be called a realization of this function. In this situoid we find a layer fulfilling the requirement of the function (`blood is present`) and the one fulfilling the goal (`blood is being pumped`). Moreover, this situoid has as its layer part a process of behavior of the heart, which is an adhesive cause for the process of blood being pumped.

Since the goal of the function is not a culmination of that situoid but rather it is maintained during the whole situoid, this kind of realization in contrast to culminative, is called a non-culminative.

5.2.3 Actual Situational Realization

The third actual realization kind, corresponding to the instantaneous function, is called a *situational* realization and is defined as follows.

Definition 35 (Actual Situational Realization). An individual x is called an *actual situational realization* of an instantaneous function f , and is denoted by $RI_{ActSit}(x, f)$, iff x is a situation fulfilling the requirements and the goal of the function f and containing as its part a causal factor for the goal being fulfilled.

$$\begin{aligned}
 RI_{ActSit}(x, y) \leftrightarrow & \text{Sit}(x) \wedge Fu_{Instant}(y) \wedge \\
 & \forall st(\text{Req}(s, y) \wedge \text{GoalOf}(t, y) \rightarrow \\
 & \exists vw(\text{CPart}(v, x) \wedge \text{CPart}(w, x) \wedge v @ s \wedge w @ t \wedge \\
 & \forall q(\text{Cause}_{Inst}(q, w) \rightarrow \text{CPart}(q, x))).
 \end{aligned} \tag{49}$$

In contrast to the two previous kinds of realization the situational realization is not considered as a time-extended entity but is a presential situation. This grasps the intuitions of instantaneous function realization typical of functions realized by structures rather than behaviors. In order to illustrate that definition let us consider the example of the realization of the instantaneous function `to camouflage a moth in environment`. Since the function is instantaneous it is required that whenever a moth is present in the environment, then instantaneously, and not by means of some process extended in time, it is safe from predators in that environment.

Now, if we consider the situation of an individual pepper moth sitting on dark bark, we see that due to its dark color it is safe from predators in that environment. Thus, we may say that this situation is the realization of the function. It not only fulfills the function's

requirements (moth is present in environment) and the goal (moth is safe from predators) but moreover it contains as its part a causal factor for the goal being fulfilled, namely the dark color of the moth.

The causal factor denoted by $Cause_{Inst}(x,y)$ is the relation between two presentials x and y , namely between the presential fact that the moth has dark color and the presential fact that the moth is safe from predators. In contrast to the causal connection between presentials, which the causal cohesion and the causal adhesion refer to, here the presentials are considered not at two coinciding boundaries but at the *same* time boundary, as both the causal factor and its effect, are parts of the same situation. This understanding of the causal factor neither disturbs the regularity nor the manipulation condition of causal relation. For instance, there is a statistical dependency between the color of the moth and its safety. Analogously, manipulation of the moth's color influences its safety.

We see therefore that such an understanding of causality⁴⁹ violates only the temporal order of the causal relation and could thus be considered as the less restrictive form of it. On the other hand, it seems to be in agreement with the common usage of the notion of cause, which also refers to such instantaneous cases.

Some of the non-culminative functions can be reduced to the number of situational functions. In particular those which do not involve the active behavior but are founded directly on the structure. For instance, the function to support the roof for a given period realized by the column of some house can be understood in terms of the number of the (presential) situations in which the presential columns support the roof⁵⁰. In contrast the process of the heart behavior cannot be reduced to the number of situational realizations, but should be considered on the processual level.

The three technical definitions above can be underpinned by the general notion of the actual realization, $RI_{Act}(x,y)$.

$$RI_{ActNonCulm}(x,y) \vee RI_{ActCulm}(x,y) \vee RI_{ActSit}(x,y) \rightarrow RI_{Act}(x,y). \quad (50)$$

Intuitively, the actual realization is the individual entity which fulfills the requirements and the goal of the function and provides an additional cause for the goal being fulfilled.

⁴⁹ $Cause_{Inst}(x,y) \leftrightarrow Pres(x) \wedge Pres(y) \wedge Reg(x,y) \wedge \exists q,w(Man(x,q,y,w)) \wedge \exists t(At(x,t) \wedge At(y,t))$, where the predicates *Reg* and *Man* refer to the regularity and the manipulation conditions. For details see appendix A.

⁵⁰ For details see the distinction between active and passive realizers in section 5.6.

5.3 Minimal Actual Realization and its Components

An actual realization can be decomposed to the *minimal actual realization* which has a particular structure, namely it contains entities *contributing* to achievement of the goal and those *executing* the realization. The former we call the *means of realization* and the latter the *realizers*.

5.3.1 Minimal Actual Realization

Often an individual realization is a complex entity. In one of the above examples it is the flight of a plane, which is a complex process composed of a particular plane, together with all its properties, like shape, color, and cargo space; with the crew, eventual passengers, the airports, the route of the flight, and many others entities. Out of them we can distinguish those, contributing to the realization of the function and those, having no influence on it. For example, the persistants plane and crew, and the process of weather contribute to the transportation of goods, whereas passengers do not. Thus, we can introduce the ternary relation $Contribute(x,y,z)$, with the meaning that x contributes to the realization y of function z . The contribution to the realization should be understood in terms of the causal impact, e.g. the plane contributes to the realization of the function of transporting goods, since it has a causal impact on the situation of goods being located in the destination. Analogously, in the situation realizing the function of camouflage, the dark covering of the pepper moth contributes to this realization, since it is the causal factor for the moth being camouflaged.

On the basis of the notions of the individual actual realization and the contribution relation we may introduce the *minimal actual realization*.

Definition 36 (Minimal Actual Realization). A *minimal actual realization* r of a function f , denoted by $RI_{ActMin}(r,f)$, is such a realization of f that every entity involved in r contributes to the realization of the function f .

$$RI_{ActMin}(x,y) \leftrightarrow RI_{Act}(x,y) \wedge \forall z (Part(z,x) \rightarrow Contribute(z,x,y)). \quad (51)$$

For every actual realization there may be found a part of it, which contains only entities contributing to this realization.

$$RI_{Act}(x,y) \rightarrow \exists z (RI_{ActMin}(z,y) \wedge Part(z,x))^{51}. \quad (52)$$

For the process of a flight of a plane a minimal realization would be a layer of that process comprising only those entities which contribute to the realization of the function of transportation, i.e. a plane, a crew, goods, weather, whereas some other elements of a flight would be left apart e.g. passengers. In this example the minimal realization is a process, which we could call the process of transporting goods by plane, and which is a layer contained in the process of flight of the plane. Note that just as in the case of the individual actual realization there may be more than one minimal actual realization of a non-primitive function. For example, the process of transporting goods by a car is a different minimal actual realization of the function of transporting goods.

5.3.2 Means of Realization

In the process of minimal realization are involved only those entities which contribute to the realization of the function. Each of those entities is related to the minimal realization by a contribution relation and therefore plays some role in that realization. The role of each entity contributing to the realization of the function we call a *contributor*. Thus, we may say that the plane has a role of contributor in the realization of the function of transportation.

As a role of an entity in the contribution relation we understand such an aspect of that entity, by which the entity contributes to the realization, i.e. a contributor is a structural role including structural aspects of a role-filler. For example, the contributor role of a plane in the realization of the transportation function comprises such structural aspects of a plane as the maximum flight distance or the cargo space, whereas it abstracts from such aspects as the color of the plane, the number of passenger seats, etc. Often a contributor-role has its own name, in our example a plane's contributor-role is *transporter*. All contributor-roles may be combined into one entity, called *actual means of realization*:

Definition 37 (Actual Means of Realization). A composition x of contributor-roles of all the entities contributing to the actual realization r of a function y is called the *actual means of realization* of y , and is denoted by $Means_{ActRI}(x,y)$.

⁵¹ This formula trivially follows from the definitions of realization and the reflexive understanding of the notion of part.

In other words, the means of realization is a relational entity composed of all aspects of entities involved in the realization of the function that are relevant for that realization. The means of realization typically form a complex role composed of several contributor-roles, optionally interrelated. In our example, the means of realization is the composition of contributor-roles of the plane, the crew, the goods, the weather conditions, and the airports in the process of transporting goods.

This example illustrates also the other feature of means of realization, namely that its structure is not homogenous. Firstly, different types of entities may be role-fillers of particular role-contributors. For instance, we may consider a plane, a person, and goods as persistants, whereas the weather as a process. Secondly, the particular components of the means of realization may be interrelated by various relations. For example, a *pilot*, which is a role of a person, may be related to a *plane-transporter* role of a plane by the relation of *piloting*: a *pilot* *pilots* a *plane transporter*, whereas *transportee* (goods *qua* *transportee*) is related by the relation of *transported on to* *plane transporter*.

5.3.3 Actual Realizer

Among the entities contributing to the minimal actual realization some have a primary status in that realization, namely they are said to *execute* the realization of a function. For instance, to the realization of the function *to pump blood*, contribute *heart*, *blood*, and *veins*. However, the role of the heart is different from veins and blood, namely it is the heart which pumps blood, and thus executes this realization. The relation of execution is thus a sub-relation of the contribution relation: $Execute(x,y,z) \rightarrow Contribute(x,y,z)$. The role of entities executing a realization in comparison to mere contributor-roles we would call an *executor-role* or an *actual realizer*.

Definition 38 (Actual Realizer). An individual executing a realization r of a function f plays in that realization a role x called an *actual realizer* of function f and denoted by $R_{Ac}(x,f)$.

$$R_{Ac}(x,y) \leftrightarrow \exists uv(RI_{Ac}(u,y) \wedge Execute(v,u,y) \wedge HasRole(v,x) \wedge RoleIn(x,u)). \quad (53)$$

For every minimal actual realization there is an entity executing it, and thus there is an actual realizer which is a role of that entity in this realization.

$$Rl_{MinAct}(x,y) \rightarrow \exists z (RoleIn(z,x) \wedge R_{Act}(x,y)). \quad (54)$$

The realizer might be a complex role composed of roles of several entities. For example, in the case of the transportation function we could say that a pilot piloting a plane realizes a function of transporting goods. Here, the realizer role of the function would be a complex role composed of a pilot role and a plane role. A complex actual realizer of a function f we denote by $R_{ComplAct}(x,f)$ and define as follows:

Definition 39 (Complex Actual Realizer).

$$R_{ComplAct}(x,y) \rightarrow \exists vw (R_{Act}(v,y) \wedge R_{Act}(w,y) \wedge w \neq v \wedge \\ RolePPart(w,x) \wedge RolePPart(v,x)). \quad (55)$$

A realizer can be identified also by means of a functional item. A functional item is depicting in purely teleological terms the role of an entity executing the realization of a function. A realizer being an individual role of an entity(-ies) executing the realization of a function is an either an instance of the functional item or in case of the individual functional item it contains it as a part. In this sense we can say that a realizer *fulfills* a functional item. For instance, a pure teleological role transporter is the functional item of the function to transport goods saying that the role filler executes the transportation of goods. Among all contributors of the realization of this function realizers are those which are instances of the functional item, i.e. the complex role of a pilot and a plane. Every actual realizer of a function f fulfilling the functional item of f must satisfy the requirements of the functional item, $Req_F(x,f)$ defined in the requirements of the function f .

Note that the same configuration of entities may form the means of realizations for more than one function. For example, the above-discussed means of realization of the function to transport goods are also the means of realization of the function to be transported. What distinguishes realizations of those two functions is that for the first function the role of pilot piloting a plane forms a realizer, whereas for the second the realizer is the transportee role of the goods.

In contrast to the functional item the realizer is not considered only in teleological terms but instead it comes with a rich structural description exceeding that imposed by the goal. For example, in case of the realizer a pilot piloting a plane each of roles composing it is structurally described. For example, plane-transporter has such properties as cargo space or maximum flight distance, whereas the pilot role is depicted by the required skills.

5.3.4 Dynamic and Passive Realizer

If we analyze the dynamics of the realizer we find that it may be either *dynamic* or *passive*.

Definition 40 (Dynamic Realizer). A realizer r is called a *dynamic* realizer of a function f , denoted by $R_{Dyn}(r, f)$, iff r is a process with some dynamics involved in it, i.e. if it changes over time. Formally,

$$R_{Dyn}(x, y) \leftrightarrow Proc(x) \wedge R(x, y) \wedge \exists e_1 e_2 b_1 b_2 (Prb(x, b_1, e_1) \wedge Prb(x, b_2, e_2) \wedge Change(e_1, e_2)). \quad (56)$$

The predicate $R(x, y)$ is the generalized notion of realizer comprising both the actual realizer and the dispositional realizer introduced below; $Change(e_1, e_2)$ is an abbreviation of GFO predicate⁵² having the intuitive meaning that there is a change between two presentials e_1 and e_2 ; $Prb(x, b, e)$ denotes the relation of projection between the process x , the time boundary b of the chronoid framing this process and the process boundary e of x located at b . In GFO process boundaries are presentials located at time boundaries of chronoids framing a given process. In this sense the processual realizer whose two boundaries form a change is called dynamic. For example, *chameleon camouflage*, considered as a processual role of chameleon's covering in the process of camouflaging, which dynamically adapts to the environment is a dynamic realizer of the function *to camouflage*, since it is a process which has two boundaries such that there is a change of the color of the covering between them.

The dynamics of a realizer may be classified on the basis of the classification of changes developed within GFO, where two kinds of changes are distinguished – *extrinsic* and *intrinsic*. Extrinsic changes are discontinuous, instantaneous changes, involving two coincident process boundaries, which are instances of two disjoint universals; whereas intrinsic changes are continuous, for example locomotions. Depending on the kind of change a dynamic realizer undergoes during the realization of the function it can be classified as *extrinsically dynamic*, *intrinsically dynamic*, or as a mixture of both kinds. For example, a chameleon's covering is an extrinsically dynamic realizer, since the change of the color in the realization is extrinsic.

Beside active also *passive* realizers are introduced to OF:

⁵² Full version of the GFO predicate is introduced in appendix A.

Definition 41 (Passive Realizer). A realizer r of a function f is called *passive*, and denoted by $R_{Pass}(x, y)$, iff the realization does not involve any changes of the realizer. Formally,

$$R_{Pass}(x, y) \leftrightarrow R(x, y) \wedge \neg R_{Dyn}(x, y). \quad (57)$$

For example, the covering of a frog is the passive realizer of the function `to camouflage`, since, in contrast to the chameleon's covering, it does not undergo any change (extrinsic or intrinsic) during the realization of the function.

Note that from formulae 49, 56 and 57 follows that all realizers of situational realizations are passive since they are not processes but presential situations.

5.4 Universal Realization and Realizer

A minimal actual realization and actual realizer are individuals and as such they have corresponding universals, which we call *universal minimal realization* and *universal realizer*, respectively.

5.4.1 Universal Minimal Realization

A minimal actual realization is an individual, and thus there may be found a *structural* universal which it instantiates directly⁵³. Such a direct universal we call a *universal minimal realization*.

Definition 42 (Universal Minimal Realization). A structural universal u is called a *universal minimal realization* of a function f , denoted by $UniRl_{Min}(u, f)$, iff u is directly instantiated by a minimal actual realization of f .

$$UniRl_{Min}(x, y) \leftrightarrow \exists z (Rl_{MinAct}(z, y) \wedge z \vdash x). \quad (58)$$

⁵³ We are interested only in such individuals about which something may be predicated, and since universals are considered to be entities that may be predicated of other entities and are expressed and represented in terms of language, thus for every individual we speak about, we may point to its corresponding universal.

The universal minimal realization is composed of universals directly instantiated by the parts of the individual actual realization:

$$UniRl_{Min}(x,y) \rightarrow \forall z (CatPart(z,x) \rightarrow \exists wv (w ::: z \wedge Part(w,v) \wedge Rl_{MinAct}(v,y))). \quad (59)$$

The direct instantiation, denoted by the predicate $x ::: y$ means that x is an instance of the universal y and there is no such universal z which is a sub-universal of y such that x is its instance. Formally,

$$x ::: y \leftrightarrow x :: y \wedge \neg \exists z (x :: z \wedge Subsume(y,z)). \quad (60)$$

In the case of the individual process of transporting goods by plane P a universal realization would be the universal of the process of transporting goods by plane, containing the universals corresponding to all individuals being parts of the minimal actual realization. For each component of the actual minimal realization one should try to find a corresponding universal that is instantiated directly by it. Only the directly instantiated universals are taken into considerations, since we are interested in most specific universals. This, however, shows the dependency of the notion of the universal minimal realization on the granularity of a given taxonomy of universals. For example, the universal realization corresponding to the individual process of transporting goods by plane is the universal process of transporting goods by plane, containing a plane universal rather than the process of transporting goods by a physical object containing a physical object universal, since the second is not instantiated directly by the individual plane involved in the individual process. This, however, is relative to the granularity assumed. If one refers to the most top level taxonomy of universals, which lacks the category of plane but contains only the category of physical object then the corresponding universal minimal realization would be the process of transportation by a physical object.

A universal realization is a structural universal, which means that only structural aspects of the actual realization are taken into account when constructing it. A universal realization does not reflect the non-structural aspects of a process of transportations, such that it is a causal transformation to the state where goods are transported.

The notion of the universal realization corresponds to the intuitions behind the notion of *the-way-of-achievement* introduced and delimited from functions in FCO [Kitamura et al., 2002]. The-way-of-achievement relates the function to “the principle of the achievement” ([Kitamura et al., 2002], p. 150). Similarly, a universal realization represents the principle

underlying the individual realizations instantiated by it. In this sense the universal realization is a general specification of the way the function is realized by some individual realizations.

5.4.2 Universal Realizer

In an analogous way to the universal realization the *universal realizer* is constructed.

Definition 43 (Universal Realizer). A structural universal u is called a universal realizer of a function f , denoted by $UniR(u, f)$ iff u is directly instantiated by the individual actual realizer of f .

The structural universal is a universal depicting only structural features of corresponding individuals. In this sense the structural universal directly instantiating a plane-transporter role is the universal depicting a plane only in the context of the structural features relevant for the realization of the transportation function such as the cargo space and the maximum flight distance. It does not include the characteristic saying that a plane is involved in the process of flight since this characteristic is not structural.

Neither should a universal realizer be identified with the universal functional item of the realized function nor put in the intensional subsumption relation to it. A universal functional item is characterizing corresponding individuals only in teleological terms and abstracts from the structural features not imposed by the goal, whereas a universal realizer is a structural universal and it lacks functional characteristics typical of a functional item.

5.5 Dispositional Realizer and Realization

We can describe entities not only due to their actual state of being a realization of a function or executing such a realization, but also due to their capability of doing that. For that purpose we introduce the notions of *dispositional realizer* and *dispositional realization*.

5.5.1 Dispositional Realizer

If we consider some arbitrary plane, it is intuitive to judge whether or not it could be used as a realizer of the function of transportation. As long as it is not involved in the process of goods transport it is not an actual realizer; however, we would often say that it has a disposition to do so. In this sense we can speak about a *dispositional realizer of a function*.

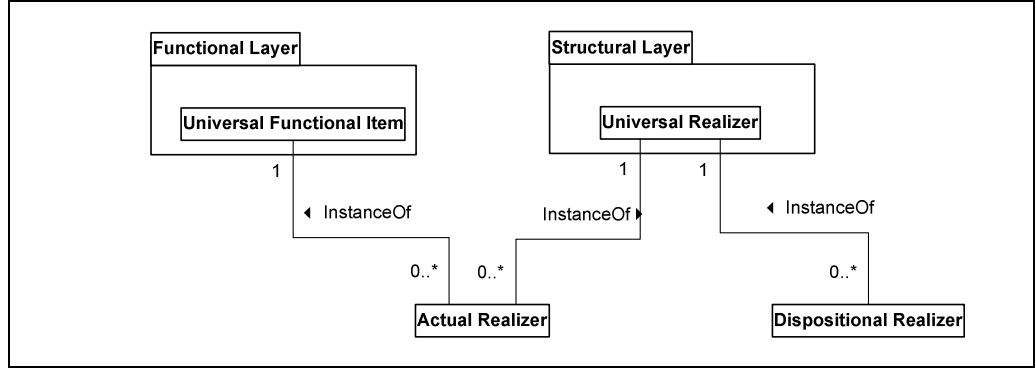


Figure 17. The UML diagram representing relations between Functional Item and Realizers. Classes represent reified universal roles (Universal Functional Item and Universal Realizer) and reified individual roles (Actual Realizer and Dispositional Realizer). Universals are grouped in two disjoint Layers: Functional Layer and Structural (non-functional) Layer. Universal Functional Item is a universal role depicted in purely functional (teleological) terms, whereas Universal Realizer is a universal role depicted in purely structural terms. Actual Realizer comprises both structural and functional aspects, whereas Dispositional Realizer only structural aspects.

Commonly dispositional realizers are identified by means of analogy. If the actual realizer of a given function is known, we consider the entities similar to it as potential realizers. In OF an individual dispositional realizer is an entity relevantly similar to the actual realizer, where the relevant similarity means that the individual dispositional realizer is similar to an actual realizer in all aspects relevant to the realization of a function. In this sense it is an instance of the same universal realizer. We say that every instance of the universal realizer which is not an individual actual realizer is called an individual dispositional realizer. An individual dispositional realizer r of a function f is denoted by $R_{Disp}(r, f)$, and is defined formally as follows:

Definition 44 (Dispositional Realizer).

$$R_{Disp}(x, y) \leftrightarrow \exists z (UniR(z, y) \wedge x :: z \wedge \neg R_{Act}(x, y)). \quad (61)$$

From the above we can see that each individual actual realizer is an instance of both a universal functional item, which captures its teleological character, and a universal realizer, which captures its structural features. In contrast, an individual dispositional realizer is an instance of a universal realizer only. Since it does not realize the function actually, it lacks the teleological characteristic and hence is not an instance of a functional item (see figure 17).

5.5.2 Dispositional Realization

Often not only persistants are judged against their disposition to execute the realization of a function but also an entity, say a process, may also be judged against its ability to be a realization of a function. For example, if we consider an arbitrary flight of the plane, it is intuitive to judge whether it could be used as a realization of the function of transportation or not. In this sense the process of flight is considered as a potential realization of the function, but not as an actual realization, since it does not actually realize a function but only could do so. So if we consider an individual process of flight, by which goods are not transported, call it x , then it is clear that formula 47 does not hold for x . Since x is not an actual realization of the function but only a *dispositional* one, thus it is not the case that when x occurs the goal of the function is achieved, but rather it is only *possible* that the goal is achieved. For a dispositional realization x of function y , denoted by $RI_{Disp}(x,y)$, formula 47 can be replaced with the following formula of modal logic: $RI_{Disp}(x,y) \rightarrow \forall z(GoalOf(z,y) \rightarrow \Diamond \exists v(v @ z \wedge TransTo(x,v)))$.

Unfortunately, the above formulation gives no hints about the structural conditions a flight should fulfill in order to be a dispositional realization of the function of transporting. That formula gives no help in the determination of the structural properties a flight should have in order to realize the function, which in fact is the crucial issue when looking for possible function realizations. One is not interested only in knowing that a dispositional realization is such that may turn out to be an actual realization. Rather one wants to know what features make it a possible realization. In practice, analogously to the case of dispositional realizers the answer to this question is given by analogy. One looks into the current realizations and searches for entities that are similar to them. However, not an absolute similarity to the actual realizations is needed, but only similarity in those aspects which are relevant for the realization. Helpful in the determination of those aspects is the above-introduced notion of the universal minimal realization. We say that every instance of the universal minimal realization of a function f which is not an actual realization of f is an *individual dispositional realization* of f .

Definition 45 (Dispositional Realization).

$$RI_{Disp}(x,y) \leftrightarrow \neg RI_{Act}(x,y) \wedge \exists s(UniRI_{Min}(s,y) \wedge x :: s). \quad (62)$$

A universal minimal realization is thus considered as a template of the realization – it states all the structural features, like plane cargo space, maximum flight distance,

duration of flight, flight route, skills of the crew, that must be fulfilled by an individual realization. Each individual entity which fulfills those conditions and thus is an instance of a universal minimal realization, but is not actually realizing a function is a dispositional realization⁵⁴. Since usually there are many ways of realizing a function, every universal minimal realization depicts a particular way of realization, e.g. transport by car, or transport by air (see figure 18).

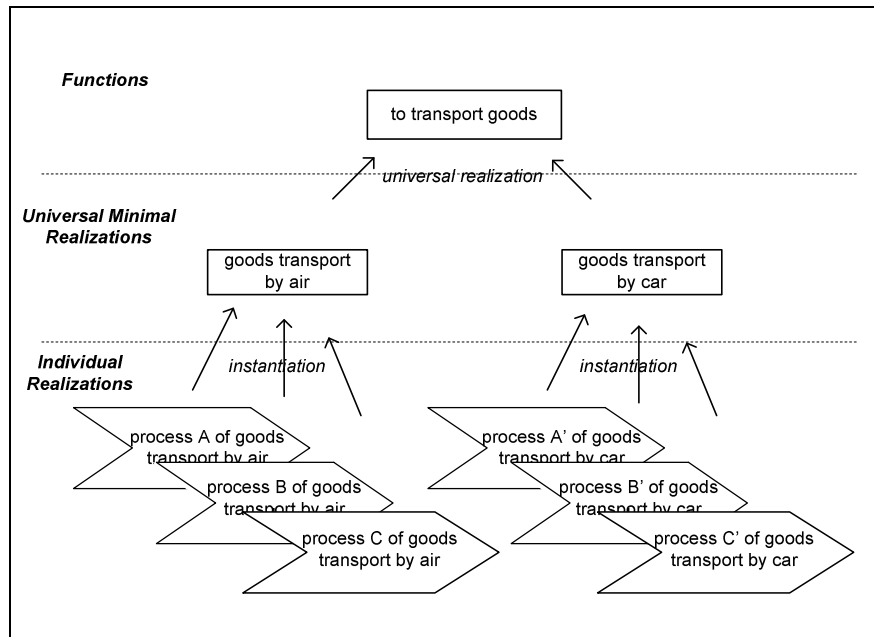


Figure 18. Universal minimal realization as an template for individual realizations. Every individual realization is an instance of some universal minimal realization. Analogously, universal realizers are templates for individual realizers.

5.5.3 Strong Dispositional Realizers

Now that the notions of dispositional realization and dispositional realizer are introduced we may see that some of dispositional realizers are involved in dispositional realizations. For example consider two planes⁵⁵, which are dispositional realizers of the function to

⁵⁴ In the current approach dispositional and actual realizations (and analogously realizers) are considered to be mutually exclusive. However, this we find rather as a matter of convention and we are aware that for the purpose of some applications it may be beneficial to consider actual realizations (and realizers) as a subclass of dispositional ones. This can be easily achieved by slight modifications to axioms 62 and 63.

⁵⁵ To be more precise we do not speak about planes as wholes here but only on their aspects relevant for the realization of the function, thus plane-transporters.

transport goods from Leipzig to Berlin. One plane is on the flight to Berlin and the second is locked in a hangar in Leipzig. In the first case not only the plane is a dispositional realizer, but moreover the flight in which it participates is a dispositional realization of the function to transport goods from Leipzig to Berlin. In the second case the plane is a dispositional realizer but it is not involved in the process of the dispositional realization. Clearly, the former plane is considered to be a better candidate for the function realization than the latter plane, and is called a *strong* dispositional realizer.

Definition 46 (Strong Dispositional Realizer). A dispositional realizer x of a function f involved in the dispositional realization of f is called a *strong dispositional realizer* of f and is denoted by $R_{DispStr}(x, f)$.

$$R_{DispStr}(x, y) \leftrightarrow R_{Disp}(x, y) \wedge \exists z (Part(x, z) \wedge RI_{Disp}(z, y)) \quad (63)$$

5.6 Functional Realization

In the previous section, on the basis of the how & what question test, we identified several cases, in which one function realizes another. Those cases are problematic, since they significantly differ one from another and no common definition for the realization that joins all of them was found.

Our strategy for introducing the notion of realization that holds between functions is to reduce it to the above-defined notion of individual realization that holds between a function and a non-functional entity.

Definition 47 (Functional Realization). A function f *realizes* a function f' , denoted by $Realize(f, f')$, iff all individual realizations of f are the individual realizations of f' and not all of the realizations of f' are the realizations of f . Formally,

$$Realize(x, y) \leftrightarrow \forall w (RI_{Act}(w, x) \rightarrow RI_{Act}(w, y)) \wedge \exists v (RI_{Act}(v, y) \wedge \neg RI_{Act}(v, x)). \quad (64)$$

The second part of the definition blocks the trivial case where the function realizes itself, as it does not answer *how* the function is realized which should be answered by a realization.

Let us now examine the cases of functional realization reported in section 5.1 against that definition:

1. *An instance function realizes the universal function it instantiates.* By definition 31, for $f ::_{Fu} f'$ every individual determinant of a function f instantiates a corresponding universal determinant of f' . Thus, the realization of f comprises instances of determinants of f' and in this sense is the realization of f' as well. On the other hand an arbitrary realization of a universal function is not necessary a realization of the instance function.
2. *A specialized function realizes a general function.* Since every determinant of a specialized function is subsumed by the appropriate determinant of the generic function then it can be clearly seen that every entity being a realization of a specific function is also a realization of a generic function. Analogously to the point above, not every realization of a function is a realization of its subfunctions.
3. *A whole function realizes its function-part.* Every realization of a given function is also a realization of its function-parts, but not every realization of its function-part entails its realization.
4. *A sequence of functions realizes the whole-function; every sequence-part function in separation does not realize the whole-function.* Each realization of the whole sequence is an individual realization of the function. For instance, each realization of the sequence `search item, view item, order item, deliver item` is a realization of the function of `purchase item`. Note that not every individual realization of the function is a realization of the sequence. It is possible that a purchase is realized by a different sequence of functions. This distinguishes the sequence-part from the function-part, for which the realization of the whole function is also the realization of its parts.

From the above we see that every case of one function realizing another function reported in section 5.1 can be reconstructed in terms of an individual realization.

5.7 Realization-dependent Relations between Functions

In section 4.5 there have been introduced three kinds of relations between functions based on the role of the goal of one function in the requirements of the other, i.e. support, enable, and prevent. It is often the case that the relations between functions are not modeled independently of the realizations of functions but are intrinsic to them. One such relation introduced already is sequence-part. Now, we consider two more, namely the *trigger* and the *improve* relation.

5.7.1 Trigger

A goal of one function might be relevant for the realization of another function not only when it has an influence on its requirements, like in the case of the support, the enable and the prevent relations but also when it is a trigger of the other function realization.

Definition 48 (Trigger). A function f *triggers* a function f' in a realization r , denoted by $Trigger(f, f', r)$, iff the goal of the function f is a trigger of the realization r of the function f' .

$$Trigger(x, y, z) \leftrightarrow \exists v (GoalOf(v, x) \wedge Trig(v, z) \wedge RI_{Act}(z, y)). \quad (65)$$

A trigger together with the requirements provides the necessary and sufficient condition for the function realization. In this sense a trigger can be understood as a direct cause of the function realization. In contrast to the requirements which are common for all realizations of a given function, different triggers may be assigned to different realizations of the same function. For example the realizations of the transportation function can be triggered by such triggers as `phone order` or `invoice signed`. Therefore the trigger relation between functions is realization-dependent - one function has an influence on another only relatively to its particular realization.

5.7.2 Improve

Two kinds of side effects are considered in OF. Beside the function side effects discussed in section 3.7, which belong to the very nature of the function's goal, we now introduce side effects of the function realization. A realization side effect is not a result of a function as such but it is a result of its particular realization. Having introduced the notions of realization, fulfillment and transition we define realization side effect as follows:

Definition 49 (Realization Side Effect). An individual x is called a *side effect of the realization r* of a function f and denoted by $SideEf_R(x, r, f)$ iff x is a part of r 's result and no effect of f is fulfilled by x .

$$SideEf_R(x, y, z) \leftrightarrow RI_{Act}(y, z) \wedge Cause(y, x) \wedge \forall v (GoalOf(v, y) \vee SideEf(v, y) \rightarrow \neg x @ v). \quad (66)$$

The predicate $Cause(x,y)$ is a common umbrella for three causal relations introduced in section 5.2 and has the reading “ x causes y ”⁵⁶. As an illustration of the above definition consider the process of transporting people by car. This process is the realization of the function `to transport people` and results among others in polluting the environment. However, the fact of the environment’s pollution, considered as a situation, does not fulfill a goal or any side effect of that function. Therefore, the environment pollution is considered to be a side effect of that particular realization of the function of transportation. Relying on the notion of the realization side effect we introduce the *improvement* relation between functions.

Definition 50 (Improve). A function f *improves* a given realization r of a function f' , denoted by $Improve(f,f',r)$, iff a realization of f neutralizes some side effect of the realization r of f' .

For example, the car transportation process which realizes the transportation function and pollutes the environment might be improved by the function of `reducing pollutant emission` which has the goal `pollutant emission reduced`.

Both trigger and improve are ternary relations, involving as a third argument the realization of improved/triggered function, which reflects the realization-focused character of those relations.

5.8 Summary

In this chapter we have investigated the problem of function realization, which can be stated as the search for the answer for the question “what does it mean that an entity realizes a function?”.

Two basic senses of function realization have been found. In the first sense an individual entity, e.g. a process, is said to be a realization of a function. In the second sense an entity, e.g. a persistent via its execution of the process of realization is said to realize the function. The former have been called a realization of a function, the latter - a realizer of a function.

We believe that realizations should not be considered as functions, and in particular as individual functions, since there are at least two differences between individual functions and individual realizations:

1. Functions are subjective, whereas realizations are objective entities.

⁵⁶ $Cause_{adh}(x,y) \vee Cause_{inst}(x,y) \vee TransTo(x,y) \rightarrow Cause(x,y)$

2. Realizations are function-independent entities and conversely.

The first difference between a function and a realization concerns the structure of functions and realizations. The former are determined by agents, and thus are agent dependent and subjective⁵⁷. Realizations on the other hand are not subjective but are objective entities having causal powers to reach some goals. For a given function an individual entity is objectively its realization or not.

Secondly, realizations and functions are independent from each other. The relation between function and realization is many-to-many. One function may be realized in various ways – it may have many dispositional realizations. On the other hand, one entity may be a realization of more than one function. Moreover, realizations may be described independently of the functions they realize. For example, the process of transporting goods by plane may be described independently of the function it realizes by means of physics as the movement of bodies. The physical description of that process does not refer to the function which the process realizes.

The important point to be mentioned here is that realizations in OF are not reduced to processes as it is the case for instance in [Johansson, 2004] where the state of functioning of an item (which corresponds to our individual realizer) is considered as the participation of the item in a process. Johansson postulates: “no state of functioning without some process” ([Johansson, 2004], p. 5) and illustrates his postulate with an example: “In (its state of) function(ing), the screwdriver participates in a process, namely a certain characteristic movement. In other words, it is in a state of being involved in a process.” ([Johansson, 2004], p. 6). Paraphrasing Johansson’s postulate in our terminology we could say that an entity may have a role of an actual realizer only in the context of a process. In this sense the only realization of a function accepted by Johansson is a process. Although we agree that a function may be realized by a process, we however, do not claim that it can be realized only (and in all cases) by a process. As already mentioned in section 3.5 we do not excluded from our framework instantaneous functions and thus do not exclude realizations which are presentials, e.g. the presential situation containing a presential moth, its covering and an environment is a realization of the function `to camouflage a moth in environment`.

Next, the different modes of both realizations and realizers have been introduced, namely actual and dispositional realizations/realizers. The actual realization/realizer refer to the individuals which actually realize or actually participate in the actual realizations of the function. By means of induction and the mechanism of direct instantiation for every individual actual realization and realizer a corresponding universal, called a minimal realization universal

⁵⁷ Subjectivity of functions is discussed in section 7.2.1.

and a realizer universal, respectively, is found. Those universals are used for identification of the individual dispositional realizations and realizers. Thus, the framework provides means not only for representing individual actual and dispositional realizations and realizers but moreover it handles the general patterns of realizations considered as universals.

Additionally, on the basis of the notion of individual realization, functional realizations are introduced, which enable to predicate that one function realizes another function.

Finally, two relations between functions have been identified, namely trigger and improvement. In contrast to the functional relations discussed in the previous chapter those two are relative to the particular realizations of functions.

6 Ascription of Functions

6.1 Introduction

So far in our considerations we have taken a function-oriented perspective, which means that we analyzed the structure of an isolated function in the first place, the relations between functions in the second place, and we have finally defined the notion of function realization, which glues functions with non-functional entities. However, the perspective taken in function modeling is often different. The starting point is not the function's structure but rather an entity, to which a function is assigned. It is motivated by the need of representing the functionality of entities, in particular of artifacts. In many of the works on functions reported in chapter 2, the question “what is a function” is identified with the question “what does it mean that an entity *has* a function”. This shows the importance of functions and functional knowledge in describing entities. Functions permit to describe entities in an alternative way to a structural or behavioral description. In OF we delimit the issue of what a function is, from the issue of what does it mean that an entity has a function. The former issue, which was investigated so far is intended to be used as a foundation for defining the latter.

We ascribe a function to an entity by means of the *has-function* relation, denoted by $HasFu(x,y,z)$ having the meaning that an entity x , called a *function bearer*, has a function y in context z . A function bearer should not be confused with a functional item or a realizer. A functional item is a role by means of which a function is described, a realizer is a role of an entity in the context of function realization, whereas a function bearer is an entity that is said to have a function. Functions usually are defined independently of the function bearer, but function bearers are often defined in terms of the functions they have ascribed.

The third argument of the has-function relation pointing to the context corresponds to the function in interpretation of Cummins [Cummins, 1975]. Cummins observes that function ascription is relativized to a context (see section 2.3.1). In his understanding, the context of the has-function has an epistemological character, it is an analytical account of a capability of some entity in which an object under question is considered. For example, he writes “it is appropriate to say that the heart functions as a pump against the background of an analysis of the circulatory system's capacity to transport food, oxygen, wastes and so on” ([Cummins, 1975], p. 762). We do not restrict ourselves to the epistemological context but permit also a situational (topological) context. For example, a hammer lying on my desk on a pile of papers

may have a function of holding paper, but clearly when I remove it from my desk it would not have that function anymore. Some considerations on the nature of context involved in the has-function relations appear in chapter 7. Nevertheless, it is outside the scope of the present work to investigate in detail the notion of context which is an issue in itself (see e.g. [Akman, Surav, 1996; McCarthy, Buvač, 1998]). For our purposes it is enough to represent it by means of GFO notions of situations and situoids.

We believe there is not only one kind of the has-function but rather it comes in several flavors. The aim of a top-level ontology of functions in our opinion is to name explicitly those flavors and put them together into one coherent theory. Therefore, we do not define one has-function relation but rather, by analysis of its different usages, we introduce several kinds of it and show their interrelations.

Usually, functions are ascribed to objects that are actors of function realizers. Thus, the first candidates for function bearers are the role-fillers of realizers. Consider as an example a `heart`, which is an actor of the role of a `blood pump`, and it is often said to have a function of `pumping blood`. However, function bearers are not restricted to actors of realizers only, but functions can be ascribed also to realizations. In [Chandrasekaran, Josephson, 2000] authors report the difference between functions of objects and functions of processes. It is not only a `heart` that may have a function of `blood pumping`, but also the process of `heart's behavior` may be said to have that function. In section 5.2.3 we demonstrated that the realization is not necessarily a process; thus we propose to draw the distinction, referred to by Chandrasekaran and Josephson, not between objects and processes but rather between actors of realizers and realizations. We say that a function may be ascribed to a realization and/or to the actor of a realization.

Often to have a function means to realize (actually or potentially) that function. Therefore, one could identify the has-function relation with the realization relation. We, however, distinguish those relations for a number of reasons. Firstly, extensionally those relations are not equal, since, as we will show in the current chapter, it is possible that an object has a function although it is not realizing that function or is not its realization. The differences in extension follow from the differences in intension. The realization relation is the objective relation observable in the world, denoting a goal achievement, whereas the relation of the has-function often involves an element of subjectivity⁵⁸.

In the current chapter we will first consider the ascription of function to individuals, but we do not exclude universals from having the function ascribed as well. For example, about an individual hammer lying on my desk I can say that it has a function of driving in nails, but that

⁵⁸ A function is sometimes assigned to an item due to the subjective feeling –intentions of some agent, see for details section 6.3.

statement can be generalized to the statement that hammers (in general) have the function of driving in nails. In the second case one is not interested in a particular individual hammer but in the class of hammers. In this sense the function is not ascribed to an individual hammer only but rather to a hammer universal. Thus, we say that not only an individual hammer has a function of driving in nails but a universal hammer as well. The former we call an *individual has-function*, the later a *universal has-function*. Let us start with the notion of the individual has-function and on its foundations we will develop its universal counterpart.

6.2 Actual and Dispositional Function

The first two kinds of individual function ascriptions that we will discuss are the *actual* and the *dispositional* has-functions.

Definition 51 (Actual Function). An individual x has an *actual function* f in a situation or situoid s , denoted by $HasFu_{Act}(x, f, s)$, iff x is an actual realization of f in s or x is a role-filler of an actual realizer r of f in s .

$$HasFu_{Act}(x, y, z) \leftrightarrow (RI_{Act}(x, y) \wedge CPart(x, z)) \vee \exists w (R_{Act}(w, y) \wedge CPart(w, z) \wedge HasRole(x, w)). \quad (67)$$

The above definition grasps the dual character of function ascription. A function may be ascribed either to a realization or to the actor of the realizer participating in a realization. An actual realization is not restricted here to the minimal realization - not only a process of transporting goods by a plane, but also the flight has a function of transporting goods.

Frequently, an item has a function ascribed although it does not actually realize this function. For example a car on the parking lot has a function of transporting people although it is not realizing it currently.

We recognize several kinds of non-actual function ascriptions. Firstly, we will introduce is the *dispositional has-function*. Just as in the case of the actual has-function, the dispositional has-function is defined both for realizations and actors of realizers.

Definition 52 (Dispositional Function). An individual x has a *dispositional function* f in a situation or situoid s , denoted by $HasFu_{Disp}(x, f, s)$, iff x is a dispositional realization of f in s or x is an actor of a dispositional realizer of f in s .

$$\begin{aligned}
HasFu_{Disp}(x,y,z) &\leftrightarrow (R_{Disp}(x,y) \wedge CPart(x,z)) \vee \\
&\exists w(R_{Disp}(w,y) \wedge CPart(w,z) \wedge HasRole(x,w)).
\end{aligned}
\tag{68}$$

If a flight from A to B is a dispositional realization of transporting goods, then it follows that it has a dispositional function of transporting goods. Analogously, a plane being a dispositional realizer has a dispositional function to transport goods.

An actor of a dispositional realizer may have a dispositional function ascribed in a *weak* or a *strong* sense, which correspond to the weak and the strong dispositional realizer respectively. For example, a plane involved in the flight being a strong dispositional realizer of transporting goods has a strong dispositional function of transporting goods.

Actual and dispositional functions may seem on the first glance to be too liberal and include also the cases of accidental effects. However, not everything that an entity does or is capable of doing is its function. For example, although a car, actually or potentially, pollutes the environment it does not mean that it has a function to pollute an environment. In OF cases as the above are already excluded by the definitions of function and goal. According to definition 2 the only valid goals of functions are such intended effects that are established for explicit reasons by some agents. In this sense, as long as an effect is not established for a good reason by an agent, one cannot speak about functions. Thus, we cannot consider polluting of an environment as a function until there is a reason for establishing this as a goal. For example, if we consider some mad scientist to have a goal to destroy the ecosystem of the Earth then he may find it useful for his purpose to pollute an environment and thus may define such a function and then assign it to a car.

An entity may have many actual and dispositional effects. Especially the number of the latter is high, if not infinite. However, we would not find that intuitive to say that all dispositions of an item are its functions. In OF we restrict the actual/dispositional function of an item only to those actual or dispositional effects of an item which are related to some (pre)defined system of functions and goals.

6.3 Intended Function

The two types of function ascriptions discussed above seem to meet the intuitions of Johansson when he writes: “to say that functional entities have a function is to say either that they have a disposition to be in a state of functioning or that they are in fact in such a state” [Johansson,

2004]. However, we do not think that function ascription can be restricted to those two cases only.

The common and intuitive interpretation of the statement “ X has a function F ” is “ X was designed to have a function F ”. Those intuitions are reflected by several approaches reported in chapter 2. However, from the statement that an item was designed to realize a given function, does not follow that the item actually realizes that function or that it has a disposition to realize that function, since it may be broken or ill-designed.

Therefore, the function ascription founded on the designer’s intentions cannot be reduced to the actual or the dispositional has-function and should be considered as a third kind of function ascription.

On the other hand, if one agrees to rely in function ascription on the designer’s intentions, then the question could be raised why to take only the designer’s intentions into account. Apart from the designer, the process of artifact construction also involves other parties that have an influence on the function of an artifact. Let us take the example of software engineering process. Most typically the process starts with the requirements of *stakeholders*. On the basis of their requirements software is designed by designers (system analysts) and developed by developers. If software does not meet the stakeholders’ requirements, then we could say that it does not perform its function. It may be the case that software is ill-designed: intentions of designers do not reflect the requirements, or ill-developed: designers’ intention may meet the requirements but they may be inappropriately implemented.

Therefore, it seems that not only the intentions of designers have an influence on the artifact’s function, but other parties like stakeholders should also be included. Thus, the statement that the function of an item is what it was designed for seems to be too narrow, since the function of an item may be not what an item was designed for but what an item was (is) expected to do.

To grasp under one umbrella the intentions of designers, stakeholders and other parties in the assignment of functions we introduce the third kind of function ascription, namely the *intended has-function*.

Definition 53 (Intended Function). An individual x has an *intended function* f in a situation or situoid s , denoted by $HasFu_{Inten}(x, f, s)$, due to some agent who intends x to have the function f in s . Formally,

$$HasFu_{Inten}(x, y, z) \rightarrow \exists q \forall r (Agent(q) \wedge Intent(q, y) \wedge IntCont(y, r, x, y, z) \wedge r :: HasFu). \quad (69)$$

The relation $Intent(a, i)$ has the meaning that an agent a intends i . The content of the intention i is depicted by the predicate $IntCont(i, R, a_1 \dots a_n)$ where R is an n -place relation and a_1, \dots, a_n are arguments of R . In the above definition the relation r is the has-function relation. Thus, herein the content of the belief is “ x has function y in situation z ”.

To illustrate this definition let us follow the example of the hammer lying on my desk. It has a function of `driving in nails` due to the intentions of some agent, a designer, who intended the hammer to have that function during its existence. Note that the situation in which an agent intends an item to have a function, and the situation, in which an item has a function, may be distinct. For instance a designer intends a hammer to have a function of driving in nails during its existence, but the act of intending may take place before the existence of a hammer or during the manufacture process.

The drawback of definition 53 is that it makes function ascription too general and too liberal. Not all agents who intend an item to have a function have the power to ascribe functions to items. The statement that an artifact has function f is not treated equally when expressed by an artifact’s designer and by a person having no particular knowledge about the artifact.

This shows that not every agent’s intention should be considered when ascribing functions to items. Two factors are of importance when ascribing intended functions to items:

- reliability⁵⁹ of an agent ascribing a function,
- number of agents ascribing a function.

As shown in the above example, some agents are more reliable in ascribing functions than others. Some agents (or, to be more precise, some social roles of agents) are of particular importance for the ascription of functions to artifacts. Those are: *stakeholder*, *designer*, *user* and *researcher* roles. The role players of those roles are not disjoint, thus one agent can play several of them.

Each of those roles reflects a different interest in the artifact. *Stakeholder* is the agent, whose requirements should be satisfied by the artifact, and which motivate the manufacture of the artifact. The function required by a stakeholder is called a *required* function of an artifact and is denoted by the predicate $HasFu_{Req}(x, y, z)$ having the reading that x has a required function y in situation/situoid z .

⁵⁹ Reliability of agents is considered as the attribute of an agent in the context of a given community, which was discussed in section 3.3.2.

Definition 54 (Required Function).

$$HasFu_{Req}(x,y,z) \rightarrow \exists qvr(Stakeholder(q,x) \wedge Intent(q,v) \wedge \\ IntCont(v,r,x,y,z) \wedge r :: HasFu). \quad (70)$$

A designer is an agent that designs an artifact. A function designed by a designer is called a *designed* function and is denoted by $HasFu_{Desig}(x,y,z)$, having the reading: x has a designed function y in situation/situoid z .

Definition 55 (Designed Function).

$$HasFu_{Desig}(x,y,z) \rightarrow \exists qvr(Desig(q,x) \wedge Intent(q,v) \wedge \\ IntCont(v,r,x,y,z) \wedge r :: HasFu). \quad (71)$$

A user is an agent who uses an artifact or some other entity. A function intended by a user is called a *user function*, or *ad hoc function* and is denoted by $HasFu_{User}(x,y,z)$, having the reading: x has a user function y in situation/situoid z .

Definition 56 (User Function).

$$HasFu_{User}(x,y,z) \rightarrow \exists qvr(User(q,x) \wedge Intent(q,v) \wedge \\ IntCont(v,r,x,y,z) \wedge r :: HasFu). \quad (72)$$

Finally a researcher, $Researcher(x,y)$ is an agent x who examines an artifact or some other entity y . A function assigned to an item by a researcher is called a *researched* function. The researched function is denoted by $HasFu_{Res}(x,y,z)$, having the reading: x has a researched function y in a situation/situoid z .

Definition 57 (Researched Function).

$$HasFu_{Res}(x,y,z) \rightarrow \exists qvr(Researcher(q,x) \wedge Believe(q,v) \wedge \\ BelCont(v,r,x,y,z) \wedge r :: HasFu). \quad (73)$$

The above function kinds differ among each other not only due to the role of an agent, who intends them, but also due to the time when they are assigned to an artifact. The required and the designed function are intended before or during the existence of the artifact. The user function is intended during the existence of the artifact, whereas the researched function is intended during or after the existence of the artifact. Consider, for instance, an e-commerce software in a given moment m of its existence. We may say that it has a required function of bringing profit in m , if we know that there was a stakeholder who intended, before or during the existence of the software, that it would have that function in m . Analogously, the software may have a designed function in a given situation s , for example to replicate to central database at every 00:00, due to the designer who intended it before s . Next, each user during the existence of an artifact intends that it has some functions, for example for a customer it may have the function of buying cheap second-hand photographer equipment. Finally, a sociologist, during or after the existence of the software, may analyze its social impact and assign to it the function of changing customer habits.

The intended functions are only the matter of intentions of particular agents and do not involve actual nor dispositional realization. Therefore, none of the above function ascriptions guarantees that the software really realizes any of those functions. For example, the needs of stakeholders might not be satisfied, and the software might be unsuccessful in yielding profit. Due to bad programming it might not perform data replication every 00:00, thus the designed function would not be an actual function, either. The user may fail to buy the desired item cheaply. Finally, the theory of social impact of the software may turn out to be false.

The second important factor influencing the intended function, beside the reliability of an agent, is the number of agents intending the function. If, for example, an ancient artifact found on an archeological spot, has an intended function f according to only one archeologist, it is of lower reliability than another function f' , ascribed to it by the scientific community. Again, the high number of agents does not guarantee that the function ascribed is actual or dispositional.

Definitions 53-57 has a recursive character. An item has an intended function in a situation s iff it is intended to have a function in s . The question, then, is what does it mean that an agent intends an item to have a function in a given situation? We distinguish three possibilities, as three kinds of the *has-function* have been distinguished. In the first two cases an item has a function in a given context, because an agent intends that an item actually realizes, or has a disposition to realize, the function in that context. In those cases an agent intends an item to have respectively an actual or dispositional function. In the third case an agent intends that an item has an intended, if only by some other agent, function. Take as an

example an archeologist exploring an ancient artifact. The archeologist may have reasons to claim that the artifact was designed to perform a given function. In this case a function that is assigned to an artifact is an intended-intended function. The researcher intends that a designer intended that the artifact should have a function f in s . Note that the artifact may be wrongly constructed so that not only the artifact never actually realized that function, but also never had a disposition to realize it. This, however, does not disturb the archeologist's claim concerning the intended function. The intended-intended function is independent of an actual or a dispositional function.

In order to block infinite chains of intended functions, we postulate that behind each intended function must stand an actual or a dispositional function. This postulate provides the grounding of the function ascription in reality. In our example the grounding may be provided by an actual function intended by a designer: The archeologist intends that the designer intended the artifact to actually realize the given function in s .

So far we have considered intended function only in the context of artifacts, understood as role-players of the function's realizers. However, an intended function, just as a dispositional and an actual function, may be ascribed not only to realizer-fillers but to realizations as well.

This point is of particular importance in the context of processes and services. Take our running example of an individual process of transporting goods by plane, which is a realization of the function `to transport goods`. This process may be considered as a service delivered by some provider. For that service there may be a stakeholder – a top manager in a provider enterprise, who wants, by means of this service, to shorten the time required for delivering goods. Thus, the required function of that process is then `to improve the time of delivery`. For an individual customer of that service it might have the function `to deliver the wardrobe to Rome`. In turn, for the analyst the service may have a function of `increasing the number of customers`. Thus, we see that an intended function may be ascribed to entities being realizations, just as the actual and the dispositional functions are.

Against the intended has-function one could raise the objection referring to the designer's erroneous intention. In section 2.3.2 we have paraphrased the argument of Keil [Keil, 2003] concerning the nature of the categorial essence to the question concerning the nature of the has-function relation. One may argue that the intentions of the designer cannot be taken as a function of the artifact because if they are erroneous they do not justify the function of an artifact. This problem is solved in OF by reference to the agent's reliability. When concerning the example of Adam, who intended to copy a surgical tool and instead copied a plumber's tool, we can say that Adam's tool has an intended function `to be used in`

the surgery since there is a designer – Adam who intends it to have this function. But on the other hand Adam’s reliability in the area of the surgery tools manufacture is low, and thus the function established by Adam and assigned to the tool due to his intentions is of low reliability.

6.3.1 Inherited Intended Function

In the age of mass production it is seldom that each artifact is individually designed. Rather it is a prototype, which is individually designed, and the remaining mass produced items are copies of that prototype. If we consider mass produced artifacts then we find that many of them lack the designer that designed them individually, and therefore it would follow that they have no designed function, which is counterintuitive. To handle such cases we extend definition 55 of designed function by the *inherited designed* function:

Definition 58 (Extension of Designed Function definition). An individual x has a *designed function* f in a situation or situoid s , denoted by $HasFu_{Desig}(x, f, s)$, if there is a designer who intends x to have a function f in s or x is a copy of an individual x' which has an intended function in a situation s' similar to s ⁶⁰.

$$Designer(q, x) \wedge Intent(q, v) \wedge IntCont(v, r, x, y, z) \wedge r :: HasFu \rightarrow HasFu_{Desig}(x, y, z). \quad (74)$$

$$HasFu_{Desig}(x', y, z') \wedge CopyOf(x, x') \wedge z :: s \wedge z' :: s' \rightarrow HasFu_{Desig}(x, y, z). \quad (75)$$

In the second case an individual inherits the designed function from some other individual out of which it was copied. The relation of copying, denoted by $CopyOf(x, y)$ has the straightforward reading: x is a copy of y , and should be considered as transitive. It is outside the scope of this work to investigate in detail the mechanism, which stands behind that relation.

6.4 Universal Has-Function

So far we have analyzed the ascription of functions to individuals, but universals may have functions ascribed as well. A function may be ascribed to a universal in two different modes.

⁶⁰ In a similar way the definition of the required function can be extended.

In the first mode a universal may have a function ascribed to it due to the fact that corresponding individuals have the function ascribed. For example, the process of boiling of water considered as a universal has an actual function of producing steam in the sense that all individual processes of water boiling have an actual function of producing steam. In the second mode, a function is ascribed not via individual instances but directly to a universal. To illustrate this consider a universal ideology, which is a particular type of a universal, namely a conceptual structure. It may be said that a function of ideology is to have a political impact. Thus, we may say that a conceptual structure has a function ascribed, and it seems that the same holds for other types of universals as well.

In the current work we are interested only in the first mode of function ascription to universals. The ascription of function to a universal can be understood as a universal quantification over the instances of the universal.

Definition 59 (Universal Has Function). A universal x has a function f in a universal situation or situoid s , denoted by $UniHasFu(x,f,s)$, iff all instances of x have the function f in the situations or situoids which are instances of s . Formally,

$$UniHasFu(x,y,z) \leftrightarrow \forall v s (v :: x \wedge s :: z \rightarrow HasFu(v,y,s)). \quad (76)$$

Every type of the above-discussed individual function ascription may be generalized to the universal level, and thus the following universal function ascriptions are introduced:

Definition 60 (Universal Actual Has Function). A universal x has an actual function f in the context of a universal situation or situoid z , denoted by $UniHasFu_{Act}(x,f,z)$, iff all instances of x have the actual function f in the situations or situoids being instances of the universal z . Formally,

$$UniHasFu_{Act}(x,y,z) \leftrightarrow \forall v s (v :: x \wedge s :: z \rightarrow HasFu_{Act}(v,y,s)). \quad (77)$$

Definition 61 (Universal Dispositional Has Function). A universal x has a dispositional function f in the context of a universal situation or situoid s , denoted by $UniHasFu_{Disp}(x,f,z)$, iff all its instances have the dispositional function f in situations or situoids being instances of the universal z . Formally,

$$UniHasFu_{Disp}(x,y,z) \leftrightarrow \forall v s (v :: x \wedge s :: z \rightarrow HasFu_{Disp}(v,y,s)). \quad (78)$$

Definition 62 (Universal Intended Has Function). A universal x has an intended function f in the context of a universal situation or situoid z , denoted by $UniHasFu_{Inten}(x, f, z)$, iff all its instances have the intended function f in situations or situoids being instances of the universal z . Formally,

$$UniHasFu_{Inten}(x, y, z) \leftrightarrow \forall v s (v :: x \wedge s :: z \rightarrow HasFu_{Inten}(v, y, s)). \quad (79)$$

In most cases functions are ascribed to universals rather than individuals. For instance it is often said that the function of heart is to pump blood, not that this is a function of *my* heart.

6.5 Function Bearers

An entity, to which a function is assigned by means of the has-function relation we call a *function bearer*. Function bearers, and especially artifacts, are often defined in term of functions they have ascribed. So far we have permitted both individuals and universals to be function bearers, but we have not investigated of what ontological kinds function bearers are. In fact we put no ontological constraints on the ontological kind of function bearers. We think that an arbitrary entity can have a function ascribed. The table below lists the exemplary function bearers of several ontological kinds of GFO.

Ontological kind	Function bearer
Universal	Ideology has a function of political impact.
Persistent	A hammer considered as a persistent has through out its lifetime the designed function to hammer nails.
Presential	A stone at a given time boundary t , considered as a presential may have a user function to provide sitting at t .
Process	A process of flight may have an actual function to transport goods.
Situoid	A picket considered as a situoid may have a function to influence the government's decisions.
Quality Value	A color of a moth has a function to camouflage.

Table 4. Function bearers of various ontological kinds.

6.6 Functionality and Multiple Function Ascriptions

Different kinds of the has-function relation have been presented in the current chapter. Now, let us demonstrate how they can all be used for the description of an items' functionality. We introduce two straightforward postulates: (1) an entity may have more than one function attributed; (2) a function may be attributed to an entity by different kinds of has-function relations.

The first intuitive postulate can be well illustrated with such an object as for example a Swiss pen knife, which has more than one function. The total of all functions of an individual we call the *functionality* of an item. The second postulate is more interesting and is especially important in the context of malfunction, which is discussed later. As an illustration consider an individual hammer in the situation *s* of lying on my desk. The hammer was designed to realize the function of hammering nails, which means that the designer intended the hammer to have a function of driving in nails during its lifetime. The situation *s* is a part of the hammer's lifetime, hence a designed function of the hammer lying on my desk is *F*: to drive in nails. The hammer, however, is currently not participating in a realization of that function, but is lying on my desk. Thus, *F* is not its actual function. Nor is *F* its dispositional strong function, because the process of the hammer lying on the desk is not a dispositional realization of the function to drive in nails. However, since the hammer is well designed and manufactured, and because it is not broken, it has a weak dispositional function of driving in nails.

Moreover, the hammer is lying on a pile of papers, because I have put it there in order to prevent the papers from being blown off by the wind, and thus its user function is to prevent the papers from being blown off by the wind. Because the day is windy and the window is open, the hammer *actually* prevents the papers from being blown off by the wind. Therefore, preventing the papers from being blown off is not only an intended user function of the hammer but it is its actual function as well. If the wind stopped blowing tonight, the function of preventing the papers from being blown off would become a dispositional weak function.

The example shows yet another important feature of the framework. Although the intended functions are most common to artifacts, it does not follow that artifacts have only functions of that kind ascribed. This feature is important when considering the example presented in ([Kitcher, 1993] p. 380 – 381). During the process of machine manufacture, unbeknownst to the designer there appears an error in the design. The design does not include the connection between two parts of the machine, which is necessary to make the machine work properly. Luckily, an accidental dropping of a screw provides that connection. Now, the

screw clearly does not have a designed function, but it still has a function to provide the connection. Kitcher derives that function from the overall intended function of the machine, whereas in OF it would be considered as an actual, but not intended function of the screw in the machine.

6.7 Malfunctions

Beside functions there also malfunctions that can be ascribed to items. About a broken engine one says that it malfunctions, in the sense that it is not functioning properly. In OF malfunction is introduced as a ternary relation, denoted by $Malfu(x,y,z)$, having the meaning that an individual x is malfunctioning with respect to a function y in a context z .

Especially in philosophical literature many works have been committed to the problem of malfunctioning (see section 2.3), since the representation of malfunctioning brings difficulties for the philosophical theories of functions. For example, the Cummins theory of functions meets a problem in representing malfunctions, since it considers functions only as the current dispositions of an item, and thus lacks the normative aspect. Neither are etiological theories without problems. For example, Davies [Davies, 2000b; Davies, 2000c] argues that etiological (or selective) theory has no means to handle malfunctions. He maintains that etiological theories define functional types as the items having heritable properties that resulted in selective success. This implies that the items that lack a relevant success property cannot be considered as functional types and thus cannot be said to malfunction.

Malfunction is important not only from the perspective of a philosophical discussion, but also in the context of validation, control and evaluation of artifacts. It is also relevant for the purpose of representing biological knowledge which commonly refers to malfunctions. The main points can be formulated as the following questions: (1) Under what conditions is an item malfunctioning? (2) What is the difference between malfunctioning and a mere lack of function realization?

Considering (1) we postulate that for the assignment of malfunction to an item in a given context c two conditions must be satisfied:

1. The item is not realizing (actually or dispositionally) function f in context c .
2. The item should realize (actually or dispositionally) function f in context c .

The first condition is clear - only items that are not realizing a function may be considered malfunctioning with respect to this function. On the other hand, it is not always the case that an entity which does not realize a given function is said to be malfunctioning. Even though a car engine is not realizing the function to enable steering the vehicle, it is not

considered to be malfunctioning, since this function is simply not what an engine is supposed to do. The example shows the need of delimiting malfunction from the ordinary lack of function realization. This difference is the subject of condition (2), and requires a better understanding of the normativity of function ascription. The question to be stated here is what does it mean that an item *should* do something.

We find three general answers for this question: an item should do something because (1) it was required/designed to do so, or (2) it used to do it in the past, or (3) other items of that kind do it. On the basis of these three answers there are three kinds of malfunctions defined: the malfunction with respect to the intended function, (2) the malfunction with respect to the item's history, and (3) the malfunction with respect to other instances of the item's kind.

6.7.1 Malfunction with respect to Intended Function

The item is malfunctioning when it is not able to do what it should. In the case of artifacts, agents involved in the creation of the artifact, dictate what an item should do. Thus, on the basis of the intended function we define the first type of malfunction – *malfunction with respect to intended function*:

Definition 63 (Malfunction with respect to Intended Function). An individual x malfunctions in a given context c with respect to an intended function f , denoted by $MalFu_{Inten}(x, f, c)$, iff:

- (a) x has required/designed function f in c and
- (b) x does not have dispositional or actual function f in c . Formally,

$$MalFu_{Inten}(x, y, z) \leftrightarrow (HasFu_{Req}(x, y, z) \vee HasFu_{Desig}(x, y, z)) \wedge \neg(HasFu_{Act}(x, y, z) \vee HasFu_{Disp}(x, y, z)). \quad (80)$$

Intuitively, the definition says that an item is malfunctioning if it is not able to do what some agent intended it to do. Malfunction is therefore agent-dependent and the degree of malfunction, just as the degree of intended function, varies with respect to the reliability and the number of agents, who intend an item to have a given function. For example, if a hammer lying on my desk turns out not to prevent the wind from blowing the papers off my desk, then it does not perform the intended user function and thus it malfunctions with respect to that function. Surely no one will accept the warranty return of such a hammer, simply because preventing papers from being blown off is not the required or designed function of the

hammer. Therefore, only malfunction with respect to the designed or the required function are considered as intended malfunction.

Dominant Intended Function

The artifact design is a process extended in time and within it artifacts are often redesigned. In the software engineering process several methodologies even assume that software is redesigned. For example in prototyping, software evolves from the prototype to the final product. It happens that the final product does not satisfy the functions that were required in the first stages of the design because the design evolved. In this case although the product does not realize the functions that it was initially required or designed for, we would not say that it malfunctions. In order not to treat such cases as malfunctioning we define the subsets of required and designed functions, called dominant required functions and dominant designed functions, respectively.

Definition 64 (Dominant Intended Function). A function f of an artifact a in a context c is called a *dominant required/designed* function iff a was designed or required to realize f in c and f was not rejected in the later process of design.

Now, in order to handle the cases of prototyping and redesign the definitions of required and designed functions should be restricted to the dominant required and designed functions.

6.7.2 Malfunctions with respect to History

So far we have restricted malfunctions to artifacts, but it is not only artifacts that malfunction. Take for example body organs - a heart is considered to malfunction when not pumping blood. However, malfunction of a heart cannot be explained by reference to designed or required functions, since this would assume some agent, who designed a heart. In case of organs it is problematic to find such an agent without slipping into the theological discussion about the origins of life. The ontological framework of Chandrasekaran and Josephson in our opinion falls into difficulties of that kind. In [Chandrasekaran, Josephson, 1997] function is understood in terms of design, and nature or evolution is taken as a designer of biological organs. However, the understanding of nature or evolution as a designer we find as an oversimplification, since in the broader ontological framework this requires to regard them as some kind of agentive and intentional entities. Alternatively, beside the notion of intended malfunction, applicable for artifacts, we suggest introducing other kinds of malfunction suited

also for non-artifacts. We rely here on the ideas found in [Upton, 2004], which we develop, formalize and incorporate into our ontology.

The first non-intentional criterion is provided by the reference to the history of a given individual. Often, when complaining about the malfunction of some item, we say “so far it was working fine”. Here, we judge an item to be malfunctioning not by references to the intentions of a designer or a stakeholder, which in fact for most of us are unknown, but rather we refer to the history of an item, which we know. In this sense, an item is recognized as malfunctioning when it does not realize the function in a given situation, although it used to do so in the past.

Definition 65 (Malfunction with respect to history). An individual x *malfunctions with respect to its history* in a situation or situoid s with respect to a given function f , denoted by $MalFu_{His}(x, f, s)$, iff x used to have a dispositional or actual function f in a situation/situoid s' similar to s and does not have it in s .

Two situations or situoids we call similar when they are instances of the same universal. Thus, we say that an item has a historical dysfunction if it is not functioning in a situation which is the instance of the same universal as the situation in which an item had a disposition to f ⁶¹.

$$MalFu_{His}(x, y, z) \leftrightarrow \neg (HasFu_{Disp}(x, y, z) \vee HasFu_{Act}(x, y, z)) \wedge \exists uv(z \vdots v \wedge u \vdots v \wedge (HasFu_{Disp}(x, y, u) \vee HasFu_{Act}(x, y, u))). \quad (81)$$

Malfunction with respect to the history holds for both artifacts and for non-artifacts. By analogy to a malfunctioning heart an engine is considered as malfunctioning if it is not working today although it was working in the past in exactly the same conditions.

Malfunction with respect to the history has difficulties concerning the change over time. An item may by natural (healthy) evolution lose some functions but it does not make it malfunctioning. This issue is discussed below in section 6.7.4.

6.7.3 Malfunction in Comparison

Yet another way of identifying malfunction is the comparison of an item with similar individuals in similar conditions. For example, one may judge one's car to be malfunctioning if it does not manage to drive up a steep hill, whereas other cars of the same model manage to

⁶¹ We refer to direct instantiation here since we are interested in the maximal similarity of the situations compared.

do so. In this case one is not referring to the history of an individual but to other individuals instead.

Definition 66 (Malfunction in comparison) An individual x being an instance of a kind u malfunctions in comparison with other individuals with respect to a function f in a situation/situoid s , denoted by $Malfu_{Kind}(x, f, s)$, iff:

- (a) other instances of u have a dispositional or actual function f in situations or situoids similar to s ,
- (b) x does not have the dispositional or actual function f in s . Formally,

$$\begin{aligned}
 Malfu_{Kind}(x, y, z) \leftrightarrow \exists prst \ (x \vdash t \wedge s \vdash t \wedge z \vdash r \wedge p \vdash r \wedge s \neq x \wedge \\
 (HasFu_{Disp}(s, y, p) \vee HasFu_{Act}(s, y, p))) \wedge \\
 \neg (HasFu_{Disp}(x, y, z) \vee HasFu_{Act}(x, y, z)).
 \end{aligned} \tag{82}$$

In this definition, similarity of two individuals is represented by the common universal. In this sense an item is compared with other instances of the universal, which it instantiates. However, the universal used here is not a universal of an arbitrary type but instead it is a *kind universal*. It is not our purpose to investigate the problem of natural kinds (for discussion see for example [Wilkerson, 1995]) or the problem of typology of universals. For our purposes only three types of universals are distinguished: universal functional item – a universal role depicting items in purely functional terms, universal realizer – a structural universal depicting structure of items in the context of the realization of some function, and finally the kind universal which is a universal grouping entities considered to be of the same kind, such as e.g. human beings. Kind universals extend natural kinds and we do not exclude kinds such as artifacts, but they should not be identified with universal realizers. Thus, two individuals can be instances of the same kind but not of the same realizer, e.g. the plane with loaded cargo and the plane with empty cargo space are of the same kind but are not instances of the same universal realizer goods transporter, since only the latter can be loaded with goods, and thus can realize the function of goods transportation.

6.7.4 Priorities of Malfunctions

The three introduced kinds of malfunction are independent from another. For example, it may be the case that an artifact malfunctions with respect to its history but not with respect to its intended function. Consider a space probe designed to work only for a given period of time. After the end of the period it is not expected to be functioning anymore. If in a situation which

takes place after the end of that period, the space probe is not realizing its function, then according to the design it is not malfunctioning. However, if this situation satisfies the requirements of the function, then the space probe is malfunctioning in that situation with respect to its history, since it does not realize the function which it was realizing in an analogous situations in the past. Therefore, there arises a conflict – the space probe is malfunctioning with respect to its history, but not with respect to its intended function. To solve conflicts of this kind the priority of has-function and malfunction kinds should be given.

In OF the strongest malfunctions are considered to be those with respect to the designed and required functions. An artifact may be considered malfunctioning with respect to its intended function, although it does not malfunction with respect to its history or in comparison to other instances of the kind. Concerning the first case, an artifact could never have realized the function in question, for example due to a design flaw or a production defect and thus it would not be said to malfunction with respect to its history. Concerning the second case, all members of the kind might be wrongly designed and none of them might realize the function. Therefore, the item would not malfunction with respect to other instances of the kind, neither.

Similarly if an artifact is not malfunctioning with respect to its required/designed function but is malfunctioning with respect to its history or with respect to other members of the kind then it is said not to be malfunctioning.

For non-artifacts, which lack designed and required functions, the malfunction due to comparison with other members of the kind is prior to the malfunction with respect to the item's history. An item defective throughout its lifetime is not malfunctioning in a given situation s with respect to its history but nevertheless it may be detected malfunctioning in comparison to other items of its kind. For example, an inborn defect cannot be detected as malfunction with respect to the history but it can be detected when comparing an item with other instances of the kind.

From the above considerations we can provide the following hierarchy of malfunctions:

1. Required/ designed malfunction.
2. Malfunction with respect to other instances of universal.
3. Malfunction with respect to item's history.

6.7.5 Side-Effect Malfunction

We have considered malfunction with respect to a given function f as the lack of a dispositional or actual realization of f . In this sense an item is malfunctioning if the goal of a function cannot be reached by the item. Such a malfunction we call a *goal failure* malfunction. However, there are malfunctions which do not concern the goal. A goal of a function may be

reached but nevertheless the item may malfunction, since it results additionally in undesired side effects. Such a malfunction we call a *side effect* malfunction. For example, an engine may dysfunction not because it does not generate circular motion, which is its designed function but because of a too high consumption of fuel.

We handle the side effect malfunctions in OF with the help of a broad understanding of the function's goal, comprising also restrictions on goals. As already discussed in chapter 3, we understand a function as a specification of what is supposed to be done. In this sense everything that is intended to be achieved is considered as the goal of a function, also that which we treat as a secondary goal, or a restriction on the main goal. In our example we would not consider the function assigned to an engine as to generate circular motion, but rather we would extend it to the following: to generate circular motion AND not to exceed the given level of fuel consumption. In this case an engine that generates circular motion, but uses too much fuel, dysfunctions, since it does not reach the second of its goals. In this sense it is not realizing its multiple-goal function. The distinction between primary and secondary goal can be represented by the usage of goal priorities. In our example, if required, one could say that the generation of circular motion has a higher priority than the low level of fuel consumption.

6.8 Summary

In the current chapter we have investigated the notion of function ascription and have introduced a number of notions that permit to model it. Several types of function ascription have been recognized, among them the intended has-function typical of artifacts and the actual and the dispositional has-function, founded on the notions of actual and dispositional realizations applicable to non-artifacts as well. The notions introduced permit to ascribe functions to arbitrary entities, involving both individuals and universals, processes, persistants and presentials.

OF permits to assign not only functions to entities but also malfunctions. Several types of malfunction together with their interdependencies have been introduced. Among them are the malfunctions with respect to the intended function, the history of an entity, and the comparison to other members of the kind. The side effect malfunctions are handled by the extended understanding of the goal of a function.

7 Ontological Status of Functions, Classifications and Architecture

7.1 Introduction

In the current chapter we touch the fourth main problem area of the top-level ontology of functions mentioned in chapter 2, namely the ontological status of functions and the incorporation of OF into a wider ontological framework of GFO. In the first place we discuss characteristics of functions, and analyze the candidates for function definition against them. Those investigations result in the definition of function and enable to plug the notion of function in an appropriate place in the taxonomy of GFO.

Moreover, we introduce the most general classifications of functions as well as the architectural principles for the organization of functional knowledge. These two issues are of particular importance in the context of the application of OF in functional modeling and the design of domain ontologies combining both functional and non-functional knowledge.

7.2 Characteristics of Functions

On the basis of the developed framework and the provided analysis of other works, in the current section we will investigate the characteristics of functions, namely *contextuality*, *subjectivity*, and *goal-orientedness*.

7.2.1 Context and Subjectivity

As a first characteristic of functions we will investigate this saying that functions are contextual entities. Our aim here is not to investigate the broadly discussed notion of context (see e.g. [Akman, Surav, 1996; McCarthy, Buvač, 1998]), but rather we intend to show that a context is commonly involved in the definition of a function, and that several types of context can be distinguished, when speaking about functions.

In AI approaches to functional modeling the notion of context is often involved in the definition of a function. For instance, in CFRL function is defined by means of context considered as an environment in which the device is supposed to function. In turn, in Chandrasekaran and Josephson's ontological framework, context, called there Mode of Deployment (MoD), is the selective description of the environment of the object to which a function is assigned. MoD is a specification of causal interactions between an object and other objects in the world. In the example provided in [Chandrasekaran, Josephson, 2000] the battery lying on the piece of paper may be considered in two MoDs: one, in which it is connected to electrical terminals of some object, and the other, where the bottom surface of the battery is in the *a_top* relation with an object paper. Those two MoDs result in two different functions ascribed to the battery - the function of supporting paper and the function of providing voltage. MoD therefore can be considered as a context in which an object is investigated and which determines function ascription.

Context was also recognized as a factor in philosophical definitions of functions. Cummins [Cummins, 1975] refers in his definition of function not only to the system containing an object, but to an analytical account of that system's capacity. Thus, the function of an object varies not only with respect to some larger whole, in which an item participates, or is located in - that is to some MoD, but also with respect to the way this whole, or more precisely its capacity, is explained. In this sense a context seems to be considered on the epistemological level, where the object's function is a part of explanation/analytical account of some capacity of the containing system.

Finally, a particular type of context that influences a function is an agent. Searle [Searle, 1995] touches on that point when he says that a function is always determined by an agent, an observer, who finds the function in a given object.

The above examples show that the notions of function and function ascription in contrast to, for example the notion of behavior, are often recognized as contextual. Moreover, the examples show that there is no unified understanding of the function's context but rather various intuitions are hidden under that notion. On the basis of the brief review of functional contexts presented above, the following preliminary kinds of functional context can be distinguished:

1. *Situational (topological) context* includes the containing system, other components of the containing system, and the environment which an item effects. In GFO terms the situational context is provided by the situation or situoid in which an entity is participates.
2. *Causal context* includes entities, on which the entity, having the function ascribed, has a causal impact. Causal context may be considered as a part of the situational context.

3. *Epistemological context* involves the way a phenomena is explained. It often concerns some bigger whole, in which the entity under question is involved, namely its situational or causal context.
4. *Context of an agent* determining and ascribing a function. The function is agent-relative and thus to some extent has a subjective character.

The first three types of context touch only the relation of function ascription but not the function itself. Due to the different situational, causal or epistemological contexts, different functions may be ascribed to a given item.

In the examples discussed in chapter 6 mostly topological and causal contexts were considered. However, it seems that has-function and malfunction can be ascribed also in the epistemological context. For example, when considering the Cummins example, a heart has-function not only in a circulatory system (topological system) but also in the epistemological context of an explanation of the circulatory system's ability of transporting stuff. Similarly, a malfunction may be assigned against an epistemological context.

The fourth context kind, which implies subjectivity, in contrast to the three former ones affecting only the ascription of function, is present also at the level of the structure of functions. Considering the function structure, subjectivity is represented in OF by reference to an agent, who by the establishment of a goal determines the structure of a function. We postulated that there is no function without an agent and every function is relativized to an agent establishing its goal. If some other agent does not recognize that goal then the function is not to be recognized by him either. In this sense a function is always subjective.

In addition, subjectivity is also present in the ascription of a function to an item. Here, however, not every kind of the function ascription involves an agent, but only the intended has-function. An item has an intended function due to some agent that ascribes the function to that item. In this sense the has-function relation is subjective - function ascribed to an item by one agent does not have to be ascribed to it by another agent.

7.2.2 Teleology and Goal-Orientedness

The next commonly discussed aspect of a function is the goal-orientedness. For example, paraphrasing the definition of a function as a teleological interpretation of behavior, one could say that a function is a goal-oriented behavior. Several definitions of function discussed in chapter 2 include a goal. And likewise in OF a goal is considered as a determinant of a function. Moreover, as was demonstrated when discussing the structure of function other determinants are necessary in order to adequately determine function, but it is a goal, which

seems to be crucial for the notion of function. Even a function's label, which is an informal but commonly used technique for modeling functions, is goal-oriented.

Summing up, the following characteristics of a function should be taken into account when investigating the ontological status of functions:

1. Function ascription is contextual. Whether an entity has a function depends on the context.
2. Function is subjective. The function structure depends on an agent establishing a goal, whereas the intended function ascription depends on the agent, intending an item to have a function.
3. Function is goal-oriented. A goal is a crucial determinant of a function.

7.3 Candidates for Functions

The purpose of this section is to investigate the ontological kind of a function. Firstly, we will investigate against the above function characteristics two popular candidates for functions, namely processes and goals. Secondly we will provide the definition of function, which enables integration of the notion of function into GFO.

7.3.1 Functions as Processes

Functions are often considered to be processes. The notion of a process differs across formalisms, however some general intuitions are common. Here, we understand a process in the GFO sense, where it is considered as a perduring entity extended in time, which, in contrast to a presential located at the time boundary, is not fully present at any time boundary. Processes in GFO are said to happen throughout time and are contrasted to persistants, which grasp the endurantistic view. Process are often labeled with goal-oriented labels, which suggests treating functions in terms of processes. For instance, the label `to transport goods` could be adequate both for the function and the process of transportation.

In AI approaches to functional modeling there are close correlations of functions and processes. For example, in [Sasajima et al., 1995] function is defined as a teleological abstraction of behavior which can be seen as a process or a layer of a process. Goal-orientedness can be reconstructed in terms of culmination, which is a criterion for classification of events [Casati, Varzi, 2002]. An event has a culmination if it has a finishing point. Thus, a goal-oriented process is a culminating process which reaches the goal in its

finishing point. It therefore seems intuitive to consider functions as a particular type of processes, namely achievements or accomplishments.

However, despite those similarities we found, as already mentioned in passing in section 5.8, that functions cannot be identified with processes. Firstly, we observe that not all functions can be regarded as culminative processes; in particular continuous functions are realized by non-culminative processes. For example, the function of pumping blood cannot be treated in terms of a culminative process, because the process of pumping blood lacks a culmination. Secondly, functions cannot be considered to be processes in general, since their realizations, in contrast to processes, are not necessarily time-extended. A situational realization of the instantaneous function of camouflaging a moth is a presential and thus is not time extended.

Hence, since we do not consider functions in terms of processes, the definition of function as a teleological abstraction of behavior is too narrow for our purposes, and in our opinion does not fully grasp the ontological nature of functions.

7.3.2 Functions as Goals

Because a function is a teleological entity, one could identify it with the notion of a goal. In this sense a goal in which a function results could be treated as a function itself or, in words of Chandrasekaran and Josephson, the function of an object could be defined as the effects of the object on its environment [Chandrasekaran, Josephson, 1997]. This intuitive solution seems to be adequate for many purposes, however the scope of its applications depends on the accepted interpretation of the goal. In FR and CFRL a goal is interpreted very broadly - it includes not only the state of the world, which is the result of a function but also the whole chain of causal factors that lead to that result state. A goal, then, is not only regarded as what should be achieved but also as the way it is achieved.

In OF we refer to a much more restricted notion of a goal. A goal in our understanding is an intentional entity referring to an arbitrary chunk of reality that is expected to be achieved by the function realization and is distinguished for some reason by an agent. A goal, regarded in this narrow sense, is insufficient for the adequate identification of a function, since functions resulting in the same goal may be different with respect to their requirements, or functional items.

The next argument against the reduction of functions to goals, or effects, comes from analysis of the temporal location of goals and functions. In our framework functions lack a direct relation to time. However, as presented in section 3.5 this relation can be derived from the temporal relations of the requirements and goal of a function. In this sense we may

informally speak about temporal extension of functions. In our opinion a function may have a different time location than a goal, and since one entity cannot have two different time locations, therefore a function and a goal should be considered distinct. This holds especially for sequential functions. Consider the function to deliver a letter on 5th June. Here, the goal is a presential relation (`located_in`) at the time boundary $T = \text{the fifth of June}$ ⁶². The realization of that function is the process which has a right boundary at T . In this case the goal is “at the end” of the function realization. Concluding, we argue that a function is a goal-oriented entity, not a goal itself.

7.3.3 Functions as Intentional Entities

On the basis of the constructed framework and the discussed above characteristics of functions we propose to define functions as follows:

Definition 67 (Function). A function is an intentional entity defined in purely teleological terms by the specification of a goal, requirements and a functional item which commonly is ascribed by means of the has-function relation to entities that in some context are the realizations of the goal, execute such realizations or are intended by a reliable agent to do so.

The first part of the definition refers to the structure of the function introduced in chapter 3. According to it functions are teleologically defined against goals valued by some agent but are not identified with the goals since they also include the specification of requirements and the functional item. In contrast to the definitions discussed in section 2.1.2, the above definition does not refer neither to behaviors nor processes, and thus does not exclude non-processual functions, but instead it is founded on the notion of *intentional entity*.

An intentional entity is understood as an *abstract* entity of the mental strata, i.e. a mental representation, a thought, an idea which is dependent on some agent. Hence, functions in our understanding are not entities of an objective world but are subjective, agent-dependent entities. However, we admit that functions can be shared by a group of agents and in this sense they may be relativized not to an individual agent but to a community of agents and inside the community they may be considered to be *objective*.

The second part of the definition refers to the introduced relation of the has-function which grasps the contextual character of function ascription and is founded on the realization

⁶² The *fifth of June* is regarded here for the sake of simplicity as a presential although it could be considered as a chronoid.

and realizer relations. Reference to both - the realizer and the realization - covers, among others, the distinction between functions of objects and of processes. The subjectivity of functions is stressed additionally by the intended has-function, which is an assignment of functions to entities due to the intentions of reliable agents.

The definition introduced reveals similarities to the definition of [Sasajima et al., 1995], where a function is considered as the teleological interpretation of behavior. It seems that “the interpretation of behavior” can be interpreted as an abstract mental entity which is depicted in purely teleological terms. The difference between those definitions, apart from the problem of references to the notion of behavior discussed before, lies in the presupposed order of function construction. In [Sasajima et al., 1995] goals and behaviors are primary to functions - for a given goal and behavior, or set of goals and behaviors, a function is constructed by the interpretation of that behavior in the context of the goal. In OF function is secondary to the goal but not to the behavior or any other entity realizing it. Thus, our model assumes the order from function to realization, whereas in [Sasajima et al., 1995] the function is constructed or interpreted from the given behavior.

7.4 Classifications of Functions

For the purpose of modeling it is useful to provide not only a general framework of representing functions and the function ascription, but also a taxonomy of functions. In the current section we propose several classifications of functions, which are organized into two groups called *intrinsic* and *extrinsic* classifications. Intrinsic are those classifications having the principle of distinction (differentia) intrinsic to functions. A differentia is considered to be intrinsic to functions if it refers to the function structure, and functional relations only. On the other hand, a differentia is extrinsic to a function if it refers to notions beyond the structure and functional relations, for example to the realization of a function. Thus, functions may be classified according to their realizations, but one should keep in mind that this classification is secondary and is founded on the classification of realizations. For the sake of completeness, the current section also includes the classifications discussed already in previous chapters.

7.4.1 Intrinsic Classifications

Classification Based on the Time-Extent of a Goal

A goal is an arbitrary chunk of reality, and as such may have various time-extents. In the context of GFO we recognize two primary kinds of temporal locations of entities, namely the projection to a chronoid and the location at a time boundary. The former is typical for processes, whereas the latter for presentials. On the basis of this distinction two kinds of basic functions are found:

- *Accomplishment Function*. If the goal of a basic function f is a presential, then we call f an *accomplishment* function, and denote it by $Fu_{Accomp}(f)$. An exemplary accomplishment function is to deliver mail at given time point.

$$Fu_{Accomp}(x) \leftrightarrow Fu_{Basic}(x) \wedge \forall y(GoalOf(y,x) \rightarrow Pres(y)). \quad (83)$$

Two kinds of accomplishment functions are distinguished in OF: sequential and instantaneous functions. The difference between them concerns the temporal distance between the requirements and the goal. In the former case the goal appears after the requirements, whereas in the latter both are present at the same time boundary.

- *Continuous Function*. If the goal of a basic function f is an entity extended in time, then the function f is called a *continuous* function and is denoted by $Fu_{Contin}(f)$. For example, for the goal: blood is being pumped, which is a process extended in time, the corresponding function to pump blood is a continuous function.

$$Fu_{Contin}(x) \rightarrow Fu_{Basic}(x) \wedge \forall y(GoalOf(y,x) \rightarrow Proc(y)). \quad (84)$$

Intuitively the difference between the accomplishment and continuous functions is that, whereas the former is aimed only at a disposable accomplishment of a given result, the latter aims also at a continuous support of that goal for a given period of time.

This classification may also be applied for functions having universal goals. Although universals in GFO are not related to time, they can still be classified due to the time extent of their individual instances. In this sense the function with a universal goal, whose instances are presentials, is classified as an accomplishment function.

The classification may be extended to non-basic functions by the decomposition of the non-basic goals. And so a non-basic function, all of whose goals are presentials located at

coinciding boundaries, is called an accomplishment function, whereas a function whose at least one goal is a process is called a continuous function.

Classification Based on the Relations between Functions

The classification below is founded on the classification of the relations between functions. For a given goal an arbitrary function may be classified as one of the following:

- *Performer*. For a given chunk of reality x if x is a goal of a function f , then f is called a *performer* function of x . An actual realization of a performer function is a sufficient condition for the fulfillment of the goal x , and is denoted by $Fu_{Perform}(f, x)$.

$$Fu_{Perform}(x, y) \leftrightarrow GoalOf(y, x). \quad (85)$$

- *Enabler*. For a given chunk of reality x if a function f enables a function f' , which is a performer function of x , then f is called an *enabler* function of x , and is denoted by $Fu_{Enable}(f, x)$.

$$Fu_{Enable}(x, y) \leftrightarrow \exists z (Enable(x, z) \wedge GoalOf(z, y)). \quad (86)$$

- *Supporter*. For a given chunk of reality x if a function f supports a function f' , which is a performer function of x , then f is called a *supporter* function of x , and is denoted by $Fu_{Support}(f, x)$.

$$Fu_{Support}(x, y) \leftrightarrow \exists z (Support(x, z) \wedge GoalOf(z, y)). \quad (87)$$

- *Preventer*. For a given chunk of reality x if a function f is preventing a function f' , which is a performer function of x , then f is called a *preventer* function of x , and is denoted by $Fu_{Prevent}(f, x)$.

$$Fu_{Prevent}(x, y) \leftrightarrow \exists z (Prevent(x, z) \wedge GoalOf(z, y)). \quad (88)$$

- *Neutral*. For a given chunk of reality x if x is not a goal of f and f is not related by any of functional relations to a function f' which is a performer function of x , then f is called a *neutral* function for x .

$$Fu_{Neutral}(x, y) \leftrightarrow \neg (Fu_{Perform}(x, y) \vee Fu_{Enable}(x, y) \vee Fu_{Support}(x, y) \vee Fu_{Prevent}(x, y)). \quad (89)$$

The classification of functions based on their role in the achievement of a given goal is especially useful in cases of hierarchical goal modeling, where one goal is central for the whole model. In such cases this classification helps to organize functions, with respect to their influence on the root goal.

The distinction between performers, enablers, supporters, preventers and neutral functions is also applicable for functional items and realizers. A functional item/realizer of the performer function of a goal g is called a performer of goal g , of the enabler function of goal g - an enabler of g , etc.

Classification Based on the Complexity of the Goal

A goal is the chunk of reality which is an intended result of the function. Goals may be either basic or non-basic. Depending on the type of goal the following function kinds are distinguished:

- *Basic Function*
- *Complex Function*
- *Coherent Function*
- *Multiple Goal Function*

If the elements of a non-basic goal do not compose a coherent entity, then the function is called a multiple goal function. The basic, complex, coherent and multiple goal functions are discussed in section 3.3.8.

7.4.2 Extrinsic Classifications

Beside intrinsic classifications of functions also extrinsic ones can be provided. In contrast to intrinsic classifications the principles of distinction used in extrinsic classifications neither pertains to the structure of a function nor to the interrelations between functions. Instead, they are founded on the function's realization and ascription. Thus, these classifications pertain primarily to realizations and function ascriptions, and only indirectly to functions.

Classification Based on the Realization-Dependent Relations

This classification is the extension of the classification of functions based on their interrelations. Here, the classification is based on the role one function has in the given

realization of the other function. For a given goal an arbitrary function may be classified as one of the following:

- *Trigger*. For a given chunk of reality x if a function f triggers a realization r of a function f' , which is a performer function of x , then f is called a *trigger* function of x and is denoted by $Fu_{trigger}(f,x,r)$.

$$Fu_{trigger}(x,y,z) \leftrightarrow \exists w(Trigger(x,w,z) \wedge GoalOf(y,w)). \quad (90)$$

- *Improver*. For a given chunk of reality x if function f improves a realization r of a function f' , which is a performer function of x , then f is called an *improver* function of x and is denoted by $Fu_{improve}(f,x,r)$.

$$Fu_{improve}(x,y,z) \leftrightarrow \exists w(Improve(x,w,z) \wedge GoalOf(y,w)). \quad (91)$$

Classification Based on the Kind of Function Ascription

In the context of an item to which a function is ascribed it may be classified as one or more of the sorts below:

- *Actual*
- *Dispositional*
- *Intended: Designed, Required, User or Researched Function*

For example transporting people is a required, designed, and dispositional or actual function in the context of a car.

Classification Based on the Dynamics of Function Realizer

The dynamics of a function realizer is the criteria for distinguishing the following kinds of functions:

- *Passive Function*. A function f is passive in context of its realizer r and is denoted by $Fu_{PassR}(f,r)$ iff the function realizer r is passive.

$$Fu_{PassR}(x,y) \leftrightarrow R_{Pass}(y,x). \quad (92)$$

A function is absolutely passive iff all its realizers are passive.

$$Fu_{Pass}(x) \leftrightarrow \forall y(R(y,x) \rightarrow R_{Pass}(y,x)). \quad (93)$$

- *Dynamic Function.* Analogously, a function is active in context of a realizer r and is denoted by $Fu_{DynR}(f,r)$ iff r is a dynamic realizer of f .

$$Fu_{DynR}(x,y) \leftrightarrow R_{Dyn}(y,x). \quad (94)$$

A function is absolutely dynamic iff all its realizers are dynamic.

$$Fu_{Dyn}(x) \leftrightarrow \forall y(R(y,x) \rightarrow R_{Dyn}(y,x)). \quad (95)$$

Note that in our framework the passive or active character of the continuous or sequential function cannot be determined on the pure functional level but it is relative to the function's realization. Take as an example the function `to enable sitting`. A chair which is a typical realizer of this function is passive, thus the function is also considered to be passive. However, one can imagine an intelligent chair made of foam that actively adjusts the shape of the seat to the shape of the body of the person who seats on it. Clearly such a realizer will not be passive anymore. Thus, it is not the function `to enable sitting` which is passive, but particular realizers of that function. On the other hand, all instantaneous functions are regardless of their realizations considered to be passive functions.

7.4.3 Reconstruction of the Current Classifications

In the current section we will refer to the classification of functions, introduced by Keuneke [Keuneke, 1991] and formalized by Iwasaki and Chandrasekaran [Iwasaki, Chandrasekaran, 1992]. We will demonstrate how far this can be reconstructed and improved by means of the classifications presented above.

Keuneke recognized four types of functions: *toMake*, *toMaintain*, *toPrevent*, and *toControl*. *ToMake* is the basic function type; the aim of the *toMake* function is to achieve a desired state of the world. For example, the function of `locking the doors` is aimed to achieve a state in which the doors are locked. In terms of Iwasaki and Chandrasekaran [Iwasaki, Chandrasekaran, 1992] *toMake* is a function whose trajectory's final state satisfies a

desired goal⁶³. In terms of OF the *toMake* function type can be reconstructed by the sequential function which is realized by the process ending with the goal state, understood as a presential.

The second type of function introduced by Keuneke is *toPrevent*. A function is of the *toPrevent* type if during the realization of that function an undesired state does not take place. The trajectory achieves the *toPrevent* function if in none of the states of the trajectory the undesired state holds. *ToPrevent* can be represented in OF by means of a continuous function where the goal is expressed in the form of negation.

The third type in Keuneke's typology is *toMaintain*. In the *toMaintain* function type the desired state not only must be achieved, but it must be sustained over a given period of time. A trajectory *Tr* is said to achieve the *toMaintain* function, if the goal holds in all states of the trajectory *Tr*. We can see that the difference between *toMaintain* and *toMake* concerns in fact the time extents of their goals. In *toMake* the goal is not extended in time - preferably it is a presential, whereas in *toMaintain* a goal is time-extended. It is not, however, clear whether *toMaintain* is only about sustaining a desired state for a given period of time as the definition of Iwasaki and Chandrasekaran ([Iwasaki, Chandrasekaran, 1992], p. 10) suggests, or whether it also involves the achievement of a desired state as the following phrase of Keuneke suggests: "Maintenance devices require components and mechanisms that both achieve a state and make it persist." ([Keuneke, 1991], p. 24). This ambiguity can be properly treated in OF. In the first sense *toMaintain* can be reconstructed as a continuous function, whereas in the second as the combination of a sequential and a continuous functions.

According to Keuneke, the *ToMaintain(Not)* function should be distinguished from *toPrevent*: "ToPrevent functions provide short-term, fail-safe mechanisms, not operations for normal and continuous maintenance" ([Keuneke, 1991], p. 24), whereas the maintain function continuously sustains a desired state. Keuneke illustrates this difference with the example of the dikes in Holland, which keep the land dry and thus have the *toMaintain(NOT Flood)* function. In contrast, the function of the little boy who, in the case of a hole in a dike, puts a finger in it and rescues the land from the thread of a flood, is according to Keuneke *toPrevent(Flood)*. Although the boy prevents Holland from flood, he does it only temporarily, and therefore his function is not *toMaintain(NOT Flood)*. This example shows that *toPrevent* does not imply *toMaintain(NOT)*. According to Keuneke *toMaintain(NOT)* does not imply *toPrevent* either. She argues that the lamps in the building have the function *toMaintain(NOT Dark)*, but do not have the function *toPrevent(Dark)*.

⁶³ Framework of Functional Representation developed by Iwasaki and Chandrasekaran is discussed in section 2.1.2.

The distinction between *toPrevent* and *toMaintain(NOT)* can be reconstructed by references to the goal's time extent. In the *toPrevent* function the goal is a presential or relatively short process, whereas in *toMaintain(NOT)* the goal is a time extended process.

Keuneke's postulate that the *toPrevent* function does not imply *toMaintain(NOT)* holds because from the fact that a goal is fulfilled at one time boundary *b*, it does not follow that it is fulfilled throughout the whole chronoid whose time boundary is *b*. However, the second claim, saying that *toMaintain(NOT)* does not imply *toPrevent*, seems to be incorrect. If a given goal is fulfilled within the whole chronoid then it is fulfilled in every time boundary of that chronoid. In the case of lamps, if they maintain the light in a building during some period of time, it seems justified to say that they prevent the building from sinking into darkness at every moment of that period.

The last function type distinguished by Keuneke is *toControl*. This function differs from the above types since those were focused on achieving some state; while the control function reflects the power of regulation. Keuneke writes: "To control something or someone implies a direct multivalued relationship between the device's action and the resulting effects." ([Keuneke, 1991], p. 24). In OF there is no straightforward counterpart of the control function. Nevertheless, it can be reconstructed by a number of sequential functions. For example, the function to control the room temperature can be decomposed to two sequential functions.

F₁. Req: room temperature lower than the given value; Goal: enable function of heating; FI: heater enabler.

F₂. Req: room temperature higher than the given value; Goal: enable function of cooling; FI: cooling enabler.

The typology of Keuneke was modified by Kitamura and colleagues [Kitamura, Mizoguchi, 1998; Kitamura et al., 2002], where yet another type of function, named *toHold*, was introduced: "'ToHold' is used when no special effort is made for the goal. For example, the function of a pipe, 'to pass fluid through', is said to be 'ToHold.'" ([Kitamura et al., 2002], p. 151). It is not completely clear how the lack of effort should be understood, since even in the case of a pipe there is a force of the water pressure which a pipe must sustain. It seems, however, that the intuitions behind the *toHold* type are close to those underlying the notion of a passive function. A pipe remains passive in realizing its function. Therefore, we suggest interpreting the *toHold* function type as a passive function.

We see therefore that the classification of functions developed in the current section permits not only to reconstruct the well known classification of Keuneke together with its formal interpretations and modifications, but moreover permit to detect some of its limitations.

7.5 Architecture

Application of OF to ontological engineering not only requires the organization of functions into a taxonomic structure, but moreover we find it useful in providing a modularization of the introduced notions. In particular, as was stated in the requirements of OF (R 1.3.), functions should be modeled independently of their realizations. Several approaches reported in chapter 2 provide architectures supporting this requirement, e.g. FBS_{state}, MFM or FBS_{structure}.

In the current section we present a four-layered architecture for representing functional knowledge. The following layers are introduced:

1. The Pure Functional Layer.
2. The Impure Functional Layer.
3. The Realization Layer.
4. The Non-functional Layer.

Each of those layers provides an answer for a different type of functional and non-functional explanation. The pure functional layer and the impure functional layer answer *what* should be done (what is the goal), and *why* it should be done (what is the reason behind the goal). The non-functional layer answers the non-functional questions *what* is present and *how* it behaves. The realization layer mediates the functional layers with the non-functional layer and provides the answer for the question *how* a function is realized.

The distinguished layers permit to separate the functional from the non-functional knowledge. It is not only important due to the fact that the functional knowledge, in contrast to the non-functional one, is highly intentional and contextual, but it also reflects the fact that those types of knowledge are highly independent. In turn the realization layer, as a mediating one, is dependent on both the functional and the non-functional layer.

Figure 19 illustrates all the four layers by the example of the function `to transport goods`.

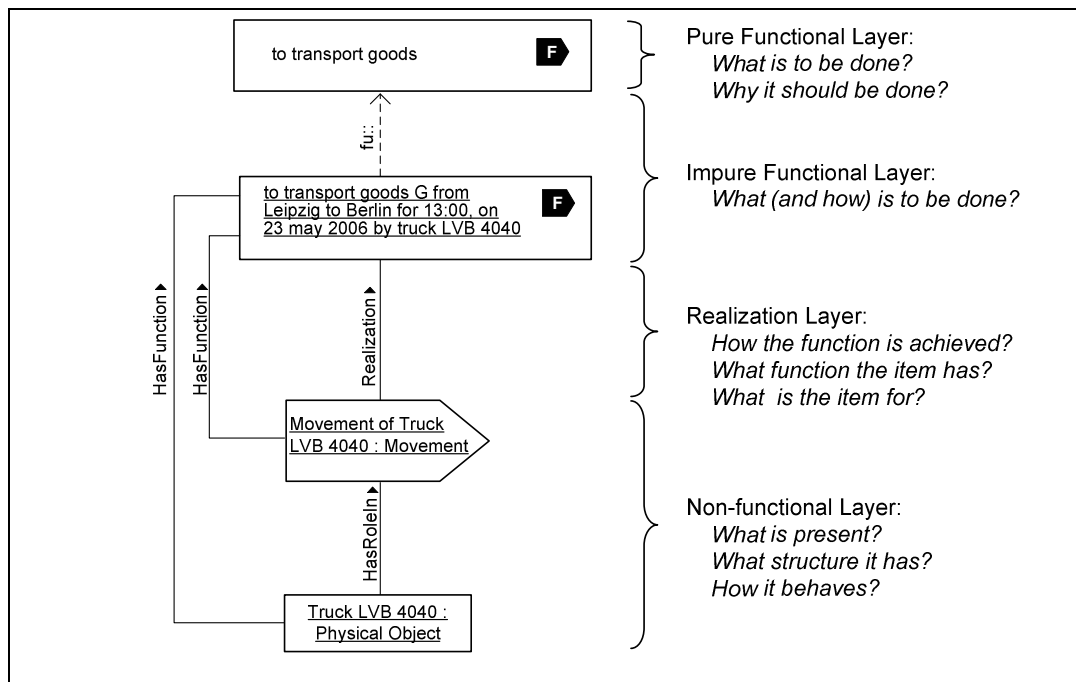


Figure 19. OF four-layered architecture presented on the exemplary function to transport goods.

7.5.1 Non-functional Layer

A vast part of human knowledge does not concern functions. Moreover, one can observe that a phenomenon described in terms of a function may be described from a non-functional perspective as well. For example, the process of goods transportation can be described in physical terms as the movement of bodies.

The non-functional knowledge is represented on the non-functional layer (NFL). On this layer are handled the explanations of what an object is and how it behaves. All categories of GFO used in the current work belong to that layer. The non-functional layer may belong to any strata of our knowledge – physical, social or cognitive. Depending on the needs, a domain or top-level ontology may be used on the non-functional layer. The non-functional layer is completely independent of the functional one, thus none of the categories used on this layer presupposes the categories of the functional layer.

7.5.2 Pure Functional Layer

The main purpose of the pure functional layer (PFL) is to represent the structure of functions and their interrelations independently of their particular realizations. It is the layer for modeling functions independently of their realizations. It is of particular importance e.g. in the early phases of design, when the particular means of realization are not taken into

consideration, but when of relevance are the goals. In this sense PFL also supports goals modeling. In addition, it is beneficial for representing purely teleological view of phenomenon.

$D(y,x)$	$Fu_{Enable}(x,y)$	$Part_{Fu}(x,y)$	$x \subset_{Fu} y$
$Enable(x,y)$	$Fu_{Instant}(x)$	$Prevent(x,y)$	$x \subseteq_{Fu} y$
$Exclude(v,w)$	$Fu_{MulGoal}(x)$	$REQ(x)$	$x \Rightarrow_{Fu} y$
$Fl(x,f)$	$Fu_{Neutral}(x,y)$	$Req(x,y)$	$x ::_{Fl} y$
$Fl_{Comp}(x,y)$	$Fu_{Perform}(x,y)$	$Req_{Env}(x,y)$	$x ::_{Gl} y$
$Fl_{Ind}(x,y)$	$Fu_{Prevent}(x,y)$	$Req_F(x,y)$	$x ::_{Req} y$
$FITEM(x)$	$Fu_{Seq}(x)$	$Req_{Op}(x,y)$	$x =_{Fi} y$
$FSt(x,y)$	$Fu_{Support}(x,y)$	$SideEf(x,y)$	$x =_{Fu} y$
$Fu(x)$	$GOAL(x)$	$Support(x,y)$	$x =_{Gl} y$
$Fu_{Accomp}(x)$	$Goal(x,y,z)$	$TFRAM(x)$	$x =_{Req} y$
$Fu_{Basic}(x)$	$GoalFor(x,y)$	$UniD_{Ab}(x,y)$	
$Fu_{Coh}(x)$	$GoalOf(u,y)$	$UniFu(x)$	
$Fu_{Comp}(x)$	$Intent(q,v)$	$UniFu_{Ab}(x)$	
$Fu_{Contin}(x)$	$IntCont(i,R,a_1...a_n)$	$UniFu_{Prim}(x)$	

Table 5. The categories of OF belonging to the Pure Functional Layer.

Although PFL enables the realization-independent functional modeling, PFL is not completely independent of the non-functional layer. The categories of NFL provide the vocabulary for the description of functions in PFL. It is clear since one cannot speak about functions and goals when one lacks the vocabulary to describe the world. For example, to model the biological function of a protein to copy a chromosome one needs to have the notion of chromosome introduced. In this sense PFL is built above NFL.

The functional layer belongs not to the physical but to the cognitive or social strata as it involves subjective, intentional aspects of descriptions, such as goals and reasons behind them. The categories of OF belonging to that level are listed in table 5.

7.5.3 Realization Layer

The realization layer is the mediating layer between the pure functional layer (PFL) and the non-functional layer (NFL). It delivers the answer to the questions: (1) how a given function is realized, (2) what function is realized by a given entity and (3) what function an entity has ascribed (4) is the entity malfunctioning. The mediation between the non-functional and the functional layer is provided therefore by four relations: realization, instantiation between

realizer and universal functional item, has-function and malfunction. A full list of OF categories belonging to that layer is listed in table 6.

$Contribute(x,y,z)$	$Malfu(x,y,z)$	$R_{DispStr}(x,y)$	$UniHasFu(x,y,z)$
$Execute(x,y,z)$	$Malfu_{His}(x,f,s)$	$R_{Dyn}(x,y)$	$UniHasFu_{Act}(x,y,z)$
$HasFu_{Act}(x,y,z)$	$Malfu_{Inten}(x,f,c)$	$RI_{Act}(x,y)$	$UniHasFu_{Disp}(x,y,z)$
$HasFu_{Desig}(x,y,z)$	$Malfu_{Kind}(x,f,s)$	$RI_{ActCulm}(x,y)$	$UniHasFu_{Inten}(x,y,z)$
$HasFu_{Disp}(x,y,z)$	$Means_{ActRI}(x,y)$	$RI_{ActMin}(x,y)$	$UniR(u,f)$
$HasFu_{Inten}(x,y,z)$	$R(x,y)$	$RI_{ActNonCulm}(x,y)$	$UniRI_{Min}(x,y)$
$HasFu_{Req}(x,y,z)$	$R_{Act}(x,y)$	$RI_{ActSit}(x,y)$	$x @ y$
$HasFu_{Res}(x,y,z)$	$R_{ComplAct}(x,y)$	$RI_{Disp}(x,y)$	
$HasFu_{User}(x,y,z)$	$R_{Disp}(x,y)$	$R_{Pass}(x,y)$	

Table 6. The categories of OF belonging to the Realization Layer.

7.5.4 Impure Functional Layer

The demand of the functional modeling is to delimit the functional model, which provides the information *what* should be achieved from the behavioral, or structural model, which provide the information *how* it should be achieved. In practice, however, when modeling functions, some aspects of realization are often taken into account. Sometimes there are realizations that are the only good choices (or best practice) and sometimes there is no choice at all, since there is only one way of realization available. Moreover, the side effects of realizations and functions handling them are often included in the functional model. Those aspects should be covered by OF, but surely they should not be treated as pure functional models in order not to mix the order of *what* is achieved and *how* it is achieved.

Therefore, we introduce the impure functional layer, which is a sub-layer of the realization layer. In contrast to the realization layer, it is not concerned primarily with the realization of functions but it focuses on the same problem as the pure functional layer, namely - what is to be done. In contrast to the pure functional layer, it considers this question not independently of the realization but in the context of a particular realization and particular realizers.

For example, consider a logistics company which has a strong fleet of transporter cars. Considered on the purely functional level the company may have a primitive function to transport goods from Leipzig to Berlin for 13:00, 23 May 2006. However, due to the fact that there is a particular truck available, call it `truck LVB 4040`, it is reasonable to state that this individual truck is intended to be used for that purpose. In this

sense the business function of the company is an individual multiple-goal function to transport goods *G* from Leipzig to Berlin for 13:00, 23 May 2006 by truck LVB 4040.

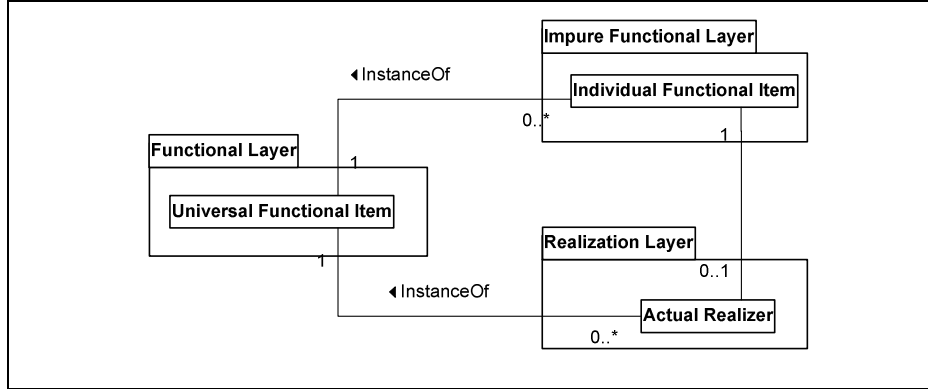


Figure 20. The UML diagram representing relations between Universal Functional Item, Individual Functional Item and Realizer. Classes represent reified relations of OF. Every class is embedded in the corresponding Layer.

The individual functional item resembles the individual actual realizer and the individual function resembles the individual realization. However, those notions should not be confused. An individual function is an intentional entity which may be realized or not, whereas the individual realization is an objective entity having the causal power of achieving the goal of the function. The relation between an individual function and an individual actual realization is not one-to-one but instead one-to-zero-or-one, which means that an individual function may be, but is not necessarily, actually realized by one individual entity. Analogically an individual functional item is distinct from an actual realizer. The former is a purely teleological entity while the second contains non-teleological structural description as well (see figure 20).

$Fu_{Dyn}(x)$	$Realize(x,y)$
$Fu_{DynR}(x,y)$	$Seq(y, L)$
$Fu_{Improve}(x,y,z)$	$SideEff_R(x,y,z)$
$Fu_{Pass}(x)$	$Trig(v,z)$
$Fu_{PassR}(x,y)$	$Trigger(x,y,z)$
$Fu_{trigger}(x,y,z)$	$x ::_{Fu} y$
$Improve(f,f',r)$	$x ::_{FI} y$
$IndFu(x)$	$x ::_{GI} y$
$Part_{Seq}(x,y)$	$x ::_{Req} y$

Table 7. The categories of OF belonging to the Impure Functional Layer.

Additionally, impure functional models enable to handle the realization-dependent relations between functions, the side effects of realizations as well as the external classifications and

typologies of functions. The categories that belong to the impure functional layer are listed in table 7.

7.6 Summary

The current chapter is oriented around the problem of the incorporation of the developed ontology of functions into a wider ontological framework, namely into GFO. This task, in our opinion, requires the identification of function characteristics, the analysis of some of the candidates for function definition against those characteristics as well as the organization of the developed notions into a taxonomic architecture. This led us to the definition of function and enabled to plug the notion of function in an appropriate place in the taxonomy of the top-level entities of GFO ontology.

We proposed to understand functions as subjective, contextual and goal-oriented intentional entities. In addition, we discussed the limitations of treating functions as processes or goals. On the basis of the notions developed in the previous chapters several classifications of functions have been developed which, if organized in a combinatorial manner, provide a general taxonomy of forty basic function types and an additional twenty four realization-dependent ones.

Finally, the notions of OF have been organized into the four-layered architecture which permits to keep separately the functional model from the model of the realization of function. In addition, the modular architecture of OF permits to extend non-functional ontologies with functional notions without significant changes to them.

8 A UML Profile for Functional Modeling founded on OF

8.1 Introduction

In the present chapter we outline the UML profile for functional modeling based on the ontology developed in chapters 3 to 7. The Unified Modeling Language (UML) is a powerful and widely accepted tool for software engineering and conceptual modeling. Ontologies, on the other hand, have been getting an increasing impact in recent years in a number of application areas. Several attempts have been made to integrate UML and ontologies, which could be subsumed by two main generic scenarios. The first is to apply UML to ontological engineering, whereas the second is to improve UML with ontological analysis. Our intention is to follow both of those strategies: on the one hand we aim to apply UML to ontological engineering, but on the other we intend to provide an extension of UML founded on a developed ontology of functions, which would permit to represent functional knowledge.

8.2 UML and Ontology Engineering

Many of the applications of UML are far beyond the area of object-oriented analysis, where it is the current de facto standard. For example, UML has been used in relational database design [Dermuth, Hussmann, 1999], data modeling [Gornik, 2003], business modeling [Eriksson, Penker, 2000], multi-agent systems [Cranefield et al., 2001; Wagner, 2002; Bergenti, Poggi, 2000], and knowledge based systems [Abdullah et al., 2004].

Moreover, there is recognized a long list of benefits that the adoption of UML may bring to ontology engineering [Cranefield, Purvis, 1999; Cranefield et al., 2001; Kogut et al., 2002]:

- The graphical notation of UML in comparison to text based logical languages, used as ontology representation languages, is easily comprehensible for a human user, also for domain experts who are not trained in logic. On the other hand there is no standard for visual representation of ontologies. UML in this sense could be treated as a graphical front-end of ontology representation languages.
- UML is based on many years of modeling experience.

- UML is an open standard commonly known and accepted both in industry and academic world. None of the techniques developed for support of ontology development has such a wide impact outside the research community.
- UML-oriented CASE tools are more accessible to software practitioners than current computer-aided ontology engineering tools coming from the research community.
- Real world industrial ontology-based systems need to interact with legacy enterprise systems, which often have existing UML models. One representation common for both systems would simplify the mediation between those systems.
- There is a large source of ontological knowledge already available in UML models of existing applications. Ontology extraction from existing UML diagrams can be simplified if UML is used also for ontology development.
- In ontology-driven systems UML used both for ontology modeling and software engineering allows to reduce the number of the needed modeling tools and techniques to one.
- UML supports a modular approach to ontology development. UML models can be changed easily due to the modular nature of object-oriented modeling.
- Many of the UML class diagram constructs can be directly mapped to traditional ontology representation languages, i.e. class, generalization, instantiation, package.

Two scenarios of applying UML to ontological engineering can be distinguished. The first involves direct application of UML to the development of ontologies. This line is followed, among others, by Cranefield, Purvis and colleagues who in [Cranefield, Purvis, 1999] proposed to apply UML combined with OCL as a formalism for ontology modeling and in [Cranefield et al., 2001] discussed the application of UML to ontology modeling for agent systems.

The second scenario involves attempts to make UML compatible with the ontology modeling languages, especially those applied to Semantic Web such as DAML or OWL, by extending UML metamodel. For example, [Baclawski et al., 2001] suggest using UML for developing and displaying complex DAML ontologies. For that purpose UML is extended to handle those elements of DAML which cannot be mapped straightforward by UML constructs, such as property and restriction. [Falkovych et al., 2003] in turn propose to use UML to overcome the ontology development bottleneck. For that purpose authors introduced a translation mechanism between UML and OWL and introduced an alternative to Baclawski's way of handling incompatibilities between UML and OWL.

This scenario is also assumed in the OMG request for proposals for the Ontology Definition Metamodel (ODM), which is aimed at supporting the development of ontologies

using UML modeling tools, the implementation of ontologies in the OWL and forward and reverse engineering for ontologies [OMG, 2003a]. The required ODM should provide: (1) a standard metamodel grounded in the Meta Object Facility; (2) a UML Profile defining a visual notation for depicting ontologies, and (3) mappings between the profile and metamodel, and between ODM and OWL DL. The request got response from both academia (e.g. [Djurić et al., 2004; Brockmans et al., 2004]) and industry (e.g. [IBM, Sandpiper, 2005]).

Despite the benefits UML brings to ontology engineering a number of problems from which it suffers have also been recognized. Among others it was pointed out by the Precise UML Group (pUML) [pUML, 2005] that UML lacks in formal semantics. On the other hand, as [Guizzardi et al., 2002a; Guizzardi et al., 2002b; Guizzardi et al., 2004a; Guizzardi et al., 2004b] observed, defining UML constructs only in terms of its mathematical semantics, although essential, is not sufficient to make UML a suitable ontology representation language. In addition it was proposed to provide ontological correctness and so called real-world semantics of UML constructs by means of upper-level ontology, in particular by means of General Ontology Language and Unified Foundational Ontology [Guizzardi, 2005].

From the above introductory remarks we see that UML not only brings benefits to ontological engineering but it also can be improved by means of ontologies, in principle top-level ontologies upon which UML models can be built. In the current study we try to combine both of those approaches. We pursue the idea that UML may be used as an ontology modeling language, but on the other hand we propose to extend UML by means of the developed top-level Ontology of Functions.

8.3 Objectives

As was recognized in section 2.2.2, UML has some limitations in representing functions and functional knowledge. In particular, it does not permit to model functional knowledge independently of the behavior realizing it. UML 2.0 is composed of two main views: structural and behavioral, and lacks an independent functional view. Although such a functional view is perhaps not required in the context of object-oriented modeling, it is however necessary if UML is supposed to be used as a general language for conceptual modeling applicable also in ontology engineering. Thus we propose to introduce to UML a third, functional view, based on the developed top-level ontology of functions. The proposed extension of UML would enable modeling of domain ontologies which require functional concepts.

8.4 Method

For extending UML with a functional view we use a UML-built-in extension mechanism and by its means we adopt the notions of OF into UML. UML comes with two mechanism of extension - a lightweight and a heavyweight approach ([OMG, 2004], p. 11):

- Lightweight approach - a new dialect of UML can be defined by using profiles to customize the language for particular platforms (e.g., J2EE/EJB, .NET/COM+) and domains (e.g., finance, telecommunications, aerospace).
- Heavyweight approach - a new language related to UML can be specified by reusing part of the InfrastructureLibrary.

Here we intend to develop a lightweight extension, by developing a profile for functional modeling. The UML profile is “a stereotyped package that contains model elements that have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. A profile may also specify model libraries on which it depends and the metamodel subset that it extends.” ([OMG, 2003b], p.14)

8.5 Overview of the Architecture

The architecture of the profile is founded on the OF architecture presented in section 7.5. It is composed of two profiles dependent on the Core Package: the Ontology Profile and the Function Ontology Profile. The former is the profile for representing concepts of a non-functional ontology and corresponds to the non-functional layer. Here, we refer to GFO but also other ontologies, in particular domain ontologies, may be represented in this profile as well. The Function Ontology Profile depends on the Ontology Profile and handles the concepts introduced in the Ontology of Functions (figure 21).

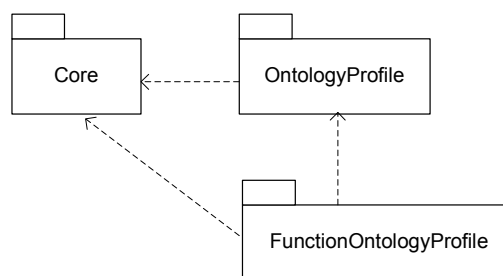


Figure 21. Dependencies between Profiles.

Function Ontology Profile contains three packages: the Functions package, the Function Ascriptions package and the Impure Functions which correspond to the functional layer, realization layer and impure functional layer, respectively (figure 22).

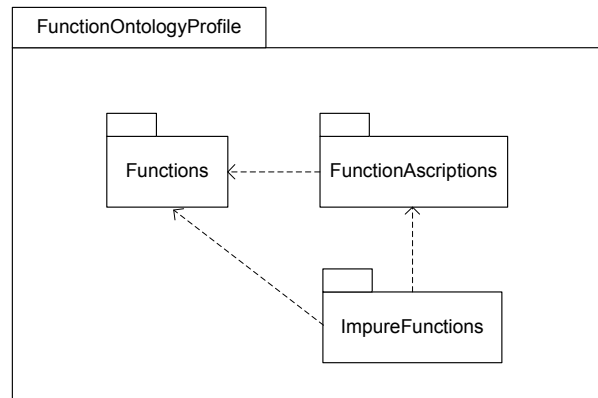


Figure 22. Dependencies within the Function Ontology Profile.

The Functions package supports modeling of the structures of functions and interrelations between functions independently of their realization. It contains two packages: Function Structures and Functional Relations (figure 23).

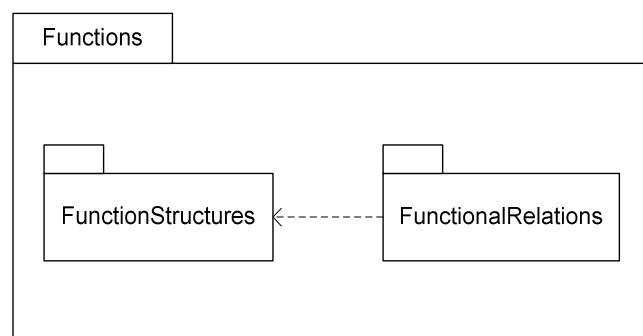


Figure 23. Dependencies within the Functions Package.

The Function Ascriptions package relates functions to the structure and behavior, which realize functions. Moreover, it provides means for modeling ascriptions of functions to the elements of the Ontology Profile.

The Impure Functions package is structured analogously to Functions package and contains the Impure Function Structures package and dependent on it the Impure Functional Relations package. The former permits to model structure of individually determined functions, whereas the latter permits to model interdependencies between functions, which depend on particular realizations, such as the triggering, sequence-part and improvement relations⁶⁴.

⁶⁴ For more detailed discussion on the layers of functional knowledge see section 7.5.

8.5.1 Ontology Profile

In the present section we outline the Ontology Profile. However, we do not provide the full definitions of the elements of the package. For our purposes it is enough to say that both relations and categories of GFO are introduced as stereotypes in the Ontology Profile. On the basis of them we define the stereotypes of the Functions, Function Ascriptions and Impure Functions packages. Below we give only a short informal specification of the selected stereotypes of the Ontology Profile which are used later in the specification of the Function Ontology Profile:

- Entity (from Ontology Profile) – a most general notion comprising all kinds of entities.
- Individual (from Ontology Profile) – an element of the model that refers to exactly one individual entity in the domain. Typically it is a UML object. However not all UML objects are individuals, e.g. `ape:species` is not an individual but a universal.
- Universal (from Ontology Profile) – an element of the model which refers to more than one individual in the domain.
- Complex Whole (from Ontology Profile) - it is a complex entity comprehended as a whole. In particular these are facts, configurations, configuroids, situations and situoids⁶⁵.
- Agent (from Ontology Profile) – proactive agentive entity capable of establishing goals.
- Role (from Ontology Profile) – an aspect of an Entity in some context.
- Part (from Ontology Profile) – general domain independent part-of relation.
- Proper Part (from Ontology Profile) - non-reflexive part-of relation.
- Process (from Ontology Profile) – an entity happening in time.

⁶⁵ Note that in the profile developed we do not introduce the GFO distinction between situoids and situations but integrate them under the notion of situation.

8.5.2 Functions Package

Function Structures

Class Diagram

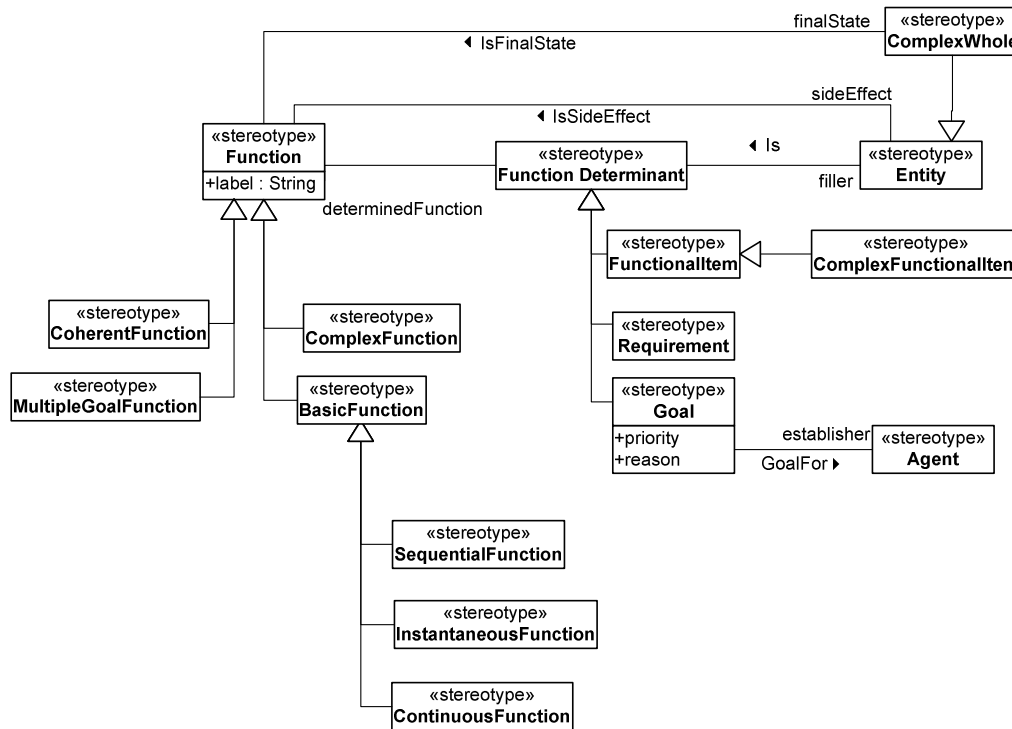








Figure 24. Class diagram of the Functions Package.

Class Descriptions

Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	None.
Semantics	A Function is an intentional entity defined in purely teleological terms by the specification of a Goal, a Requirement and a Functional Item which commonly is ascribed by means of the Has-Function relation to the entities that in some context are the Realizations of the Function, execute such a Realization or are intended by a reliable Agent to do it.
Specialization	{disjoint, complete} Basic Function, Complex Function.

	{disjoint, complete}	Coherent Function, Multiple-Goal Function.												
Attributes	label: String [1..*]	Provides the label of the Function. Typically, it is an expression of the form “to do something”.												
Associations	determinant: FunctionDeterminant [3...*]	References the entities which determine the Function.												
	sideEffect: Entity [0..*]	References the unintended effects of the Function. A Side Effect is an entity of an arbitrary kind, which is affected by the function realization but is not a part of the function goal. In particular Side Effects are all unintended entities existentially dependent on a Goal.												
	finalState: Entity[1..*]	References the Comprehensible Whole, which has a role of a Final State.												
Notations	<p>Function is represented as a rectangle marked with a black arrowhead with a white “F” inside and named by the label. Inside the rectangle in separated compartments are listed determinants: associated Goal (optionally (1) together with the Agent establishing it, the reason, and the priority (2) together with the Final State containing it), the Requirements, the Functional Item, and optionally Side Effects. Eventually, a function may be represented in compact form as a labeled rectangle with a black arrowhead with “F” attached.</p> <div><table><tr><td>Goods Transport</td><td></td></tr><tr><td colspan="2"><<req>> goods located in A</td></tr><tr><td colspan="2"><<goal>> goods located in B</td></tr><tr><td colspan="2"><<fi>> goods transporter</td></tr><tr><td colspan="2"><<se>> goods damaged</td></tr></table><table><tr><td>to transport goods</td><td></td></tr></table></div>		Goods Transport		<<req>> goods located in A		<<goal>> goods located in B		<<fi>> goods transporter		<<se>> goods damaged		to transport goods	
Goods Transport														
<<req>> goods located in A														
<<goal>> goods located in B														
<<fi>> goods transporter														
<<se>> goods damaged														
to transport goods														

Coherent Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Function (from Function Structures).
Semantics	A Function all of whose Final States compose a Coherent Entity.
Notations	Similarly to Functions, but the black arrowhead is marked with “CohF”.

Multiple-Goal Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Function (from Function Structures).
Semantics	A Function whose Final States do not compose a single Coherent Entity.
Notations	Similar to Functions, but the black arrowhead is marked with “MultF”.

Basic Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Function (from Function Structures).
Specialization	{disjoint, incomplete} Sequential Function, Instantaneous Function, Continuous Function.
Semantics	Basic Function is a Function having a single Final State being a Fact.
Constraints	Shall have assigned exactly one Final State. The Final State is single Fact.
Notations	Similar to Functions, but the black arrowhead is marked with “BscF”.

Complex Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Function (from Function Structures).
Semantics	A non-basic Function.
Notations	Similar to Functions but the black arrowhead is marked with “CmplxF”.

Sequential Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Basic Function (from Function Structures).
Semantics	A Basic Function whose Requirements and the Goal are Presentials and the Requirements are intended to be present before the Goal.
Notations	Similarly to Functions, but the black arrowhead is marked with “SeqF”.

Instantaneous Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Basic Function (from Function Structures).

Semantics	A Basic Function whose Requirements and the Goal are Presentials located at the same or at coincident Time Boundaries.
Notations	Similar to Functions but the black arrowhead is marked with “InstF”.

Continuous Function (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).
Generalizations	Basic Function (from Function Structures).
Semantics	A Basic Function whose Requirements and the Goal are Processes having the common starts and endings.
Notations	Similar to Functions but the black arrowhead is marked with “ContF”.

Function Determinant (from Function Structures)

General information	A stereotype extending Classifier (from Kernel).	
Generalizations	None.	
Semantics	Function Determinant points to the Entity which determines the structure of the Function.	
Associations	determinedFunction: Function [1]	References the Function which is determined.
	filler: Entity[1]	References the Entity which determines the Function.
Notations	Represented by a group of compartments within the function rectangle.	

Goal (from Function Structures)

General information	A stereotype of Classifier (from Kernel).	
Generalizations	Function Determinant (from Function Structures).	
Semantics	A Goal points to an Entity, which is intended to be affected by the Function. Every Goal is intended by some Agent for some Reason and has a Priority.	
Attributes	reason: String [1..*]	Provides the justification of a Goal.
	priority: Integer [1]	Defines a priority of a goal; based on the reason and the reliability of an Agent in a given community.
Associations	establisher: Agent [1..*]	References the Agent who establishes a Goal.
Notations	A compartment within the function rectangle labeled with “Goal”.	

Requirement (from Function Structures)

General information	A stereotype of Classifier (from Kernel).
Generalizations	Function Determinant (from Function Structures).
Semantics	Requirement point to the Entity which is intended to be present if the Function is to be realized.
Notations	A compartment within the function rectangle labeled with “Req”.

Functional Item (from Function Structures)

General information	A stereotype of Classifier (from Kernel).
Generalizations	Function Determinant (from Function Structures)
Semantics	A Functional Item of the Function f indicates a role of entities executing a realization of f , such that all restrictions on Realizations imposed by the Functional Item are dictated also by some Goal of f .
Constraints	The filler is restricted to Role
Notations	A compartment within the function rectangle labeled with “FI”.

Additional externally defined stereotypes are used in the package:

- Entity (from Ontology Profile).
- Complex Whole (from Ontology Profile).
- Agent (from Ontology Profile).

8.5.3 Functional Relations Package

Class Diagram

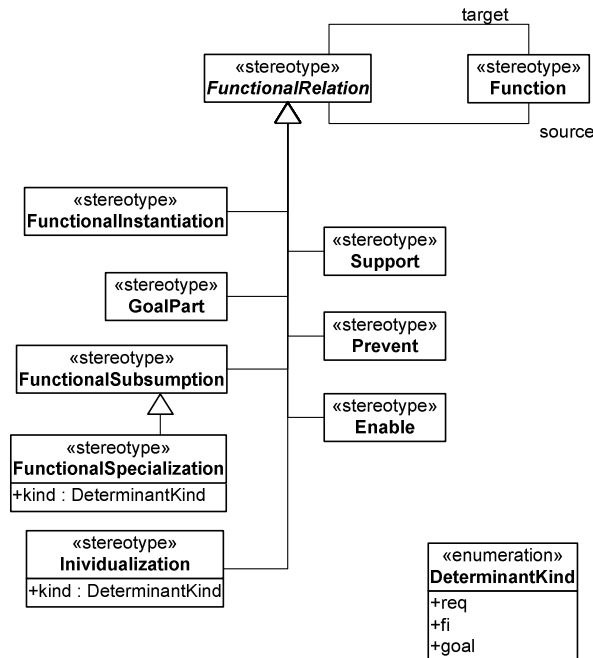


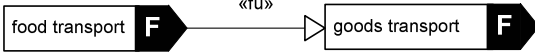
Figure 25. Class diagram of the Functional Relations Package.

Class Descriptions

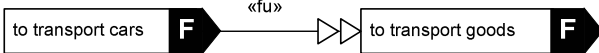
Functional Relation (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	None.	
Semantics	Functional Relation is an abstract class for all kinds of relations introduced in the Profile holding between two functions.	
Associations	source: Function [1]	Specifies the source Function of the Functional Relation
	target: Function [1]	Specifies the target Function of the Functional Relation


Functional Subsumption (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Functional Relation (from Functional Relations).	
Semantics	Functional Subsumption relates two Functions: a super-function and a sub-function, so we say that the sub-function <i>is a</i> super-function. A Function f subsumes a Function f' iff all Determinants of f subsume appropriate Determinants of f' .	
Associations	source: Function [1]	References the sub-function in the Functional Subsumption relation.
	target: Function [1]	References the super-function in the Functional Subsumption relation.
Notations	<p>A line between two Functions labeled with «fu» with a hollow triangle as an arrowhead pointing to the superordinate function.</p> 	

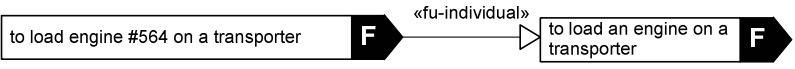
Functional Specialization (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Functional Subsumption (from Functional Relations).	
Semantics	It is the non-reflexive variant of the Functional Subsumption. A Function f specializes a Function f' iff all Determinants of f' subsume the appropriate Determinants of f and at least one Determinant of f' specializes the appropriate Determinant of f .	
Associations	kind: DeterminantKind [1..3]	Functional Specialization can be classified on the basis of the specialized determinants. The following flavors of Specialization are introduced: Requirement Specialization (req), Goal Specialization (goal), Functional Item Specialization (fi).
	target: Function[1]	References the general Function in the Functional Subsumption relation.
	source: Function [1]	References the specializing Function in the Functional Specialization relation.
Notations	<p>A line labeled with «fu» with a double hollow triangle as an arrowhead between two functions (optionally with listed kinds).</p> 	

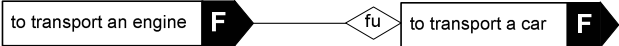
Functional Instantiation (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	<ul style="list-style-type: none"> - GFO Instantiation (from Ontology Profile). - Functional Relation (from Functional Relations). 	
Semantics	An individual Function f instantiates a universal Function f' iff individual Determinants of f instantiate the corresponding universal Determinants of f' .	
Associations	target: Function [1]	References the instantiated Function in the Functional Instantiation relation.
	source: Function [1]	References the instantiating Function in the Functional Instantiation relation.
Notations	<p>A dashed arrow labeled with <<fu::>> between two functions.</p> 	

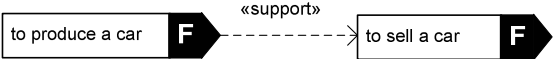
Functional Individualization (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Functional Relation (from Functional Relations).	
Semantics	A universal Function f is an Individualization of a universal Function f' iff at least one of the individual Determinants of f instantiates an appropriate universal Determinant of f' and the remaining Determinants of f are equal to the corresponding Determinants of f' .	
Attributes	kind: DeterminantKind[1..3]	The Functional Individualization can be classified on the basis of the instantiated Determinants. The following flavors of Functional Instantiation are distinguished: Requirement Instantiation (req), Goal Instantiation (goal), Functional Item Specialization (fi).
Associations	target: Function [1]	References the individualized Function in the Functional Instantiation relation.
	source: Function [1]	References the individualizing Function in the Functional Instantiation relation.
Notations	<p>A line between two functions with a hollow triangle as an arrowhead labeled with <<fu-individual>> (optionally with listed kinds).</p> 	

Function Part (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	<ul style="list-style-type: none"> - Functional Relation (from Functional Relations). - Part-Of (from Ontology Profile). 	
Semantics	Function f is a Function Part of a Function f' iff the Requirements of f are part of the Requirements of f' and the Goal of f is a part of the Goal of f' .	
Associations	target: Function [1]	References the Function which is a whole in the Function Part relation.
	source: Function [1]	References the Function which is a part in the Function Part relation.
Notations	<p>A line between two functions ended with a diamond labeled with fu at the function being the whole of the Function Part relation.</p> 	

Support (from Functional Relations)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Functional Relation (from Functional Relations).	
Semantics	A Function f supports a Function f' iff a Goal of f is a Proper Part of the Requirements of f' .	
Associations	target: Function [1]	References the Function which is supported in the Support relation.
	source: Function [1]	References the Function which supports in the Support relation.
Notations	<p>A dashed arrow between two functions labeled with <<support>>.</p> 	

Prevent (from Functional Relations)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Functional Relation (from Functional Relations).	
Semantics	A Function f prevents a Function f' iff the Goal of f excludes a Part of the Requirements of f' .	
Associations	target: Function [1]	References the Function which is prevented in the

		Prevent relation
	source: Function [1]	References the Function which prevents in the Prevent relation.
Notations	A dashed arrow between two functions labeled with <<prevent>>. <div style="text-align: center;"> </div>	

Enable (from Functional Relations)

General Information	A stereotype extending Directed Relationship (from Kernel)	
Generalizations	Functional Relation (from Functional Relations)	
Semantics	A Function f enables a Function f' iff the Requirements of f' are Part of the Goal of f .	
Associations	target: Function [1]	References the Function which is enabled in the Enable relation
	source: Function [1]	References the function which enables in the Enable relation.
Notations	A dashed arrow between two functions labeled with <<enable>>. <div style="text-align: center;"> </div>	

Determinant Kind (from Functional Relations)

Semantics	It is the enumeration class that defines literals to determine the kind of Functional Specialization and Individualization with regards to the kind of the determinant. The following kinds are distinguished: <ul style="list-style-type: none"> - Functional specialization/ Individualization with respect to requirements - Functional specialization/ Individualization with respect to the goal - Functional specialization/ Individualization with respect to the functional item
------------------	---

Additional externally defined stereotypes are used in the package:

- Function (from Function Structures).
- Part (from Ontology Profile).
- Proper Part (from Ontology Profile).

8.5.4 Function Ascriptions Package

Realization

Class Diagram

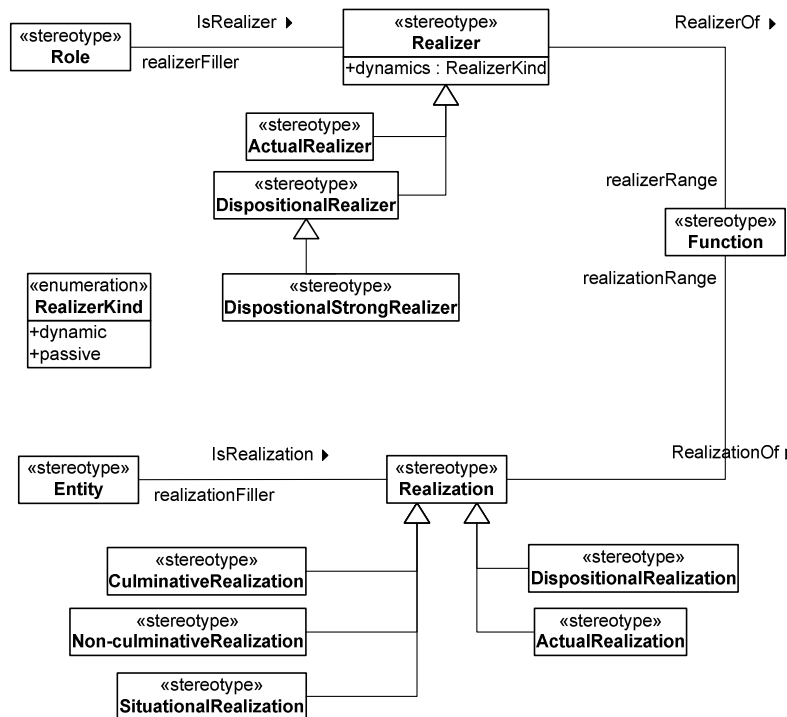


Figure 26. Class diagrams of the Function Ascriptions Package.

Class Descriptions

Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel)	
Generalizations	None.	
Semantics	It is a class for all types of Realizations introduced in the profile ⁶⁶ .	
Associations	realizationRange: Function [1]	References the Function which is realized.
	realizationFiller: Entity [1]	References the Entity realizing a Function.

⁶⁶ Comprises both individual and universal realizations. In analogous way realizer and has-function are introduced. All those relations pertain primarily to individuals and indirectly to the universals instantiated.

Notations	<p>An arrow labeled with <<realization>> between an Entity and a Function.</p>
------------------	--

Actual Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realization (from Realizations).
Semantics	Actual Realization references an Entity which fulfills the Requirements and the Goal of the function and provides the additional cause for the Goal being fulfilled.
Notations	Similar to Realization but labeled with <<act-realization>>.

Culminative Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realization (from Realizations).
Semantics	References a culminative Process realizing a Sequential Function.
Notations	An arrow labeled with <<culmin-realization>> between a culminative Process and a Sequential Function.

Non-culminative Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realization (from Realizations).
Semantics	References a non-culminative Process realizing a Continuous Function.
Notations	A labeled with <<non-culmin-realization>> arrow between a non-culminative Process and a Continuous Function.

Situational Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realization (from Realizations).
Semantics	References a Situation realizing an Instantaneous Function.

Notations	A labeled with <<sit-realization>> arrow between a Situation and an Instantaneous Function.
------------------	---

Dispositional Realization (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel)
Generalizations	Realization (from Realizations)
Semantics	References an Entity which has a disposition to be an Actual Realization of a Function.
Notations	A labeled with <<disp-realization>> arrow between an Entity and a Function.

Realizer (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel)	
Generalizations	None	
Semantics	It is a class for all types of Realizers introduced in the Profile.	
Attributes	dynamics: RealizerKind[1]	Specifies the kind of the Realizer with respect to its dynamics.
Associations	realizerRange: Function [1]	References the Function which is realized.
	realizerFiller: Role[1]	References the Role being a Realizer.
Notations	<p>An arrow labeled with <<realizer>> between a Role and a Function.</p>	

Dispositional Realizer (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realization (from Realizations).
Semantics	It is an Entity relevantly similar to the Actual Realizer. Relevant similarity means that it is structurally similar to an Actual Realizer in all aspects relevant for function Realization.
Notations	A labeled with <<disp-realizer>> arrow between a Role and a Function.

Dispositional Strong Realizer (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Dispositional Realization (from Realizations).
Semantics	A Dispositional Realizer of a Function f which is involved into a Dispositional Realization of f is called a Dispositional Strong Realizer of f .
Notations	A labeled with <<disp-str-realizer>> arrow between a Role and a Function.

Actual Realizer (from Functional Realizations)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Realizer (from Realizations).
Semantics	A Role of an Entity executing the Realization r of a Function f is called an Actual Realizer of f .
Notations	A labeled with <<act-realizer>> arrow between a Role and a Function.

Realizer Kind (from Realizations)

Generalizations	None
Semantics	<p>It is the enumeration class that defines literals to determine the dynamics of realizers. The following two kind are distinguished:</p> <ul style="list-style-type: none"> - Passive - Dynamic <p>A passive realizer is such a realizer that does not undergo a significant change during the realization of a function. A realizer is active if it undergoes a change during the realization of a function.</p>

Additional externally defined stereotypes are used in the package:

- Individual (from Ontology Profile)
- Universal (from Ontology Profile)
- Function (from Function Structures)
- Entity (from Ontology Profile)
- Role (from Ontology Profile)
- Process (from Ontology Profile)
- Situation (from Ontology Profile)

Has-Functions

Class Diagram

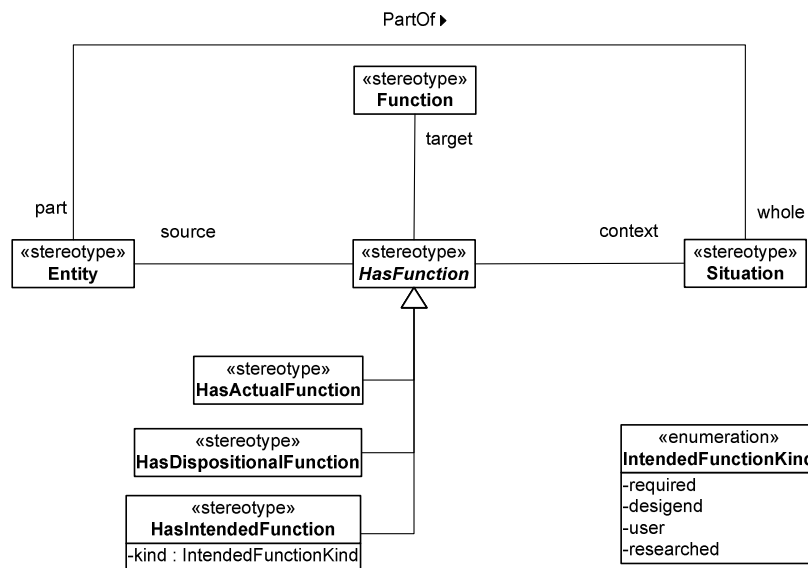
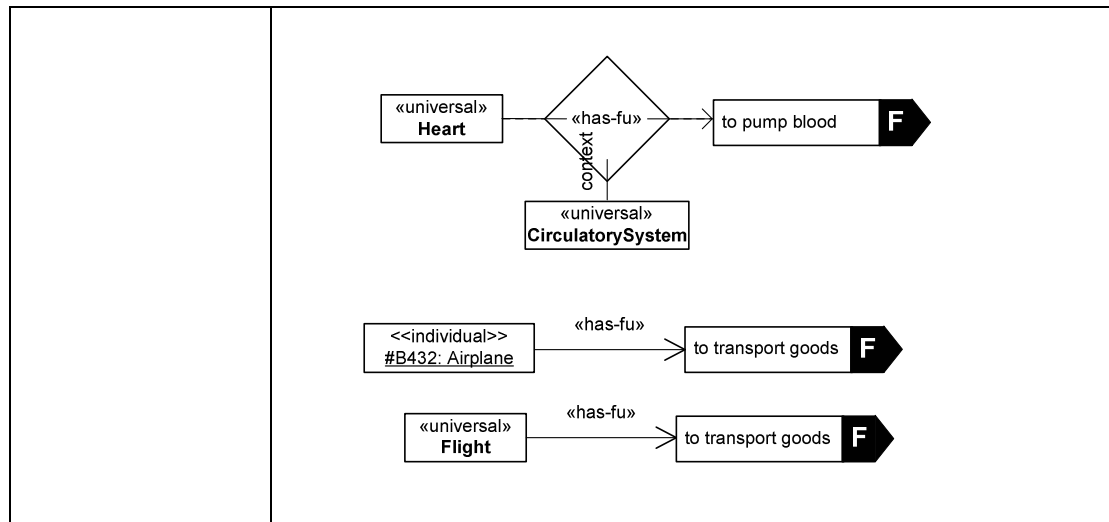


Figure 27. Class diagram of the Has-Functions Package.

Class Descriptions

Has-Function (from Has-Functions)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	None	
Semantics	A relationship comprising all types of assignments of a Function to an Entity in a Situation introduced in the Profile.	
Associations	target: Function[1]	References the Function which is ascribed in the Has-Function relation.
	source: Entity [1]	References the Entity to which a Function is ascribed.
	context: Situation [1]	References a Situation in context of which a Function is assigned to an Entity.
Notations	A diamond labeled with <<has-fu>> linked to an Entity, a Situation and a Function. The Situation is indicated by the keyword context and the Function - by the arrowhead pointing to it. Optionally, in cases where the context of the Has-Function is irrelevant it may be represented as an arrow labeled with <<has-fu>> between an Entity and a Function.	



Has-Actual-Function (from Has-Functions)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Has-Function (from Has-Functions).
Semantics	An Individual x has an Actual Function f in a Situation s iff x is an Actual Realization of f in s or x is a role-filler of an Actual Realizer r of f in s .
Notations	Similar to Has-Function but labeled with <<has-act-function>>.

Has-Dispositional-Function (from Has-Functions)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Has-Function (from Has-Functions).
Semantics	An Entity x has a Dispositional Function f in a Situation s iff x is a Dispositional Realization of f in s or an actor of a Dispositional Realizer of f in s .
Notations	Similar to Has-Function but labeled with <<has-disp-function>>.

Has-Intended-Function (from Functions)

General Information	A stereotype extending Directed-Relationship (from Kernel).
Generalizations	Has-Function (from Has-Functions).
Semantics	An Entity x has an Intended Function f in a situation s iff there is an Agent who intends x to have a Function f in s .
Attributes	kind: IntendedFunction Kind[1..4] Specifies the kind of Intended Function.
Notations	Similarly to Has-Function but labeled with <<has-int-function>>.

Intended-Function Kind (from Has-Functions)

Generalizations	None.
Semantics	<p>It is the enumeration class that defines literals to determine the kind of an intended function. The following kinds are distinguished:</p> <ul style="list-style-type: none"> - required - Has Required Function - designed - Has Designed Function - user - Has User Function - researched -Has Researched Function

Additional externally defined stereotypes are used in the package:

- Function (from Function Structures)
- Entity (from Ontology Profile)
- Situation (from Ontology Profile)

8.5.5 Malfunctions Package

Malfunctions are concerned with the lack of a dispositional or actual realization of a given function. An item is malfunctioning when a goal of the function should be reached by the item but is not.

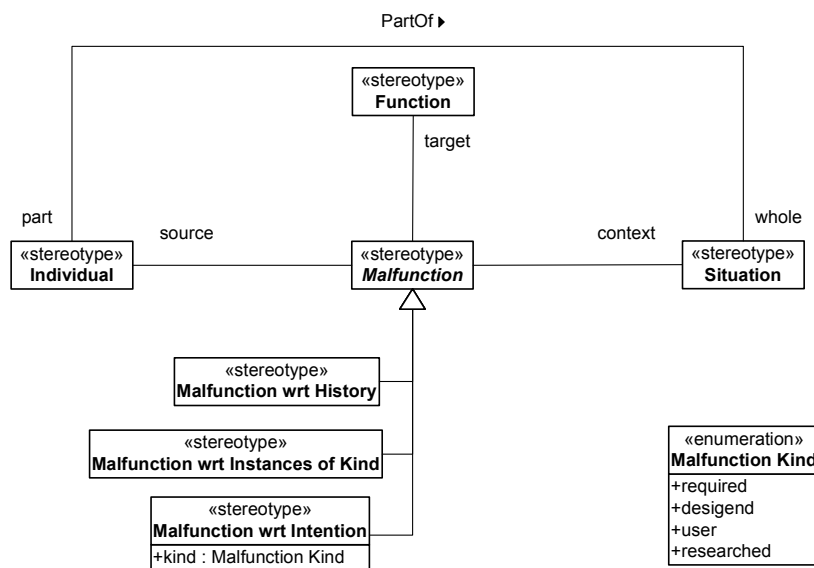
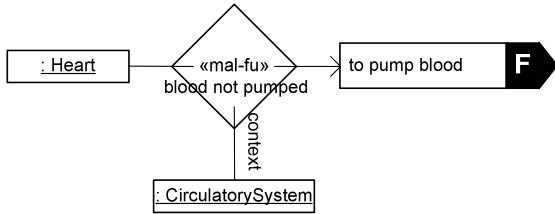
Class Diagram

Figure 28. Class diagram of the Malfunctions Package.

Class Descriptions**Malfunction (from Malfunctions)**

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	None.	
Semantics	An abstract class comprising all types of Malfunctions of an Individual in a Situation (context) introduced in the Profile.	
Associations	target: Function[1]	References the Function in the context of which the Malfunction is attributed.
	source: Individual [1]	References an Individual to which a Malfunction is attributed.
	context: Situation [1]	References a Situation in which an Entity malfunctions.
Notations	<p>A diamond stereotyped with <<mal-fu>> linked to an Individual, a Situation and a Function. The Situation is indicated by the keyword context, the Function - by the arrowhead pointing to it. Optionally, the diamond may be labeled with a description of the failure.</p> 	

Malfunction wrt. Intended-Function (from Malfunctions)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Malfunction (from Malfunctions).	
Semantics	<p>An Individual x malfunctions in a given context c with respect to the Intended Function f iff:</p> <ul style="list-style-type: none"> (a) x has Required/Designed Function f in c and (b) x does not have Dispositional or Actual Function f in c. 	
Attributes	kind: MalfunctionKind [1..2]	Specifies the kind of Malfunction.
Notations	Similar to Malfunction but stereotyped with <<int-mal-fu>>.	

Malfunction wrt. History (from Malfunctions)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Malfunction (from Malfunctions).
Semantics	An Individual x Malfunctions with respect to its History in a Situation s in the context of a given Function f iff x used to have a Dispositional or an Actual Function f in a Situation s' similar to s and does not have it in s .
Notations	Similarly to Malfunction but stereotyped with <<hist-mal-fu>>.

Malfunction wrt. other Instances of a Kind (from Malfunctions)

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Malfunction (from Malfunctions).
Semantics	An Individual x being an instance of a kind u Malfunctions in comparison with other Individuals of Kind u with respect to a Function f in a Situation s iff (a) other instances of u have a Dispositional or Actual Function f in the Situations similar to s , (b) x does not have a Dispositional or Actual Function f in s .
Notations	Similar to Malfunction but stereotyped with <<kind-malfunction>>.

Malfunction Kind (from Malfunctions)

Semantics	It is the enumeration class that defines literals to determine the kind of an intended malfunction. The following types are distinguished: <ul style="list-style-type: none"> - Malfunction with respect to required function - Malfunction with respect to designed function. - Malfunction with respect to user function. - Malfunction with respect to researched function. -
------------------	---

Additional externally defined stereotypes are used in the package:

- Individual (from Ontology Profile)
- Situation (from Ontology Profile)
- Function (from Function Structure)

8.5.6 Impure Function Structures Package

Class Diagram

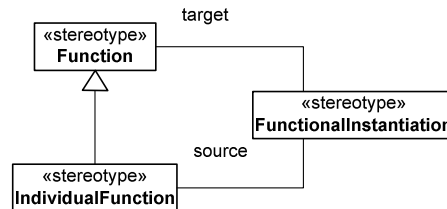


Figure 29. Class diagram of the Impure Function Structures Package.

Class Descriptions

Individual Function (from Impure Function Structures)

General information	A stereotype extending Object.
Generalizations	Function (from Function Structures).
Semantics	An individual instance of a Function. It is a Function defined in the context of a particular realization. All determinants of the Individual Function are Individuals.
Notations	<div style="display: flex; justify-content: space-between;"> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>to transport goods G from Leipzig to Berlin for 13:00, 23 may 2006 by truck LVB 4040</p> <p style="text-align: right;">F</p> <hr/> <p><<req>> goods G located in Leipzig at 10:00, 23 may 2006 AND truck LVB 4040 available at 10:00 - 13:00, 23 may 2006</p> <hr/> <p><<goal>> goods G located in Berlin at 13:00, 23 may 2006</p> <hr/> <p><<fi>> truck LVB 4040 qua goods G transporter</p> </div> <div style="border: 1px solid black; padding: 5px; width: 45%;"> <p>to transport goods G from Leipzig to Berlin for 13:00, 23 may 2006 by truck LVB 4040 : to transport goods</p> <p style="text-align: right;">F</p> </div> </div>

Additional externally defined stereotypes are used in the package:

- Functional Instantiation (from Function Structure).
- Function (from Function Structure).

8.5.7 Impure Functional Relations Package

Class Diagram

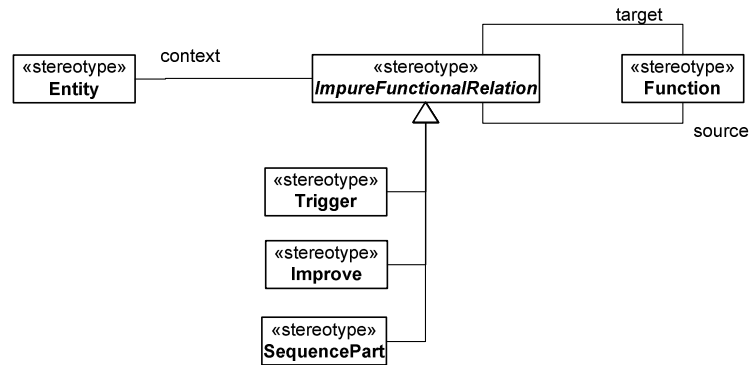


Figure 30. Class diagram of the Impure Relations Package.

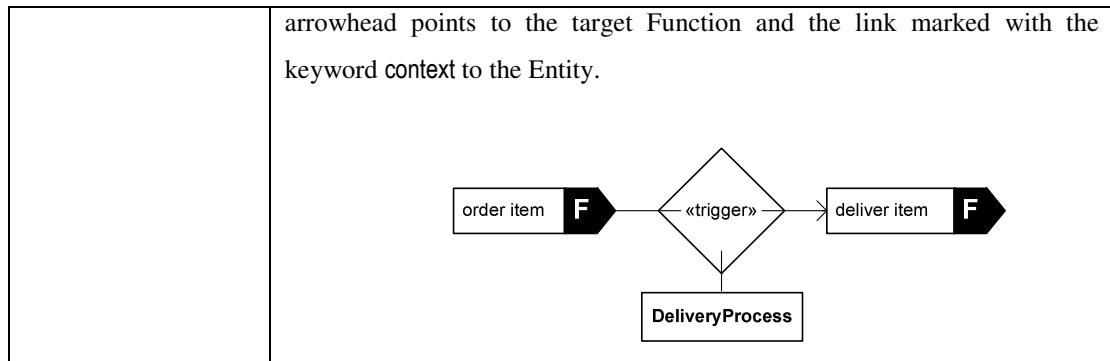
Class Descriptions

Impure Functional Relation (from Impure Functional Relations)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	None.	
Semantics	It is a grouping of all impure functional relations introduced.	
Associations	source: Function [1]	References the Function which is the source in the Impure Functional Relation
	target: Function [1]	References the Function which is the target in the Impure Functional Relation.
	context: Entity [1]	References the Entity which is the Realization of the Target Function in context of which the Impure Functional Relation holds.

Trigger (from Impure Functional Relations)

General Information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	Impure Functional Relation (from Impure Functional Relations).	
Semantics	A Function f triggers a Function f' in a Realization r iff the Goal of f is a Trigger of the Realization r of the Function f'	
Notations	A diamond stereotyped with <<trigger>> linked to a source and target Function and an Entity being a Realization of a target Function. The	

**Improve (from Impure Relations)**

General Information	A stereotype extending Directed Relationship (from Kernel).
Generalizations	Impure Functional Relation (from Impure Functional Relations).
Semantics	A Function f improves a given Realization r of a Function f' iff the Realization of f neutralizes some Side Effect of the Realization r of f' .
Notations	Similar to Trigger but labeled with <<improve>>.

Sequence Part (from Functional Relations)

General information	A stereotype extending Directed Relationship (from Kernel).	
Generalizations	<ul style="list-style-type: none"> - Impure Functional Relation (from Impure Functional Relations). - Part-Of (from Ontology Profile). 	
Semantics	A Function f is a Sequence Part of a Function f' iff f is an element of the sequence of Functions realizing the Function f' .	
Associations	target: Function [1]	References the Function which is realized by the sequence of Functions.
	source: Function [1]	References the Function which is an element of the sequence of Functions realizing the whole-function.
Notations	<p>If the context is not required - a line between two functions ended with a diamond labeled with seq at the function being the whole of the Sequence Part relation. If the context is required to be present then Sequence Part can be presented similarly to Trigger but labeled with <<seq-part>>.</p> <pre> graph LR A[to load an engine on a transporter F] -- seq --> B[to transport an engine F] </pre>	

Additional externally defined stereotypes are used in the package:

- Function (from Function Structures)
- Realization (from Realizations)
- Entity (from Ontology Profile)
- PartOf (from Ontology Profile)

8.6 Discussion and Conclusions

After presenting the developed profile let us reconsider the existing UML elements which may be used to model functions and compare them to the profile developed. In particular we will concentrate on use case diagrams which are often suggested to represent the functionality of a system, see e.g. ([Rumbaugh et al., 1999], p. 488).

On the first glance functions in OF resemble use cases. Firstly, there are a number of structural similarities: Firstly, the name of a use case resembles the label of a function. Secondly, a use case is related to the behavior realizing it (by `ownedBehavior` relation), which resembles the realization relation between function and the entity realizing it (e.g. a process) in OF. Thirdly, use cases may be specified in terms of pre- and post-conditions, which resemble requirements and goals of functions. Fourthly, use cases are assigned to actors representing the role of the external to the system entity involved in the use case. The association of the use case to an actor “describes how an instance of the classifier realizing the use case and a user playing one of the roles of the actor interact” ([OMG, 2005], p. 579). In this sense an actor resembles the functional item. In addition, use cases are organized into a generalization/specialization tree and can be glued by `<<include>>` and `<<extend>>` relations, which resemble the part-of relations between functions.

The above, then, suggests treating use cases as functions and the redundancy of the additional profile for functional modeling. However, in our opinion the above similarities are only superficial. The first strict difference concerns the essence of the use cases which are behavioral classifiers, such that “each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors” ([OMG, 2004], p. 578). From this follows that if use cases are to be considered to represent functions, then in fact they are restricted only to behavioral functions, i.e. functions realized by some behavior, i.e. a process. The relation of `ownedBehavior` confirms this understanding of the use case. In OF, on the other hand, accordingly with the principles adopted in functional device representation, we consider functions to be independent of behavior. Moreover, as reported in

section 3.5, there are functions not related to behavior, i.e. instantaneous functions. Thus, the notion of function in OF is broader than the notion of function based on a use case.

Concerning the similarities in structure between use cases and OF functions we admit that functions, analogously to use cases, are defined in terms of preconditions and postconditions, called requirements and goals in OF. However, in contrast to pre- and postconditions, which in fact are not the elements of UML meta-model, requirements and goals in OF are ontologically analyzed and formally defined. Concerning the similarity of the notion of actor and the notion of functional item we claim that they cannot be identified. Firstly, an actor represents the role of the external entity in interaction with a system, not with a single use case. For example, consider the typical actor `customer`, which is a role of a person in the context of a whole system. `Customer` can be assigned to various use cases, e.g. `view item`, or `purchase item`. In the interaction with each of those use cases `customer` has a different role, i.e. `observer`, `purchaser`. Those roles are not dependent on the actor as such but on the use case, hence the same role may be shared by the different actors. For example, `system administrator` may also be assigned to `view item` use case and have a role of an `observer` in the context of that use case. As reported in [Irwin, Turk, 2005], those more precise roles are not represented in use cases, but are represented in OF as functional items.

In addition, it seems problematic in use case diagrams to distinguish a mere participant of a use case from an actor, being an executor of it. For example, the use case `update customer data` may be defined in such a way that it requires the presence of a customer but is executed by a bank employee. This again can be grasped by the notions of OF – here in particular by the distinction between the realizer and the means of realization.

The relations between functions introduced in OF exceed the subsumption and part-of relations and thus provide the richer framework for modeling functions than the relations present in use case diagrams. Finally, the issues of function ascription and malfunction ascription crucial in many domains are out of the scope of use case diagrams, but are handled in OF.

Concluding we can say that functions in our understanding and the UML profile developed on their basis differ significantly from use case diagrams and provide a richer formalism for modeling functions. In addition, the understanding of a use case as a behavioral classifier prevents us from considering functions as an extension of use case, but more generally as an extension of classifier.

In addition, it should be mentioned that, for the purpose of modeling functions with processual realizations only, the use case diagrams can be enhanced with some of the expressiveness provided above, such as functional relations.

The profile is intended to be used just as it was discussed in the introductory section for the domain ontology development. In the first place in [Burek et al., 2006] we intend to use it for modeling functions within Open Biological Ontologies.

9 Conclusions

9.1 Problem

The current work is concerned with the development of a top-level ontology of functions and the incorporation of it into a wider ontological framework as well as providing the means for functional modeling. It was recognized that the notion of function is a domain-independent notion and is important in a high number of application areas. However, in our opinion there is a lack of a general, domain-independent treatment of this notion. The current approaches to function representation are mainly domain-oriented and the available top-level ontologies lack the notion of function or treat it scantily.

9.2 Solution

In the current work we have developed a formal, top-level ontology of functions incorporated in the wider ontological framework of GFO. The developed ontology covers four basic problem areas:

1. The representation of functions and functional relations independently of their realizations.
2. The determination of function realizations.
3. The assignment of functions to entities by the has-function relation.
4. The determination of an ontological status of function and the incorporation of the ontology of functions into the top-level ontology of GFO.

In addition, on the basis of the ontology developed the ready-to-use modeling framework has been proposed.

9.3 Advantages

Concerning the first problem area in the developed ontology functions are represented by so called functional structure, which brings several profits to functional representation. The advantages are discussed in the order of the requirements for the ontology of functions postulated in section 2.4.

- A function structure comprises the natural language label and the formally defined determinants. The label permits multiple natural language description that makes functions easily comprehensible for human users (ref. R.1.1). On the other hand, the label is only one of the components of functional representation, thus functional representation is not limited to ambiguous natural language description but may be precisely given by function determinants – requirements, goal, and functional item.
- The requirements together with the goal can be interpreted as an input-output pair, which makes the framework compatible with input-output representations (ref R.1.2.). On the other hand, it extends them, since functions are not reduced in our approach only to the pair of an initial state and goal, but a functional item is also included. In addition, all those notions are ontologically founded in GFO.
- The OF architecture permits both to represent functions independently from their realizations (on the functional level) and to represent entities involved in the realization independently from functions (on the non-functional level) (ref. R.1.3.1, R.1.3.2). The realization layer mediates those two independent layers.
- Functions are defined neither in behavioral nor in processual terms, which enables to handle not only behavioral functions but also static, structural functions (ref. R.1.4). Three basic kinds of functions permit to handle processual and non-processual functions. Sequential functions handle the cases in which the goal of the function is the culmination of the realization process. In contrast, continuous functions permit to handle functions maintaining the goal for a given period of time. Finally, instantaneous functions handle static and instantaneous realizations of functions like the functions realized by structures. Moreover, the analysis of the dynamics of a realizer permit to distinguish processual passive from processual active functions.
- Function is not defined by reference to the particular entity having or realizing it but to the functional item, which is a role depicted in purely teleological terms. This makes the function structure independent of the particular function bearers (ref. R.1.5).
- The introduced relations between functions enable not only to model a single function independently of its realizations but also a web of functions (ref. R.1.6). Classical ontological relations such as is-a, instantiation, part-of have been adopted for functions. Moreover, a number of relations specific for function such as enable, support, prevent have been defined.
- The notion of a goal permits to handle not only the proper goals of functions but also restrictions on function realizations. Side effects, both positive and negative are handled separately from the function goals (ref. R.1.7).

- The very same function in different realizations may be triggered by various triggers. Not including triggers into requirements is an additional aspect of making function structure realization-independent.
- Additionally, conflicts between functions are handled by priorities assigned to goals which permit to order functions.

Concerning the second problem area, two aspects of the notion of realization are identified. In the first sense an individual entity, in particular a process is said to be a realization of a function, and in the second - an entity, e.g. a persistent via its execution of the process of realization realizes the function. The former is called the realization of a function, the latter - the realizer of a function. This dichotomy enables to evaluate entities against being the realizations of functions, as well as against executing and contributing to the realizations of functions (ref. R.2). Moreover, since the notion of realization underlies the function ascription it permits to handle the distinction of the function of processes and the function of objects. In OF functions might be ascribed both to the entities being the realizations of functions, e.g. processes or situations as well as to the entities executing those realizations, e.g. persistents (ref. R.3.2).

Not only actual but also dispositional realizations and realizers, identified by the references to the realization and realizer universal, are handled. They permit to identify the potential realizations of functions.

Concerning the third problem area, three basic kinds of has-function relation have been introduced: dispositional has-function, actual has-function and intended has-function (ref. R.3.1). That solution seems to capture coherently most of the approaches discussed in the literature. The first two kinds correspond to the intuitions of a function understood as a capability or a actual behavior of an entity. In turn, the intended has-function takes into account the agent's intentions and justifies the normative character of function ascriptions (ref. R.3.3). In particular, it is the extension of the view of a function as the "the designed disposition", which handles not only the designed dispositions, but also those required, or intended by a user or researched. In addition, the framework permits also to assign malfunctions to entities (ref. R.3.4), which is of particular importance in item evaluation. The three modes of malfunction cover not only the malfunction of artifacts but also of other sorts of entities, e.g. body organs.

Finally, the developed ontology has been incorporated into the wider ontological framework of GFO (ref. R.4). This is of particular importance, since we have found functional knowledge not to be standing in isolation but to be related to non-functional knowledge. OF is designed as a module of GFO and the category of function is incorporated into GFO as an intentional entity, being a part of mental strata. This solution keeps the notion of function out

of processual and behavioral bias. On the basis of the developed framework we have provided three classifications of functions, which serve as the basis for the development of a taxonomy of top-level functions. Additionally, three classifications of functions based on their realizations, called external classifications, have been introduced.

The ontology is organized into a modularized architecture which, on the one hand, separates the pure functional knowledge from the non-functional one, and on the other hand enables to relate them by the realization and function ascriptions. The modularized architecture enables to apply OF into current ontologies which lack functional knowledge or comprise functional and non-functional knowledge in loose integration only without significant changes to them as will be demonstrated in the coming section.

9.4 Applications

9.4.1 Conceptual Modeling

It has been demonstrated by Guizzardi in [Guizzardi, 2005] that the top-level ontologies can provide sound and formal foundations for the structural conceptual models. Our intention is to apply OF together with the underlying GFO as a general, formal and precise language for functional modeling. In particular we are interested in extending UML - the current de facto standard in OO conceptual modeling, which has recently been proposed by many authors to be used also for ontological engineering.

UML as it was recognized in section 2.2.2 has some limitations in representing functions and functional knowledge. In particular UML permits to represent functions only in terms of behavior: “each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors”([OMG, 2004], p. 578). From this follows that if use cases are to be considered as functions, then in fact they are restricted only to behavioral functions, i.e. functions realized by some behavior (process).

UML 2.0 is composed of two main views: structural and behavioral, and lacks an independent functional view. Although, such a functional view is perhaps not required in the context of object-oriented modeling, it is however necessary if UML is supposed to be used as a general language for conceptual modeling applicable also in ontology modeling. Thus, we proposed in chapter 8 to introduce to UML a third, functional view, based on the developed OF and the underlying it GFO. The proposed extension is of particular importance for enabling modeling of domain ontologies, which require functional concepts.

For extending UML with a functional view founded on OF we developed a UML profile in which the notions of OF are introduced as stereotypes with their own graphical notations

enabling the graphical visualization of functional models. Figure 15 as well as figure 31 illustrating the application of OF in bio-ontologies make use of some of those stereotypes.

The developed UML profile serves on the one hand as a formalism for modeling functional knowledge and on the other hand it may be applied as a guide-post for specifying functions in conceptual modeling.

9.4.2 Biological Ontologies

The developed ontology of functions is intended to be domain independent and thus applicable in a number of domains. So far, the first attempts of applying it to bio-ontologies, and in particular to the Open Biomedical Ontologies (OBO), has been presented in [Burek et al., 2006]. The Open Biomedical Ontologies project (OBO) [OBO, 2005] serves as an umbrella organization providing some basic criteria and guidelines for the standardization of biomedical ontologies. It includes a large number of domain specific ontologies such as the Gene Ontology (GO) [Ashburner et al. 2000] – which provides information about processes, molecular functions and sub-cellular locations of genes and gene products – and anatomical and developmental ontologies available for specific species. In [Burek et al., 2006] OF has been recognized to be beneficial for OBO, in particular it helps in the identification and explanation of relations between processes and functions and the identification of implicit functions and processes.

Identification of Links between Processes and Functions

There has been some controversy and discussion about whether the “Molecular Function” taxonomy of the Gene Ontology describes functions or activities, and how functions are related to processes [Smith et al., 2003]. To our knowledge, no practical or theoretical solution has yet been proposed. Functions and activities are usually considered different entities, and actions or activities may realize certain functions. Therefore, while the function of an enzyme may be to catalyze a reaction, the activity performed by the enzyme is the catalysis itself, which may be embedded in another process. We assume that at least parts of the Molecular Function taxonomy refer to genuine functions in the sense of OF, and the annotation relation for some of the gene products annotated to these terms corresponds to the has-function relation. A general example is GO:0005215 (transporter activity), which we understand as referring to the function to transport. A more specific example is GO:0051119 (sugar transporter activity), which can be understood as the function to transport sugar and can be modeled in the framework of OF:

- As requirements, we assume that a sugar-molecule (CHEBI:25407 or CHEBI:25679) is located at some location.
- The goal is the location of the sugar molecule at a different location.
- The functional item is a universal role which we call `sugar transporter`.

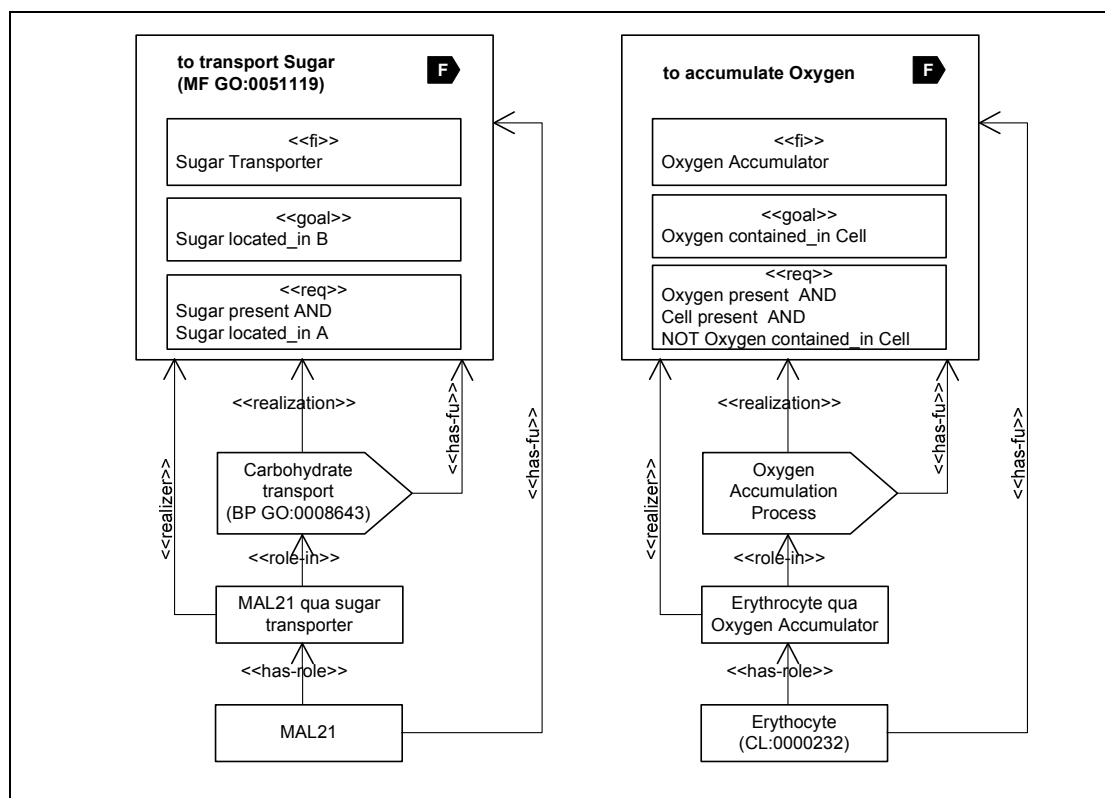


Figure 31. Two exemplary models employing OF constructed by means of the UML profile for OF. On the left-hand side, a schematic version of the function to transport sugar is shown together with its realization. Processes of the type `carbohydrate transport` realize this function, and an entity, in this case MAL21 (maltose permease), has ascribed the function to transport sugar. Whenever applicable, the identifiers from the GO are used (for the function and process). MAL21 is currently annotated to the function and the process in the GO. In this model, the annotation relation is replaced by the *has-function* relation. On the right-hand side, the function to accumulate oxygen is modeled. This is a function taken from the Celltype Ontology. Except for erythrocyte, the entities involved in this model are not present in any of the OBO ontologies but are identified by means of OF.

We find that many of the gene products annotated with the `sugar transporter` activity in GO's Molecular function taxonomy are also annotated with some sub-category of the transport (GO:0006810) or `carbohydrate transport` (GO:0008643) categories in GO's Biological Process taxonomy. Also the names of the categories indicate a link, and of course there is an obvious one: gene products which have a function to transport may participate in a transport process. With the help of OF, we can make explicit some links between categories in GO's Molecular Function and Biological Process taxonomies: Processes of type `carbohydrate transport` (GO:0008643) are

realizations of the function to transport sugar; many of the gene products annotated with either carbohydrate transport or sugar transporter activity, such as MAL21 (maltose permease), can stand in the has-function relation to to transport sugar; new categories appear, namely gene products acting as (or “qua”) transporter, e.g. MAL21 *qua* transporter which is a role of MAL21 and a realizer of the function to transport sugar. The left-hand side of figure 31 demonstrates the full interconnections of this example by means of OF. In terms of the relations we introduced this is captured by *Execute*(MAL21, GO:0008643, GO:0051119), which has the reading that MAL21 executes the process of carbohydrate transport which is a realization of the function to transport sugar. What could be directly added to GO are links of *realization* and *has-function*: *UniRl_{Min}*(GO:0008643, GO:0051119) and *UniHasFu*(MAL21, GO:0051119) saying that process GO:0008643 is the realization of function GO:0051119 and that MAL21 has a function GO:0051119.

Identification of Implicit Functions and Processes

The Ontology of Functions can be applied to existing taxonomies in order to make explicit functions and processes which are currently implied but not separately defined. This kind of use of the concept of function occurs in the Celltype Ontology [Bard et al., 2005] (CL) and the Ontology of Chemical Entities of Biological Interest [Brooksbank et al., 2005] (ChEBI). Here, we will only explore the Celltype Ontology, but the same argument can be applied to ChEBI.

CL uses the term function in the sub-tree *cell by function* which classifies cell types by the functions which they perform. A general example is *stuff accumulating cell* (CL:0000325), and more specifically *oxygen accumulating cell* (CL:0000329), of which a *red blood cell or erythrocyte* (CL:0000232) is a sub-category. The function to accumulate oxygen (by a cell) would be modeled as shown in the right-hand side of figure 31:

- The presence of oxygen (ChEBI:25805) outside of a cell (CL:0000000) is the requirement of the function.
- The goal of the function is the cell’s accumulation of oxygen: The oxygen is contained in the cell.
- The functional item is called *oxygen accumulator*.

The subsumption of *erythrocyte* under *oxygen accumulating cell* in CL reflects the fact that erythrocytes have the function to accumulate oxygen, *UniHasFu* (CL:0000232, to accumulate oxygen). Further, they may *act as* oxygen

accumulators, a new category for CL, in the process of oxygen accumulation, $UniRI_{Min}(\text{oxygen accumulation, to accumulate oxygen})$. The *execute* relation captures all these new relations appropriately: $Execute(CL:0000232, \text{oxygen accumulation, to accumulate oxygen})$. The analysis of erythrocyte in CL has led to the discovery of entities which are not yet part of CL or any other OBO ontology, but which contribute to the understanding of interactions among ontologies in cellular biology, and therefore making them amenable to automated reasoning. Additionally, we can now define oxygen accumulating cell as a cell which has the function to accumulate oxygen.

From the above we conclude that OF can be used to provide additional information for existing biomedical ontologies such as the Gene Ontology (GO), without the need for modification of the existing structure of these ontologies. In general, it provides a framework for defining functions and relating them to various other entities, such as processes, roles and even genes and gene products. This framework may benefit the annotation and curation process of domain ontologies and lead to improved definitions and completeness. The advantage of the OF is *enhanced expressivity*. For example, the curators of GO, when annotating a gene product with the appropriate terms from GO, will have the information available that a certain protein is involved in some process and how it is related to a certain molecular function. They may also have more information about the protein, for example the conditions under which it operates and other requirements which need to be satisfied for the protein to be active. By means of OF, this information can be made explicit, and will not be lost as is currently the case. OF further allows for a refinement or replacement of the annotation relation in a number of cases by means of the has-function relation. Note that the latter is an ontological relation, in contrast to the annotation relation, which is currently a database relation. Refined annotations do not only provide more information within ontologies themselves, but also with respect to the relation between categories of biomedical ontologies and genomic knowledge about biological reality. Both additional information due to enhanced expressivity and refined annotations may prove useful for the various statistical methods which have been applied to biomedical ontologies in order to detect biological correlations, such as [Beissbarth, Speed, 2004; Berriz et al., 2003; Subramanian et al., 2005].

Currently, by the effort of the Ontologies in Biomedicine Group [OBG, 2006] OF has been implemented in OWL DL and together with GFO provides the foundation of BioCoreOntology. In addition OF has been partially mapped to the molecular function taxonomy and biological process taxonomy of OBO. The mapping identifying, among others, two thousands appearances of the realization relation relies on the statistical methods and is intended to be verified by help of the appropriate curation tools being under development

9.5 Future Work

The current work presents the formal top-level ontology of functions, applicable in domain ontologies and conceptual modeling. However, the work cannot be seen as complete. Instead we recognize a number of future research directions. Firstly, several issues concerning the theoretical aspects of functions should be further analyzed; in particular evolutionary functions and etiological interpretation of function should be included into OF. Evolutionary functions are of particular importance in biology, and as the brief summary of the related work in section 2.3.2 has shown, many problems concern them. Although some aspects of etiological theories have been adopted in our framework, i.e. the intended has-function or the malfunctions with respect to the item's history, the development of the applicable notion of evolutionary function still requires much scientific effort.

The second direction of the future work concerns the interdependencies between functions and roles. The Ontology of Functions refers often to the theory of roles, especially in the context of the notions of functional item and realizer. Moreover, the notions of role and of function seem to be closely related and in fact in everyday language both notions are often used convertibly. For example, it seems that both of the following phrases share the same intuitions: “*the role* of teaching students”, “*the function* of teaching students”. Also the function ascription is often equivalent to the role ascription, e.g. compare “John has *a role* of teacher” with “John has *a function* of teaching”.

The next important direction of future research in our opinion concerns the correlation of the ontology of roles and the ontology of functions. In particular the results presented in the current work can be used for the extension of the ontology of roles incorporated into GFO [Loebe, 2006]. For instance, functional items as roles defined in purely functional terms could be considered as an additional, beside processual, social and relational, type of role.

Additional direction of future research involves the evaluation and application of the developed framework. We believe that a good evaluation method of the developed ontology is its application to the current domain ontologies of functions. We have investigated the applicability of OF in bio-ontologies as well as the partial mappings between OF and a restricted fragment of OBO has been provided and the first results seem to be promising. Now, by help of curation tools those mappings should be examined and if necessary improved. In addition we plan to introduce more notions of OF to bio-ontologies, in particular malfunction.

Moreover, it should be investigated how far OF permits to underpin other domain function ontologies, such as e.g. business function ontologies. In this sense the taxonomies of

functions developed in section 7.4 can be considered as most general functions underpinning the domain specific functions.

OF could be used as the methodological framework for the development of the domain function ontologies from scratch, which however requires the development of the methodology for the specification of domain functions based on OF. Among other things, of help in this task may be the techniques of functional design outlined in section 2.1. Some of the general principles can already be gained from the developed ontology, such as the architectural principles of delimiting function from realization or those concerning the determination of the structure of functions.

In the context of the last two issues mentioned above of particular importance is the development of appropriate tools which permit to create domain ontologies of functions, refine the current ontologies as well as support some of the methodological principles for functional modeling imposed by OF. The first of the family of such tools is a wiki-based curation system for OF [Hoehtendorf et al., 2006], being under development at the Ontologies in Biomedicine Group. It is specially suited for the annotation of gene functions by means of the relations of OF, and for enablement of the collaborative curation of the Ontology of Functions and the bio-ontologies founded on it.

Finally, the tools for graphical representation and modeling of functions can be constructed upon the UML profile introduced, which however should be developed further. In particular, in order to provide a cohesive UML-based ontological framework for modeling functions axioms of OF should be at least partially translated into Object Constraint Language.

Appendix A: GFO Terms and Definitions

The present appendix provides the reference list of the basic notions of GFO used in the thesis. Some of the notions are slightly modified and/or simplified for the purpose of the current study. The notions concerning causality are not incorporated in the current version of GFO [Heller, et al., 2006] but are published elsewhere [Michalek, 2005; Michalek, 2006]. In the below list they are labeled with OC (Ontology of Causality). The list is organized in the alphabetical order. For each notion a name, a GFO symbol and the description, optionally with selected axioms, are provided.

Name	Symbol	Description
Category		See Universal
Cause (from OC)	$Cause(x,y)$	<p>A causal relation between a cause and an effect. In [Michalek, 2005] this relation implies the following conditions:</p> <p>(1) a statistical dependency (regularity) between the existence of a cause and an effect, denoted by $Reg(x,y)$: $Cause(x,y) \rightarrow Reg(x,y)$;</p> <p>(2) the effect must be manipulable by the cause. $Man(x,Q_x,y,Q_y)$ is the relation mediating a presential x and its property Q_x with a presential y and its property Q_y, such that manipulation of the value of Q_x changes the value of Q_y: $Cause(x,y) \rightarrow \exists Q_x Q_y (Man(x,Q_x,y,Q_y))$.</p> <p>In OF the predicate $Cause(x,y)$ underpins all types of causations introduced.</p>
Cause, instantaneous	$Cause_{inst}(x,y)$	<p>The causal relation between presentials located at the same time boundary.</p> $Cause_{inst}(x,y) \leftrightarrow Pres(x) \wedge Pres(y) \wedge Reg(x,y) \wedge \exists qw (Man(x,q,y,w)) \wedge \exists t (At(x,t) \wedge At(y,t)).$
Cause, cohesive		See Process Causally Cohesive.
Cause, adhesive		See Process Casually Adhesive.

Change	$Change(e_1, e_2, u_1, u_2, u)$	Extrinsic changes are represented by $Change(e_1, e_2, u_1, u_2, u)$, where e_1 and e_2 capture the pair of process boundaries and u_1 and u_2 are disjoint sub-universals of u , such that e_1 and e_2 instantiate u_1 and u_2 , respectively. Extrinsic change holds between coinciding boundaries e_1 and e_2 , whereas in intrinsic change boundaries e_1 and e_2 are at the opposite ends of a process of arbitrary extension
Chronoid	$Chron(x)$	A temporal interval with boundaries. Chronoids are not considered in GFO as mere sets of points, but as entities <i>sui generis</i> .
Coincidence	$Coinc(x, y)$	Coincidence is a relationship between space- (spatial coincidence) and time-boundaries (temporal coincidence). Intuitively, two such boundaries are coincident if and only if they occupy “the same” space or time, but they are still different entities, e.g. in a sense that they bound different entities. For instance only a right and a left time boundary can temporally coincidence. $TCoinc(x, y) \rightarrow \exists uv ((Rb(x, u) \wedge (Lb(y, v)) \vee ((Lb(x, u) \wedge Rb(y, v))))$
Configuration	$Config(x)$	A presential which is a collection of presential facts existing at the same time-boundary i.e., it is a conglomeration of physical structures, properties and relators.
Configuroid	$Configu(x)$	An occurrence whose boundaries are configurations. $Configu(y) \rightarrow \forall x (Proc bd(x, y) \rightarrow Config(x))$
Entity	$Entity(x)$	A general notion comprising all items of GFO.
Fact	$Fact(x)$	A complex entity comprising a relator together with its relata considered as a whole. $Fact(x) \leftrightarrow \exists u y_1 \dots y_{n(i)} (rel(u, y_1, \dots, y_{n(i)}, x))$
Immanent Universal		See Universal.
Individual	$Ind(x)$	A single entity which cannot be instantiated but itself is an instance of a universal.
Instantiation	$::$	A basic binary relation, whose second argument is a universal and the first, called <i>instance</i> , is an individual or a universal. The relation $x :: y$ has the intuitive meaning that an instance x

		is of kind y . In a sense, instantiation is the intensional counterpart of the membership relation.
Level		See Stratum.
Occurrent	$Occ(x)$	An individual extended in time.
Ontological connectedness	$Ontic(x, y)$	The relation $Ontic(x, y)$ connects presentials x and y by an integrated system of spatio-temporal and causal relationships which give rise to persistants.
Part, Process Layer	$LayerPart(x, y)$	A process layer of a process p is a process which is framed by the same chronoid as p , and is a processual part of p .
Part	$Part(x, y)$	Basic relation between entities with the intuitive meaning that x is a part of y . Part-of relation comes in the number of specialized relations, these are: categorial part, constituent part, physical part, process part, proper part, spatial and temporal part.
Part, categorial	$CatPart(x, y)$	$CatPart(x, y) =_{df}$ “ x is a categorial part of y ”, where x and y are categories. It directly reflects dependencies among categories and uncovers how one category may be constructed out of others.
Part, constituent	$CPart(x, y)$	$CPart(x, y) =_{df}$ “ x is a proper constituent part of a complex entity y considered as a whole, e.g. a configuration”. Note that here we use an extended understanding of the GFO $Cpart$ relation which originally holds only for situations and situoids.
Part, physical	$PhPart(x, y)$	$PhPart(x, y) =_{df}$ “ x is a physical part of y ”, where x and y are physical structures.
Part, processual	$ProcPart(x, y)$	$ProcPart(x, y) =_{df}$ “ x is a processual part of y ”, where x and y are processes.
Part, Proper	$PPart(x, y)$	Non-reflexive part-of relation.
Part, spatial	$SPart(x, y)$	$SPart(x, y) =_{df}$ “ x is a proper spatial part of y ”, where x and y are space entities.
Part, temporal	$TPart(x, y)$	$TPart(x, y) =_{df}$ “ x is a proper temporal part of y ”, where x and y are time entities.
Persistent	$Perst(x)$	An individual extended in time but distinct from process. A persistent can be seen as a construct which binds presentials

		with the same identity, though located at different time boundaries ⁶⁷ , into one entity persisting through time.
Physical Structure	$Phys(x)$	An individual extended in space which is a bearer of properties (see property), and such that other entities cannot have it as a property.
Presential	$Pres(x)$	An individual existing wholly at a time-boundary. Every presential exists at exactly one time boundary and is called a process boundary of some process projected on the time boundary (see projection).
Process	$Proc(x)$	An individual extended in time. Processes in contrast to presentials are not located at a single time boundary but on a chronoid.
Process boundary	$ProcBd(x,y)$	If a process p is projected onto a chronoid c in terms of $Prt(p,c)$, then each boundary b of c refers to a presential e which is called the boundary of the process, denoted by $Prb(p, b, e)$, which further implies $At(e, b)$: $ProcBd(x,y) \leftrightarrow \exists t (Prb(y,t,x))$.
Process boundary, left	$ProcLBd(x,y)$	It is the presential located at the left boundary $l(c)$ of the chronoid c framing the process: $ProcLBd(x,y) \leftrightarrow \exists c (Prt(y,c) \wedge Prb(y, l(c), x))$.
Process boundary, right	$ProcRBd(x,y)$	It is the presential located at the right boundary $r(c)$ of the chronoid c framing the process. $ProcRBd(x,y) \leftrightarrow \exists c (Prt(y,c) \wedge Prb(y, r(c), x))$
Process, causally cohesive (from OC)	$Cause_{coh}(x)$	It is a process of a particular causal structure, namely every pair of coinciding (inner) time-boundaries contains presentials connected by the basic causal relation.

⁶⁷ It should be mentioned that the presented account of persistants is a simplified variant of GFO [Heller et al., 2006]. Originally, persistants are not individuals but universals instantiated by ontically connected individual presentials, and are introduced to GFO as a response to the problems yielded by the typical understanding of endurants as the entities enduring in time and wholly present at every moment of their existence. For further discussion see ([Heller et al., 2006], p. 25).

Processes, causally adhesive (from OC)	$Cause_{adh}(q,w)$	<p>A pair of temporally overlapping processes, that are causally connected throughout this overlap.</p> <p>At every pair of coinciding time-boundaries t and t', such that t is the boundary of x and t' is a boundary of y, exist causally connected presentials p and p' such that p is the boundary of the process x and p' is the boundary of the process y.</p>
Projection	$Prt(x,c)$ $At(y,t)$ $Prb(x,t,y)$	<p>It is a group of relations embedding individuals to time. We distinguish several cases of projections, denoted by $At(e, t)$, $Prt(p, c)$ and $Prb(p,b,e)$.</p> <p>The relation $At(e,t)$ assigns a presential e to a time boundary t and is read “a presential e is located at t”.</p> <p>The relation $Prt(p,c)$ assigns a process p to framing it chronoid c and is read “a process p is framed by a chronoid c”.</p> <p>If a process is projected onto a chronoid in terms of $Prt(p,c)$, each time-boundary b of c refers to a presential e which is called the boundary of the process, denoted by $Prb(p,b,e)$, which further implies $At(e, b)$.</p>
Property	$Prop(x)$	<p>A facet of an entity inherited by it:</p> $Prop(x) \leftrightarrow \exists yz (Entity(x) \wedge Inh(x,z)).$
Property Value	$Propv(x)$	An individual value of an individual property
Relator		An individual entity connecting other entities (of any kind) called relates. An instance of relation.
Role	$Role(x)$	<p>An entity played by some <i>role-player</i> in some <i>role-context</i>. A role is closely related to properties, but in contrast to those it mediates a bearer with an <i>external</i> context:</p> $Role(x) \leftrightarrow \exists yz (HasRole(y,x) \wedge RoleIn(x,z))$
Situation	$Sit(x)$	A configuration which can be comprehended as a whole and satisfies certain conditions of unity, which are imposed by relations and categories associated with the situation. Herein, we consider situations to be the most complex kind of presentials.
Situoid	$Situ(x)$	An occurrence whose boundaries are situations and which satisfies certain principles of coherence, comprehensibility, and continuity. Intuitively, it is a part of the world which is a coherent and comprehensible whole and does not need other

		entities in order to exist. Every situoid has a temporal extent and is framed by a topoid.
Stratum, Material Mental Social	$MatL(x)$ $MentL(x)$ $SocL(x)$	<p>A subsystem of GFO categories implying certain granularity. All categories of GFO are organized in three strata (called also levels): material, mental, and social. Material stratum captures categories referring to objects of physical world denoted by $MatL(x)$. Mental stratum comprises most of what is studied by cognitive science (perception, memory, reasoning, etc) and the categories referring to will denoted by $MentL(x)$. Social stratum covers such categories as agents and institutions. Items of social strata are denoted by $SocL(x)$.</p> <p>Among these levels specific forms of categorial and existential dependencies hold. For example, a mental entity requires an animate physical object as its existential bearer.</p>
Time boundary Left, right, inner.	$Tb(x)$, $Lb(x,c)$, $Rb(y,c)$	<p>Time entities distinct from chronoids. Every chronoid c has exactly two extremal boundaries – a left boundary denoted by $Lb(x,c)$ and a right boundary denoted by $Rb(y,c)$. Moreover, it has infinitely many inner time boundaries. The boundaries depend on a chronoid, i.e. they have no independent existence. Moreover, the boundaries can coincide (see Coincidence).</p> <p>$TB(x) \leftrightarrow \exists y Tb(x,y)$.</p>
Time entity	$Te(x)$	$Te(x) \leftrightarrow TB(x) \vee Chron(x)$.
Universal Primitive, Immanent, Conceptual Structure.	$Uni(x)$	<p>An entity which may be predicated of or instantiated by other entities. Universals whose all instances are individuals are called primitive universals. In addition, at least two kinds of universals can be distinguished: <i>immanent universals</i> and <i>conceptual structures</i>.</p> <p>The immanent universals are assumed to exist in the individuals (in re) but not independently from them. On the other hand, humans as cognitive subjects conceive of universals of any sort by means of concepts that are in their minds. Those are called conceptual structures.</p> <p>Universals can be classified also in accordance with the classification of corresponding individuals, e.g. process universals.</p>

Complex Whole	$Whole(x)$	The general notion underpinning all complex entities and their universals, i.e. fact, configuration, configuroid, situation and situoid.
Coherent Entity (extends GFO)		A complex entity (a whole), that all its constituent parts are interrelated. $Coh(x) \leftrightarrow \forall yz (CPart(y,x) \wedge CPart(z,x) \wedge z \neq y \wedge x = y + z \rightarrow Rel(y,z))$
Process common start	$ProcStarts(x,y)$	Two processes having the common beginnings: $ProcStarts(x, y) \leftrightarrow \exists uv (Prt(x,u) \wedge Prt(y,v) \wedge Starts(u,v))$, where $Starts(u,v)$ is the relation between two chronoids having the same left boundary.
Process common end	$ProcEnds(x, y)$	Two processes having the common endings: $ProcEnds(x,y) \leftrightarrow \exists uv (Prt(x,u) \wedge Prt(y,v) \wedge Ends(u,v))$, where $Ends(u,v)$ is the relation between two chronoids having the same right boundary.

Table 8. The reference list of GFO terms and definitions.

References

- [Abdullah et al., 2004] Abdullah, M.S., Kimble, C., Paige, R., Benest, I., Evans, A.: Developing a UML Profile for Modelling Knowledge-Based Systems. *MDAFA 2004*, p. 220-233, 2004.
- [Akman, Surav, 1996] Akman, V., Surav, M.: Steps toward formalizing context. *AI Magazine*, 17(3):55–72, 1996.
- [Allemang, 1991] Allemang, D.: Using functional models in automatic debugging. *IEEE Expert*, 6(6):13-18, 1991.
- [Allemang, Chandrasekaran, 1991] Allemang, D., Chandrasekaran, B.: Functional representation and program debugging. *Proceedings of the 6th Annual Knowledge-Based Software Engineering Conference*, p. 136-152, IEEE Computer Society Press, 1991.
- [Allen, 1984] Allen, J.: Towards a general theory of action and time. *Artificial Intelligence*, 23:123-154, 1984.
- [Ashburner et al., 2000] Ashburner, M., Ball, C. A., Blake, J. A., Botstein, D., Butler, H., Cherry, J.M., Davis, A. P., Dolinski, K., Dwight, S. S., Eppig, J. T., Harris, M. A., Hill, D. P., Issel-Tarver, L., Kasarskis, A., Lewis, J. S., Matese, C., Richardson, J. E., Ringwald, M., Rubin, G. M., Sherlock, G.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, May, 2000.
- [Ashworth, Goodland, 1990] Ashworth, C., Goodland, M.: *SSADM A Practical Approach*. McGraw-Hill, 1990.
- [Baclawski et al., 2001] Baclawski, K., Kokar, M. M., Kogut, P. A., Hart, L., Smith, J. E., Holmes, W. S., Letkowski, J., and Aronson, M. L.: Extending UML to Support Ontology Engineering for the Semantic Web. In Gogolla, M., Kobryn, C. (eds) *Proceedings of the 4th international Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools, October 01 - 05, 2001*, p. 342-360, Lecture Notes In Computer Science, Vol. 2185, Springer Verlag, London, 2001.
- [Bard et al., 2005] Bard, J., Rhee, S. Y., Ashburner, M.: An ontology for cell types. *Genome Biol*, 6(2):R21, 2005.
- [Barker, 1990a] Barker, R.: *CASE*Method: Tasks and Deliverables*. Oracle Corporation UK

- Limited, Addison-Wesley, Wokingham, England, 1990.
- [Barker, 1990b] Barker, R.: *CASE Method: Entity Relationships Modelling*. Oracle Corporation UK Limited, Addison-Wesley, Wokingham, England, 1990.
- [Barker, Longman, 1992] Barker, R., Longman, C.: *CASE*Method: Function and Process Modeling*. Oracle Corporation UK Limited, Addison-Wesley, Wokingham, England, 1992.
- [Barton, Komatsu, 1989] Barton, M.E., Komatsu, L.K.: Defining features of natural kinds and artifacts. *Journal of Psycholinguistic Research*, 18(5), 443-447, 1989.
- [Beissbarth, Speed, 2004] Beissbarth, T., Speed, T. P.: GOstat: find statistically overrepresented Gene Ontologies within a group of genes. *Bioinformatics*, 20(9):1464–1465, June, 2004.
- [Bergenti, Poggi, 2000] Bergenti, F., Poggi, A.: Exploiting UML in the design of multi-agent systems. In Omicini, A., Tolksdorf, R., Zambonelli, F. (eds.) *Engineering Societies in the Agents World*, p. 106–113, Lecture Notes in Computer Science, vol. 1972, Springer Verlag, 2000.
- [Berriz et al., 2003] Berriz, G. F., King, O. D., Bryant, B., Sander, C., Roth, F. P.: Characterizing gene sets with FuncAssociate. *Bioinformatics*, 19(18):2502–2504, December, 2003.
- [Bloom, 1996] Bloom, P.: Intention, History, and Artifact Concepts. *Cognition* 60:1-29, 1996.
- [Bo, Salustri, 1999] Bo, Y., Salustri, F.: Function Modeling Based on Interactions of Mass, Energy and Information. *FLAIRS Conference 1999*: 384-388, 1999.
- [Bonnet, 1992] Bonnet, J. C.: *Towards a formal representation of device functionality*. TR 92-54, Knowledge Systems Laboratory, Stanford University, 1992.
- [Booch, 1993] Booch G.: *Object-oriented Analysis and Design with Applications, 2nd edition*. Benjamin Cummings, Redwood City, CA, 1993.
- [Borgo et al., 1996] Borgo, S., Guarino, N., and Masolo, C.: A Pointless Theory of Space Based on Strong Connection and Congruence. In Aiello, L. C., Doyle, J. (eds.) *Principles of Knowledge Representation and Reasoning (KR96)*, Morgan Kaufmann, 1996.
- [Borgo et al., 1997] Borgo, S., Guarino, N., Masolo, C.: An Ontological Theory of Physical Objects. In Ironi L. (ed.) *Proceedings of Eleventh International Workshop on Qualitative Reasoning (QR'97)*, p. 223-231, Italy, 1997.
- [Borgo, Leitão, 2004] Borgo, S., Leitão, P.: The Role of Foundational Ontologies in Manufacturing Domain Applications. In Meersman, R., Tari, Z. et al. (eds.) *OTM*

- Confederated International Conferences, ODBASE 2004, Ayia Napa, Cyprus, October 29, 2004*, p. 670-688, Lecture Notes in Computer Science, Vol. 3290, Springer Verlag, 2004.
- [Borst, 1997] Borst, W.N.: *Construction of Engineering Ontologies for knowledge sharing and reuse*. Centre for Telematica and Information Technology, University of Twente, Enschede, The Netherlands, 1997.
- [Bracewell, Sharpe, 1996] Bracewell R.H., Sharpe, J.: Functional Descriptions Used in Computer Support for Qualitative Scheme Generation— Schemebuilder. *AIEDAM*, (10)4:333–346, 1996.
- [Brachman, 1983] Brachman, R.J.: What is-a is and isn't: an analysis of taxonomic links in semantic networks. *IEEE Computer*, 16(10):30-36, October 1983.
- [Brachman, 1985] Brachman, R.J.: I Lied About the Trees, Or, Defaults and Definitions in Knowledge Representation. *AI Magazine*, 6(3): 80-93, 1985.
- [Bratman, 1987] Bratman, M. E.: *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.
- [Britton et al., 2000] Britton G, Yimin D, Beng T. S.: *Functional Design: a system viewpoint*. School of Mechanical and Production Engineering, Nanyang Technological University, Singapore, 2000.
- [Brockmans et al., 2004] Brockmans, S., Volz, R., Eberhart, A., and Loeffler, P.: Visual Modeling of OWL DL Ontologies using UML. In van Harmelen et al. (eds.) *ISWC 2004*, p. 198- 213, Lecture Notes in Computer Science, Vol. 3298, Springer Verlag, 2004.
- [Brooksbank et al., 2005] Brooksbank, C., Cameron, G., Thornton, J.: The European Bioinformatics Institute's data resources: towards systems biology. *Nucleic Acids Res*, 33(Database issue):D46–D53, January, 2005.
- [Brown, Blessing, 2005] Brown, D.C, Blessing, L.: The relationship between function and affordance. *Proceedings of IDETC/CIE 2005:ASME 2005 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, California, USA, 2005.
- [Buller, 1998] Buller, D.: Etiological Theories of Function: A Geographical Survey. *Biology and Philosophy*, 13:505–527, Kluwer Academic Publishers, The Netherlands, 1998.
- [Burek et al., 2006] Burek, P., Hoehendorf, R., Loebe, F., Visagie, J., Kelso, J., Herre, H.: A top-level ontology of functions and its application in Open Biomedical Ontologies. [Accepted for ISMB 2006, to appear in *Bioinformatics*].

- [Burek, 2004] Burek, P.: Adoption of the Classical Theory of Definition to Ontology Modeling. In Bussler, C., Fensel, D. (eds.) *Artificial Intelligence: Methodology, Systems, and Applications. Proceedings of the 11th International Conference, AIMSA 2004, Sep 2-4, Varna, Bulgaria*, p. 1-10, Lecture Notes in Computer Science, Vol. 3192, Springer Verlag, Berlin, 2004.
- [Burek, 2005] Burek, P.: Essentialized Conceptual Structures in Ontology Modeling. In Khosla, R., Howlett, R.J., Jain, L.C. (ed.) *Knowledge-Based Intelligent Information and Engineering Systems: Proceedings of the 9th International Conference, KES 2005, Melbourne, Australia, Sep 14-16, Part II*, p. 880-886, Lecture Notes in Computer Science, Vol. 3682, Springer Verlag, Berlin, 2005.
- [Burek, Grabos, 2005] Burek, P., Grabos, R.: Dually Structured Concepts in the Semantic Web: Answer Set Programming Approach. In Gómez-Pérez, A., Euzenat, J. (eds.) *The Semantic Web: Research and Applications*. p. 377-391, Lecture Notes in Computer Science, Vol. 3532, Springer Verlag, 2005.
- [Casati, Varzi, 1995] Casati, R., Varzi, A.: *Holes and Other Superficialities*, MIT Press, Cambridge, MA, 1995.
- [Casati, Varzi, 2002] Casati, R., Varzi, A.: Events. In Edward N. Zalta (eds.) *The Stanford Encyclopedia of Philosophy* (Fall 2002 Edition), Available at: <http://plato.stanford.edu/archives/fall2002/entries/events/>, 2002.
- [Chandrasekaran et al., 1993] Chandrasekaran, B., Goel, A., Iwasaki Y.: Functional Representation as Design Rationale. *IEEE Computer*, 26(1): 48-56, 1993.
- [Chandrasekaran, 1994a] Chandrasekaran B.: Functional representation: A brief historical perspective. *Applied Artificial Intelligence*, 8(2): 173-197, 1994.
- [Chandrasekaran, 1994b] Chandrasekaran B.: Functional Representation and Causal Processes. *Advances in Computers*, 38: 73-143, 1994.
- [Chandrasekaran, Josephson, 1997] Chandrasekaran, B., Josephson, J.R.: Representing Function as Effect. *Proceedings of the Functional Modeling Workshop*, Paris, France, 1997.
- [Chandrasekaran, Josephson, 2000] Chandrasekaran, B., Josephson, J. R. : Function in Device Representation. *Engineering with Computers, Special Issue on Computer Aided Engineering*, 16:162-177, 2000.
- [Chen, 1976] Chen, P.: The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9-36, 1976.

- [Coad, Yourdon, 1991] Coad, P., Yourdon, E.: *Object-Oriented Design*. Yourdon Press Computing Series Prentice-Hall, Englewood Cliffs, NJ, 1991.
- [Cohen , 2002] Cohen, S. M.: *Lecture on the Four Causes*. University of Washington. Available at: <http://faculty.washington.edu/smcohen/320/4causes.htm>, 2002.
- [Corcho et al., 2001] Corcho, O., Fernández-López, M., Gómez Pérez, A.: *OntoWeb:Technical Roadmap v1.0 30.11.01*. Universidad Politécnica de Madrid, Madrid, Spain, 2001.
- [Corcho, Gomez-Perez, 2000] Corcho, O., Gomez-Perez, A.: A Roadmap to Ontology Specification Languages. In Dieng R. Corby O. (eds.) *12th International Conference on Knowledge Acquisition, Modeling and Management, EKAW 2000, Juan-les-Pins, France, October 2-6, 2000*, p. 80-96, Lecture Notes in Computer Science, Vol. 1937, Springer Verlag, 2000.
- [Cranefield et al., 2001] Cranefield, S., Haustein, S., Purvis, M.: *UML-based ontology modelling for software agents. Proceedings of Ontologies in Agent Systems Workshop, Agents 2001*, Montreal, Canada, 2001.
- [Cranefield, Purvis, 1999] Cranefield, S., Purvis, M.K.: UML as an Ontology Modelling Language. *Proceedings of the Workshop on Intelligent Information Integration*, volume 23 of *CEUR Workshop Proceedings*, Stockholm, Sweden, July 1999.
- [Cummins, 1975] Cummins, R.: Functional Analysis. *Journal of Philosophy* 72:741–765, 1975.
- [Cummins, 2002] Cummins, R.: Neo-Teleology. In Ariew, Perlman (eds) *Functions: New Essays in The Philosophy of Psychology and Biology*, p. 157-173, Oxford University Press, Oxford, 2002.
- [Cyc Project, 2005] Cyc Project [homepage]. Available from: <http://www.cyc.org>. Cited 2005.
- [Davies, 2000a] Davies, P.S.: The Nature of Natural Norms: Why Selected Functions are Systemic Capacity Functions. *Noûs*, (34)1: 85-107, 2000.
- [Davies, 2000b] Davies, P.S.: *Norms of Nature: Naturalism and the Nature of Functions*. MIT Press Cambridge, Massachusetts and London, England, 2000.
- [Davies, 2000c] Davies, P.S.: Malfunctions. *Biology and Philosophy*, 15:19-38, 2000.
- [Degen et al., 2001] Degen, W., Heller, B., Herre, H., Smith, B.: GOL: A General Ontological Language. In Welty C., Smith B., (eds.) *Proceedings of the International Conference on Formal Ontology in Information Systems, (FOIS 2001)*, p. 34-46, Ogunquit, Main, ACM

- Press, New York, Oct 2001.
- [Dermuth, Hussmann, 1999] Dermuth, B., Hussmann, H.: Using UML/OCL Constraints for Relational Database Design. *Proceedings <<UML>>'99*, Fort Collins, Colorado, 1999.
- [Djurić et al., 2004] Djurić, D., Gašević, D., Devedžić, V., Damjanović, V.: UML Profile for OWL. In Aßmann, U., Aksit, M., Rensink, A. (eds.) *Model Driven Architecture. European MDA Workshops: Foundations and Applications, MDFA 2003 and MDFA 2004*, Twente, The Netherlands, 2003
- [Domingue et al., 1999] Domingue, J., Motta, E., Corcho, O.: Knowledge Modelling in WebOnto and OCML: A User Guide. Available from: <http://kmi.open.ac.uk/projects/ocml/ocml-webonto-guide.zip>, 1999.
- [Dori, 2002] Dori, D.: *Object Process Methodology: A Holistic Systems Paradigm*. Springer Verlag, Heidelberg, New York, 2002.
- [Eriksson, Penker, 2000] Eriksson, H., Penker, M.: *Business Modeling with UML - Business Patterns at Work*. OMG Press, Wiley, John and Sons, 2000.
- [Falkovych et al., 2003] Falkovych, K., Sabou, M., Stuckenschmidt, H.: UML for the Semantic Web: Transformation-Based Approaches. In Omelayenko, B., Klein, M. (eds.) *Knowledge Transformation for the Semantic Web*, p. 92-106. IOS Press, 2003.
- [Fernandes, 2003] Fernandes, J.M.: *Functional and Object-Oriented Modeling of Embedded Software*. TUCS Technical Reports, No 512, Turku Centre for Computer Science, Feb 2003.
- [Gane, Sarson, 1979] Gane, C., Sarson T.: *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, N. J., Prentice-Hall, 1979.
- [Gangemi et al., 2003] Gangemi, A., Guarino, N., Masolo, C., Oltramari, O.: Sweetening WORDNET with DOLCE. *AI Magazine* 24(3): 13-24, 2003.
- [Gärdenfors, P. 2000] Gärdenfors, P.: *Conceptual Spaces: The Geometry of Thought*. A Bradford Book, MIT Press, Cambridge, Massachusetts, 2000.
- [Gelman, Bloom, 2000] Gelman, S.A., Bloom, P.: Young children are sensitive to how an object was created when deciding what to name it. *Cognition* 76:91-103, 2000.
- [Gelman, Wellman, 1991] Gelman, S. A., Wellman, H. M.: Insides and essences: early understandings of the nonobvious. *Cognition* 38:213-244, 1991.
- [Gero, 1990] Gero, J.S.: Design prototypes: a knowledge representation schema for design. *AI Magazine*, 11(4): 26-36, 1990.

- [Gero, Kannengiesser, 2004] Gero, J.S., Kannengiesser, U.: The situated Function-Behaviour-Structure framework. *Design Studies* 25(4): 373-391, 2004.
- [Glass, 2002] Glass, R. L.: The Naturalness of Object Orientation: Beating a Dead Horse? *IEEE Software*, 19(3):103–104, 2002.
- [Godfrey-Smith, 1993] GodfreySmith, P.: Functions: Consensus Without Unity. *Pacific Philosophical Quarterly*, 74: 196–208, 1993.
- [Gomez Perez et al., 2004] Gomez Perez, A., Fernandez Lopez, M., Corcho Garcia, O., Corcho-Garcia, O., Fernandez-Lopez, M., Corcho, O.: *Ontological Engineering : with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Advanced Information and Knowledge Processing, Springer Verlag, 2004.
- [Gornik, 2003] Gornik, D. *UML Data Modeling Profile*. IBM Rational Software Whitepaper TP 162 05/02, 2003.
- [Gould, Lewontin, 1979] Gould, S.J., Lewontin R.: The spandrels of San Marco and the Panglossion paradigm: a critique of the adaptationist programme. *Proc. of the Royal Society of London, Series B, Biological Sciences*, 205 (1161): 581-598, 1979.
- [Griffiths, 1993] Griffiths, P.: Functional Analysis and Proper Function. *British Journal for the Philosophy of Science*, 44:409–422, 1993.
- [Gruber, 1993] Gruber, T. R.: A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199-220, 1993.
- [Gruber,1994] Gruber, T.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human and Computer Studies*, 43(5/6): 907-928, 1994.
- [Guarino, 1997a] Guarino N.: Understanding, Building, and Using Ontologies. *International Journal of Human-Computer Studies*, 46(2):293-310, 1997.
- [Guarino, 1997b] Guarino N.: Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In Pazienza, M.T. (ed.) *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, p.139-170, Springer Verlag, 1997.
- [Guarino, 1998] Guarino N.: Formal Ontology in Information Systems. *Proceedings of the Conference On Formal Ontology In Information Systems (FOIS-98)*, p.3-15, IOS Press, Amsterdam, 1998.
- [Guarino, Giaretta, 1995] Guarino, N., Giaretta, P.: Ontologies and Knowledge Bases:

- Towards a Terminological Clarification. In Mars N.(ed.) *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, p. 25-32, IOS Press, Amsterdam, 1995.
- [Guarino, Welty, 2000] Guarino, N., Welty, C.: A Formal Ontology of Properties. In R. Dieng, O. Corby (eds.) *Knowledge Engineering and Knowledge Management: Methods, Models and Tools. 12th International Conference, EKAW2000*, p. 97-112, Springer Verlag, 2000.
- [Guarino, Welty, 2004] Guarino, N., Welty, C.: An Overview of OntoClean. In Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, p. 151-159, Springer Verlag, 2004.
- [Guizzardi et al., 2002a] Guizzardi, G., Herre, H., Wagner, G.: On the General Ontological Foundations of Conceptual Modeling. *Proceedings of 21th International Conference on Conceptual Modeling (ER2002), Tampere, Finland, Oct 2002*, p. 97-112, Lecture Notes in Computer Science, Springer Verlag, Berlin, 2002.
- [Guizzardi et al., 2002b] Guizzardi, G., Herre, H., Wagner, G.: Towards Ontological Foundations for Conceptual UML Models. In Meersman R, Tari Z. et al. (eds.) *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE. Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics, (ODBASE), 2002 Okt 29-31. Irvine, California*, p. 1100-1117, Springer Verlag, 2002.
- [Guizzardi et al., 2004a] Guizzardi, G., Wagner, G., Herre, H. 2004. On the Foundations of UML as an Ontology Representation Language. In Motta, E., Shadbolt, N., Stutt, A., Gibbins, N. (eds.) *Engineering Knowledge in the Age of the Semantic Web: Proceedings of the 14th International Conference (EKAW 2004), Whittlebury Hall, UK, Oct 2004*, p. 47-62, Lecture Notes in Computer Science, Vol. 3257, Springer Verlag, 2004.
- [Guizzardi, 2005] Guizzardi, G.: *Ontological Foundations for Conceptual Models*. PhD Thesis, Telematics Instituut fundamental Research Series, No. 015, Telematics Instituut, Enschede, The Netherlands, 2005.
- [Guizzardi, et. al., 2004b] Guizzardi, G., Wagner G., Guarino, N., van Sinderen, M.: An Ontologically Well-Founded profile for UML Conceptual Models. *CAiSE 2004*, p. 112-126, Springer Verlag, 2004.
- [Guizzardi, Wagner, 2002] Guizzardi, G., Wagner G.: Using Formal Ontologies to define Real-World Semantics for UML Conceptual Models. *First Workshop on Application of Ontologies to Biology*, European Media Laboratory, Heildelberg, Germany, 2002.
- [Halpin, 1997] Halpin, T.: *Object Role Modeling: an overview*. Electronic paper available at: www.orm.net/pdf/ORMwhitePaper.pdf, 1997.

- [Helbig, 2001] Helbig, H.: *Die semantische Struktur natürlicher Sprache: Wissensrepräsentation mit MultiNet*. Springer Verlag, Berlin, 2001.
- [Heller et al., 2005] Heller, B., Herre, H., Burek, P., Loebe, F., Michalek, H.: *General Formal Ontology (GFO): A Foundational Ontology Integrating Objects and Processes (Version 1.0 D1)*. Onto-Med Report Nr. 8, Research Group Ontologies in Medicine (Onto-Med), Leipzig University, Germany, 2005.
- [Heller et al., 2006] Heller, B., Herre, H., Burek, P., Hoehndorf R., Loebe F., Michalek, H.: *General Formal Ontology: A Foundational Ontology Integrating Objects and Processes*. Onto-Med Report 8, Univeristy of Leipzig, 2006.
- [Heller, Herre, 2004] Heller, B., Herre, H. 2004. Ontological Categories in GOL. *Axiomathes* 14(1):57-76.
- [Herre, Loebe, 2005] Herre, H., Loebe, F.: A Meta-ontological Architecture for Foundational Ontologies. In Meersman, R., Tari, Z. (eds.) *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: Proceedings of the OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, Oct 31 - Nov 4, 2005, Part II*, p. 1398-1415, Lecture Notes in Computer Science, Vol. 3761, Springer Verlag, Berlin, 2005.
- [Hoehndorf et al., 2006] Hoehndorf, R., Prüfer, K., Backhaus, M., Visagie, J., Kelso, J.: The design of a wiki-based curation system for the Ontology of Functions. *The Joint BioLINK and 9th Bio-Ontologies Meeting*, 2006.
- [Hubka, Eder, 1998] Hubka V, Eder W.: *Theory of Technical Systems*. Springer Verlag, Berlin, 1998.
- [Hughes, Cresswell, 1996] Hughes, G. E. and Cresswell, M. J. 1996. *A New Introduction to Modal Logic*. Routledge, London, 1996.
- [IBM, Sandpiper, 2005] IBM, Sandpiper Software, Inc.: *Ontology Definition Metamodel. Fourth Revised Submission to OMG/ RFP ad/2003-03-40*. Available from: <http://www.omg.org/docs/ad/05-09-08.pdf>, 2005.
- [Irwin, Turk, 2005] Irwin, G., Turk, D.: An Ontological Analysis of Use Case Modeling Grammar. *Journal of the Association for Information Systems*, 6(1): 1-36, January 2005.
- [Iwasaki et al., 1993] Iwasaki, Y., Fikes, R., Vescovi, M., and Chandrasekaran B. (1993). How Things Are Intended to Work: Capturing functional knowledge in device design. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, p. 1516-1522, Morgan Kanaufmn, Mountain View, CA, 1993.

- [Iwasaki et al., 1995] Iwasaki, Y., Vescovi, M., Fikes, R., Chandrasekaran, B.: A Causal functional representation language with behavior-based semantics. *Applied Artificial Intelligence*, 9(1):5-31, 1995.
- [Iwasaki, Chandrasekaran, 1992] Iwasaki, Y., Chandrasekaran, B.: Design verification through function- and behavior-oriented representations: Bridging the gap between function and behavior. In Gero, J.S. (ed.) *Artificial Intelligence in Design '92*, p. 597-616, Kluwer Academic Publishers, 1992.
- [Johansson, 2004] Johansson, I.: Functions, Function Concepts, and Scales. *The Monist*, 87(1):96-114, 2004.
- [Kassel, 2005] Kassel, G.: *Integration of the DOLCE top-level ontology into the OntoSpec methodology*. LaRIA Research Report: LRR 2005-08, 2005.
- [Keil, 1989] Keil, F.: *Concepts, kinds, and cognitive development*. MIT Press, Cambridge, MA, 1989.
- [Keil, 2003] Keil, F.: Categorization, Causation and the Limits of Understanding. *Language and Cognitive Processes*, 18:663-69, 2003.
- [Keleman, 1999] Keleman, D.: Function, goals, and intention: Children's teleological reasoning about objects. *Trends in Cognitive Sciences*, 3:461-468, 1999.
- [Keuneke, 1989] Keuneke, A.: Machine Understanding of Devices: *Causal Explanation of Diagnostic Conclusions*. PhD thesis, The Ohio State University, 1989.
- [Keuneke, 1991] Keuneke, A.: Device Representation: The Significance of Functional Knowledge. *IEEE Expert*, 6(2):22-25, 1991.
- [Kifer et al., 1995] Kifer, M., Lausen, G., Wu, J.: Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 1995.
- [Kitamura et al., 2002] Kitamura, Y., Sano, T., Namba, K., and Mizoguchi, R.: A Functional Concept Ontology and Its Application to Automatic Identification of Functional Structures. *Advanced Engineering Informatics*, 16(2):145-163, 2002.
- [Kitamura et al., 2004] Kitamura Y, Kashiwase M., Fuse, M., Mizoguchi, R.: Deployment of an ontological framework of functional design knowledge. *Advanced Engineering Informatics*, 18:115-127, 2004.
- [Kitamura, Mizoguchi, 1998] Kitamura, Y., Mizoguchi, R.: Functional ontology for functional understanding. *Papers of Twelfth Inter-national Workshop on Qualitative Reasoning (QR-98)*, Cape Cod, USA, May 26-29, p. 77-87, AAAI Press, 1998.

- [Kitamura, Mizoguchi, 1999] Kitamura, Y., Mizoguchi, R.: Metafunctions of artifacts, *Papers of 13th International Workshop on Qualitative Reasoning (QR-99)*, p.136-145, 1999.
- [Kitamura, Mizoguchi, 2004] Kitamura, Y., Mizoguchi, R.: Ontology-based systematization of functional knowledge. *Journal of Engineering Design*, 15(4):327-351, August 2004.
- [Kitcher, 1993] Kitcher, P.: Function and Design. *Midwest Studies in Philosophy*, 18:379-397, 1993.
- [Kogut et al., 2002] Kogut, P. A., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M. M., Smith, J. E.: UML for Ontology Development. *The Knowledge Engineering Review*, 17(1):61-64, 2002.
- [Kreos, 2001] Kreos, P.: Technical Functions as Dispositions: a Critical Assessment. *Techné*, 5(3), Spring 2001.
- [Lassila, McGuinness, 2001] Lassila, O., McGuinness, D.L.: The Role of Frame-Based Representation on the Semantic Web. Knowledge Systems Laboratory Report KSL-01-02, Stanford University, 2001.
- [Laurence, Margolis, 1999] Laurence, S., Margolis, E. (eds.): *Concepts - Core Readings*. MIT Press, 1999.
- [Lind, 1990] Lind, M. (1990): *Representing Goals and Functions of Complex Systems*. Technical Report 90-D-381, Department of Automation, Lyngby, Denmark, 1990.
- [Lind, 1994] Lind, M.: Modeling Goals and Functions of Complex Industrial Plants. *Applied Artificial Intelligence*, 8: 259-283, 1994.
- [Lind, 1999] Lind, M.: Plant Modeling for Human Supervisory Control. *Transactions of the Institute of Measurement and Control*, 21(4/5):171-180, 1999.
- [Loebe, 2003] Loebe, F.: *An Analysis of Roles*. *Onto-Med Report Nr. 6*. Research Group Ontologies in Medicine (Onto-Med), Leipzig University, Germany, 2003.
- [Loebe, 2005] Loebe, F. 2005. Abstract vs. Social Roles: A Refined Top-Level Ontological Analysis. In Boella, G., Odell, J., van der Torre, L., Verhagen, H. (eds.) *Proceedings of the 2005 AAI Fall Symposium 'Roles, an Interdisciplinary Perspective: Ontologies, Languages, and Multiagent Systems'*, p. 93-100, Menlo Park, California, 2005.
- [LOOM, 1995] *Tutorial for Loom version 2.1*. Available at: <http://www.isi.edu/isd/LOOM/documentation/tutorial2.1.html>, May 1995.
- [Majerus, 1998] Majerus M.: *Melanism: Evolution in Action*. Oxford University Press, USA, 1998.

- [Masolo et al., 2002] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: *Wonderweb Deliverable D17. Preliminary Report. Version 2.0*. 15.08.2002. Laboratory For Applied Ontology, ISTC-CNR, Padova, Italy, 2002.
- [Masolo et al., 2003] Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: *WonderWeb Deliverable D18. Ontology Library (final). Version 1.0*. 31.12.2003. Laboratory For Applied Ontology, ISTC-CNR, Trento, Italy, 2003.
- [Matan, Carey, 2001] Matan, A., Carey, S.: Developmental changes within the core of artifact concepts. *Cognition*, 78:1-26, 2001.
- [McCarthy, Buvač, 1998] McCarthy, J., Buvač, S.: Formalizing context (expanded notes). In Aliseda, A., van Glabbeek, R. J., Westerståhl, D. (eds.) *Computing Natural Language*, volume 81 of *CSLI Lecture Notes*, p.13–50, Center for the Study of Language and Information (CSLI), Stanford University, Stanford, 1998.
- [McDowell et al., 1996] McDowell, J. et al.: Conceptual Design for Polymer Composite Assemblies. In Sharpe, J. (ed.) *AI System Support for Conceptual Design*, p. 377–389, Springer-Verlag, 1996.
- [Medin, Ortony, 1989] Medin, D. L., Ortony, A.: Psychological essentialism. In Vosniadou, S., Ortony, A. (eds.) *Similarity and analogical reasoning*, p. 179–196, Cambridge University Press, New York, 1989.
- [Michalek, 2005] Michalek, H.: A Causal Relation Based on Regularity and Manipulability. In Guizzardi, G., Wagner, G. (eds.) *Proceedings of the EDOC International Workshop on Vocabularies, Ontologies and Rules for the Enterprise (VORTE'05), Enschede, The Netherlands, Sep 20*. CTIT Workshops Proceedings. Enschede (Netherlands): CTIT, 2005.
- [Michalek, 2006] Michalek, H.: *Causality: A Formal Ontological Account within the Top-level Ontology of GFO*. [forthcoming] PhD thesis, University of Leipzig, 2006.
- [Millikan 1989a] Millikan, R.G.: An Ambiguity in the Notion “Function”. *Biology and Philosophy*, 4:172–176, 1989.
- [Millikan 1989b] Millikan, R.G.: In Defense of Proper Functions. *Philosophy of Science* 56:288–302, 1989.
- [Millikan, 2002] Millikan R. G.: Biofunctions: two paradigms. In Cummins, R., Ariew, A., Perlmaneds, R. (eds.) *Functions: New Readings in the Philosophy of Psychology and Biology*, 113-143, Oxford University Press, Oxford, 2002.
- [Millikan,1984] Millikan, R.G.: *Language, Thought, and Other Biological Categories*, MIT Press, Cambridge, MA, 1984.

- [Neander, 1991a] Neander, K.: The Teleological Notion of “Function”. *Australasian Journal of Philosophy*, 69:454–468, 1991.
- [Neander, 1991b] Neander, K.: Functions as Selected Effects: The Conceptual Analyst’s Defense. *Philosophy of Science*, 58:168–184, 1991.
- [Neches et al., 1991] Neches, R., Fikes, R.E., Finin, T., Gruber, T.R., Senator, T., Swartout, W.R.: Enabling Technology for Knowledge Sharing. *AI Magazine*, 12(3):36-56, 1991.
- [Niles, Pease, 2001] Niles, I., Pease, A.: Towards a Standard Upper Ontology. In Welty, C., Smith, B. (eds.) *Formal Ontology in Information Systems: Collected Papers from the Second International Conference*, p. 2-9, ACM Press, New York, October, 2001.
- [OBG, 2006] Ontologies in Biomedicine Graoup [homepage]. <http://onto.eva.mpg.de/>, cited 2006.
- [OBO, 2005] Open Biomedical Ontologies [homepage]. <http://obo.sourceforge.net/>, cited 2005.
- [OMG, 2002] Object Management Group: *Meta Object Facility (MOF) Specification. Version 1.4, OMG Document formal/2002-04-03*. Available from: <http://www.omg.org/cgi-bin/doc?formal/02-04-03.pdf>, April 2002.
- [OMG, 2003a] Object Management Group: *Ontology Definition Metamodel - Request For Proposals*. March 2003.
- [OMG, 2003b] Object Management Group: *UML 2. 0 Infrastructure Specification*. OMG Adopted Specification. ptc/03-09-15. Available from: www.omg.org/docs/ptc/03-09-15.pdf, 2003.
- [OMG, 2004] Object Management Group: *Unified Modeling Language (UML) Specification: Infrastructure. Version 2.0. ptc/04-10-14*. Available from: <http://www.omg.org/docs/ptc/04-10-14.pdf>, 2004.
- [OMG, 2005] Object Management Group: *Unified Modeling Language: Superstructure. Version 2.0. formal/05-07-04*. Available from: <http://www.omg.org/docs/formal/05-07-04.pdf>, 2004.
- [OMG, 2006] Object Management Group[homepage]. Available from: <http://www.omg.org>. Cited 2006.
- [OntoMed, 2006] OntoMed Research Group. Ontologies in Medicine. Foundations, Development and Computer-based Applications [homepage]. Available from: <http://onto-med.de>. Cited 2006.

- [OWL, 2004] *OWL Specifications*. World Wide Web Consortium (W3C). Available from: <http://www.w3.org/2004/OWL/>.
- [Pahl, Beitz, 1988] Pahl, G., Beitz, W.: *Engineering design - a systematic approach*. The Design Council, 1988.
- [Pease, Carrico, 1997] Pease, A., Carrico.: JTF-ATD Core Plan Representation: A Progress Report. *Proceedings of the AAAI-97 Spring Symposium on Ontological Engineering*, 1997.
- [Pease, Niles, 2002] Pease, A., Niles, I.: IEEE Standard Upper Ontology: A Progress Report. *The Knowledge Engineering Review*, 17(1):65-70, 2002.
- [Pegah et al., 1993] Pegah, M., Sticklen, J., Bond, W.: Functional Representation and Reasoning About the F/A-18 Aircraft Fuel System. *IEEE Expert*, 8(2): 65-71, 1993.
- [Poli, 2001] Poli, R.: The Basic Problem of the Theory of Levels of Reality. *Axiomathes*, 12(3-4):261-283, 2001.
- [Poli, 2002] Poli, R.: Ontological Methodology. *International Journal of Human-Computer Studies*, 56(6):639–664, 2002.
- [Preston, 1998] Preston, B.: Why is a wing like a spoon? A pluralist theory of function. *The Journal of Philosophy*, 95(5):215-254, 1998.
- [pUML, 2005] The precise UML group [home page]. Available at: <http://www.cs.york.ac.uk/puml/>. Cited 2005.
- [Qian, Gero, 1996] Qian, L., Gero, J.S.: Function-behaviour-structure paths and their role in analogy-based design. *AIEDAM* 10:289-312, 1996.
- [Quang, Chartier-Kastler, 1991] Quang, P.T., Chartier-Kastler, C.: *MERISE in practice*. MacMillan, London, 1991.
- [Rodenacker, 1971] Rodenacker, W.: *Methodisches Konstruieren*. Springer Verlag, Berlin, 1971.
- [Rosch, Mervis, 1975] Rosch, E., Mervis, C.: Family Resemblances: Studies in the Internal Structure of Categories. *Cognitive Psychology*, 7:573-605, 1975.
- [Rosenman, Gero, 1998] Rosenman, M. A., Gero, J. S.: Purpose and function in design, *Design Studies*, 19(2):161-186, 1998.
- [Rosenman, Gero, 1999] Rosenman, M., Gero, J. S.: Purpose and function in collaborative CAD environment. *Reliability Engineering and System Safety*, 64, p. 167-179, Elsevier, 1999.
- [Rumbaugh et al., 1991] Rumbaugh, J., Blaha, M. , Premerlani, W. , Eddy, F. , Lorensen, W.:

- Object-Oriented Modeling and Design*. Prentice Hall International, 1991.
- [Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*. Object Technology Series, Addison-Wesley, Reading, MA, 1999.
- [Russell, Norvig, 1995] Russell, S.J., Norvig, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New York, 1995.
- [Salustri, 1998] Salustri, F.: Function Modeling for an Integrated Framework: A Progress Report. *FLAIRS Conference 1998*, p. 339-343, 1998.
- [Sasajima et al., 1995] Sasajima, M., Kitamura, Y., Ikeda, M., Mizoguchi, R.: FBRL: A Function and Behavior Representation Language. *Proceedings of IJCAI-95*, p. 1830-1836, 1995.
- [Schlenoff et al., 2000] Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., and Lee, J.: *The Process Specification Language (PSL): Overview and Version 1.0 Specification*. NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD, 2000.
- [Schmekel, 1989] Schmekel H.: Functional models and design solutions. *Annals of the CIRP*, 38(1): 129-132., 1989.
- [Searle, 1995] Searle, J.R.: *The Construction of Social Reality*. Free Press. N.Y., 1995.
- [Sembugamoorthy, Chandrasekaran, 1986] Sembugamoorthy, V., Chandrasekaran, B.: Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. In Kolodner J., Riesbeck, C.K. (eds.) *Experience, Memory, and Reasoning*, p. 47–53, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.
- [Shimomura et al., 1995] Shimomura, Y., Takeda, H., Yoshioka, M., Umeda, Y., Tomiyama, T.: Representation of design objects based on the functional evaluation process model. *Proceedings of Design Theory and Methodology, ASME-95*, p. 351–360, 1995.
- [Shlaer, Mellor, 1988] Shlaer, S., Mellor, S.: *Object-Oriented Systems Analysis*. Yourdon Press, 1988.
- [Smartkom Project, 2005] Smartkom Project [homepage]. Available from: <http://www.smartkom.org>, cited 2005.
- [Smith et al., 2003] Smith, B., Williams, J., Schulze-Kremer S.: The ontology of the gene ontology. *AMIA Annu Symp Proc*, p. 609–613, 2003.
- [Smith, 1994] Smith, B.: Fiat Objects. In Guarino, N., Vieu, L. Pribbenow, S. (eds.) *Parts and Wholes: Conceptual Part-Whole Relations and Formal Mereology, 11th European*

- Conference on Artificial Intelligence*, p. 15–23, Amsterdam, August, 1994.
- [Smith, 1996] Smith, B.: Mereotopology: A Theory of Parts and Boundaries. *Data and Knowledge Engineering* 20:287–303, 1996.
- [Smith, Welty, 2001] Smith, B., Welty, C.: Ontology: Towards a New Synthesis. In Welty, C., Smith, B. (eds.) *Formal Ontology in Information Systems*. p.3-10, Ogunquit, Maine: ACM Press, 2001.
- [Sowa, 2000] Sowa J.: *Knowledge representation - Logical, Philosophical and Conceptual Foundations*. Brooks/Cole, Pacific Grove, USA, 2000.
- [Spyns et al., 2002] Spyns, P., Meersman, R., Jarrar, M. : Data Modelling versus Ontology Engineering. *SIGMOD Record*, 31(4): 12-17, 2002.
- [Stone, Wood, 2000] Stone, R., Wood, K.: Development of a Functional Basis for Design. *Journal of Mechanical Design*, 122(4): 359-370, 2000.
- [Studer et al., 1998] Studer, R., Benjamins, R., Fensel, D.: Knowledge Engineering: Principles and Methods. *Data & Knowledge Engineering*, 25(1-2): 161-197, 1998.
- [Subramanian, et al., 2005] Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., Mesirov, J. P.: Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci USA*, 102(43):15545–15550, October, 2005.
- [SUO, 2005] IEEE P1600.1 Standard Upper Ontology Working Group (SUO WG) [homepage]. Available from: <http://suo.ieee.org>, cited 2004.
- [Takeda et al., 1996] Takeda, H., Yoshioka, M., Tomiyama, T., Shimomura, Y.: Analysis of Design Process by Function, Behavior and Structure. In Cross, N., Christiaans, H., Dorst, K. (eds.) *Analysing Design Activity*, p. 187–209, Chichester, John Wiley & Sons, 1996.
- [Umeda et al., 1990] Umeda, Y., Takeda H., Tomiyama T., Yoshikawa H.: Function, Behavior and Structure. *Applications of AI in Engineering, AIENG'90*, p. 177-193, Computational Mechanic Publications and Springer Verlag, 1990.
- [Umeda et al., 1996] Umeda, Y. et al.: Supporting Conceptual Design Based on the Function-Behavior-StateModeler. *AIEDAM*, (10)4:275–288, Sept 1996.
- [Umeda, Tomiyama, 1995] Umeda, Y., Tomiyama, T.: FBS Modeling: Modeling scheme of function for conceptual design. *International Qualitative Reasoning Workshop Proceedings*, p. 71-77, University of Amsterdam, the Netherlands, 1995.

- [Umeda, Tomiyama, 1997] Umeda, Y., Tomiyama, T. (1997). Functional Reasoning in Design. *IEEE Expert*, 12(2): 42-48, 1997.
- [Upton, 2004] Upton C.L.: Book Review: Norms of Nature: Naturalism and the Nature of Functions, by Paul Sheldon Davies. *Essays in Philosophy – A Biannual Journal*, 5(1), January 2004.
- [Vermaas, Houkes, 2003] Vermaas, P.E., Houkes, W.: Ascribing Functions to Technical Artefacts: A Challenge to Etiological Accounts of Functions. *British Journal for the Philosophy of Science*, 54:261-289, 2003.
- [Vieu, Aurnague, 2005] Vieu, L., Aurnague, M.: Part-of Relations, Functionality and Dependence. In Aurnague, M., Hickmann, M., Vieu, L. (eds.) *Categorization of Spatial Entities in Language and Cognition*, John Benjamins, Amsterdam, 2005.
- [VonWright, 1963] VonWright, G. H.: *Norm and Action - A Logical Enquiry*. Routledge & Kegan Paul, New York, 1963.
- [W3C, 2006] *World Wide Web Consortium (W3C)*. [homepage]. Available from: <http://www.w3.org>. Cited 2006.
- [Wagner, 2002] Wagner, G.: A Uml Profile for External Agent-Object-Relationship (AOR) Models. In Giunchiglia, F., Odell, J., Weiß, G. (eds.) *Agent-Oriented Software Engineering*, p. 138–149, Lecture Notes in Computer Science, Vol. 2585, Springer Verlag, 2003.
- [Wand, 1999] Wand, Y., Storey, V. C., Weber, R.: An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions on Database Systems*, 24(4):494-528, 1999.
- [Welch, Dixon, 1992] Welch, R.V., Dixon, J. R.: Representing Functions, Behavior and Structure During Conceptual Design. *Design Theory and Methodology - DTM'92*, 42:11-18, *ASME*, 1992.
- [Welty, Guarino, 2001] Welty, C., Guarino, N.: Supporting Ontological Analysis of Taxonomic Relationships. *Data and Knowledge Engineering*, 39(1):51-74, 2001.
- [Wilkerson, 1995] Wilkerson, T.E.: *Natural Kinds*. Avebury Series in Philosophy, Aldershot, UK, 1995.
- [Wirth, O'Rorke, 1993] Wirth, R., O'Rorke, P.: Representing and reasoning about function for failure modes and effects analysis. *Reasoning about function, Workshop Preprints*, p. 188-194, *AAAI-93*, Washington DC, 1993.

- [WonderWeb, 2005] WonderWeb. Ontology Infrastructure for the Semantic Web Project [homepage]. Available from: <http://wonderweb.semanticweb.org>, cited 2005.
- [Woods, 1991] Woods, W.A.: Understanding subsumption and taxonomy: A framework for progress. In Sowa, J.F. (ed) *Principles of Semantic Networks*, Morgan Kaufman Publishers, 1991.
- [Wooldridge, Jennings, 1995] Wooldridge, M., Jennings, N.: Agent theories, architectures, and languages: A survey. In Wooldridge, M., Jennings, N. (eds.) *Intelligent Agents, ECAI-94*, Lecture notes in Artificial Intelligence, Vol. 890, Springer Verlag, 1995.
- [Wright, 1973] Wright, L.: Functions. *Philosophical Review*, 82:139-68, 1973.
- [Yourdon, 1993] Yourdon Inc., C.: *Yourdon Systems Method: Model-Driven Systems Development*. Yourdon Press, 1993.

Index of Symbols

$::$	Instantiation (from GFO)	$Fu_{Coh}(x)$	Coherent function
$At(y, t)$	Location at time boundary	$Fu_{Comp}(x)$	Complex function
$CatPart(x, y)$	Part, categorial (from GFO)	$Fu_{Contin}(x)$	Continuous function
$Cause(x, y)$	Cause (from OC)	$Fu_{Dyn}(x)$	Dynamic function
$Cause_{adh}(q, w)$	Processes, causally adhesive (from OC)	$Fu_{DynR}(x, y)$	Dynamic function wrt. a realizer y
$Cause_{coh}(x)$	Process, causally cohesive (from OC)	$Fu_{Enable}(x, y)$	Function enabling y
$Cause_{ins}(x, y)$	Cause, instantaneous	$Fu_{Improve}(x, y, z)$	Function x improving the realization z achieving the goal y
$Change(e_1, e_2, u_1, u_2, u)$	Change (from GFO)	$Fu_{Instant}(x)$	Instantaneous function
$Chron(x)$	Chronoid (from GFO)	$Fu_{MulGoal}(x)$	Multiple-goal function
$Coh(x)$	Coherent Entity (extends GFO)	$Fu_{Neutral}(x, y)$	Function neutral for y
$Coinc(x, y)$	Coincidence (from GFO)	$Fu_{Pass}(x)$	Passive function
$Config(x)$	Configuration (from GFO)	$Fu_{PassR}(x, y)$	Passive function wrt. a realizer y
$Configu(x)$	Configuroid (from GFO)	$Fu_{Perform}(x, y)$	Function performing y
$Contribute(x, y, z)$	x contributes to the realization y of a function z	$Fu_{Prevent}(x, y)$	Function preventing y
$CPart(x, y)$	Part, constituent (from GFO)	$Fu_{Seq}(x)$	Sequential function
$D(y, x)$	Determinant of a function x	$Fu_{Support}(x, y)$	Function supporting y
$Enable(x, y)$	Enable	$Fu_{Trigger}(x, y, z)$	Function x triggers the realization z achieving the goal y
$Entity(x)$	Entity (from GFO)	$GOAL(x)$	Set of goals of function x
$Exclude(v, w)$	Exclude	$Goal(x, y, z)$	x is a goal of a function y established by an agent z
$Execute(x, y, z)$	x executes the realization y of a function z	$GoalFor(x, y)$	x is a goal for an agent y
$Fact(x)$	Fact (from GFO)	$GoalOf(u, y)$	x is a goal of a function y
$Fl(x, y)$	Functional item of a function y	$HasFu_{Ac}(x, y, z)$	Actual function of an item x in context z
$Fl_{Comp}(x, y)$	Complex functional item of a function y	$HasFu_{Desig}(x, y, z)$	Designed function
$Fl_{Ind}(x, y)$	Individual functional item of a function y	$HasFu_{Disp}(x, y, z)$	Dispositional function
$FITEM(x)$	Set of functional items of x	$HasFu_{Inten}(x, y, z)$	Intended function
$FSt(x, y)$	x is a final state of y	$HasFu_{Req}(x, y, z)$	Required function
$Fu(x)$	Function	$HasFu_{Res}(x, y, z)$	Researched function
$Fu_{Accomp}(x)$	Accomplishment function	$HasFu_{User}(x, y, z)$	User function
$Fu_{Basic}(x)$	Basic function	$Improve(x, y, z)$	Function x improves the

	realization z of a function y		
$Ind(x)$	Individual (from GFO)	$ProcStarts(x,y)$	Process common start (from GFO)
$IndFu(x)$	Individual function		
$Intent(q,v)$	x intends y	$Prop(x)$	Property (from GFO)
$IntCont(i, R, a_1...a_n)$	Content of the intention i	$PropV(x)$	Property value (from GFO)
$LayerPart(x,y)$	Part, Process Layer (from GFO)	$Prt(x,c)$	Projection (from GFO)
$Lb(x,c)$	Left boundary (from GFO)	$R(x,y)$	Realizer of function y
$MalFu(x,y,z)$	Malfunction of x in context z	$R_{Ac}(x,y)$	Actual realizer of function y
$MalFu_{His}(x,f,s)$	Malfunction wrt. to the history of an item x	$Rb(y,c)$	Right boundary (from GFO)
$MalFu_{Inten}(x,f,c)$	Malfunction wrt. intended function of x	$R_{ComplAct}(x,y)$	Complex actual realizer of function y
$MalFu_{Kind}(x,f,s)$	Malfunction wrt. to other instances of a kind of x	$R_{Disp}(x,y)$	Dispositional realizer of function y
$MatL(x)$	x belongs to material stratum (from GFO)	$R_{DispStr}(x,y)$	Dispositional strong realizer
$Means_{ActRI}(x,y)$	Means of actual realization of a function y	$R_{Dyn}(x,y)$	Dynamic realizer of function y
$MentL(x)$	x belongs to mental stratum (from GFO)	$Realize(x,y)$	Function x realizes function y
$Occ(x)$	Occurrent (from GFO)	$REQ(x)$	Set of requirements of a function x
$Ontic(x, y)$	Ontological connectedness	$Req(x,y)$	Requirement of a function y
$Part(x,y)$	Part (from GFO)	$Req_{Env}(x,y)$	Environmental requirement of a function y
$Part_{Fu}(x,y)$	Function part-of	$Req_F(x,y)$	Functional item's requirement of a function y
$Part_{Seq}(x,y)$	x is a part of the sequence realizing y	$Req_{Op}(x,y)$	Operand requirements of a function y
$Perst(x)$	Persistent (from GFO)	$RI_{Act}(x,y)$	Actual realization of a function y
$PhPart(x,y)$	Part, physical (from GFO)	$RI_{ActCulm}(x,y)$	Actual culminative realization of a function y
$Phys(x)$	Physical Structure (from GFO)	$RI_{ActMin}(x,y)$	Actual minimal realization of a function y
$PPart(x,y)$	Part, Proper (from GFO)	$RI_{ActNonCulm}(x,y)$	Actual non-culminative realization of a function y
$Prb(x,t,y)$	Presential y being a projection of a process x to a time boundary t	$RI_{ActSit}(x,y)$	Actual situational realization of a function y
$Pres(x)$	Presential (from GFO)	$RI_{Disp}(x,y)$	Dispositional realization of a function y
$Prevent(x,y)$	Prevent		
$Proc(x)$	Process (from GFO)	$Role(x)$	Role (from GFO)
$ProcBd(x,y)$	Process boundary (from GFO)	$R_{Pass}(x,y)$	Passive realizer of a function y
$ProcEnds(x,y)$	Process common end (from GFO)	$SideEf(x,y)$	Side effects of a function y
$ProcLBd(x,y)$	Process boundary, left (from GFO)	$Seq(y, L)$	Sequence realizing a function y
$ProcPart(x,y)$	Part, processual (from GFO)	$SideEf_R(x,y,z)$	x is a side effect of realization y of a function z
$ProcRbd(x,y)$	Process boundary, right (from	$Sit(x)$	Situation (from GFO)

$Situ(x)$	Situoid (from GFO)	$z)$	
$SocL(x)$	x belongs to social stratum (from GFO)	$UniHasFu_{Inten}(x,y,z)$	Universal intended function
$SPart(x,y)$	Part, spatial (from GFO)	$UniR(x,y)$	Universal realizer of a function
$Specialize(x,y)$	Specialization	y	
$Subsume(x,x)$	Subsumption	$UniRI_{Min}(x,y)$	Minimal universal realizer of a function y
$Support(x,y)$	Support	$Whole(x)$	Complex Whole (extends GFO)
$Tb(x)$	Time boundary (from GFO)	$x \subset_{Fu} y$	Function specialization
$Te(x)$	Time entity (from GFO)	$x \subseteq_{Fu} y$	Function subsumption
$TFRAM(x)$	Time frame of x	$x ::_{FI} y$	Functional item instantiation
$TPart(x,y)$	Part, temporal (from GFO)	$x ::_{Fu} y$	Function instantiation
$Trig(x,y)$	x is a trigger of a function y	$x ::_{GI} y$	Goal instantiation
$Trigger(x,y,z)$	Function x triggers a realization z of function y	$x ::_{Req} y$	Requirements instantiation
$Uni(x)$	Universal (from GFO)	$x @ y$	x fulfills y
$UniD_{Ab}(x,y)$	Absolute universal determinant of function y	$x =_{Fi} y$	Equivalence of functions wrt. functional items
$UniFu(x)$	Universal function	$x =_{Fu} y$	Equivalence of functions
$UniFu_{Ab}(x)$	Absolute universal function	$x =_{GI} y$	equivalence of function wrt. their goals
$UniFu_{Prim}(x)$	Universal primitive function	$x =_{Req} y$	Equivalence of function wrt. their requirements
$UniHasFu(x,y,z)$	Universal has-function		
$UniHasFu_{Act}$	Universal actual function		
$UniHasFu_{Disp}(x,y,z)$	Universal dispositional function		

Scientific Career

- 2003 – 2006 PhD studies in the postgraduate programme *Knowledge Representation*, University of Leipzig, Germany.
PhD thesis: *Ontology of Functions: A Domain-independent Framework for Modeling Functions*.
Supervisor: Prof. Dr. Heinrich Herre.
- 2005 – now Member of the Ontologies in Biomedicine Group, University of Leipzig and Max Planck Institute for Evolutionary Anthropology.
- 2003 – now Member of the Onto-Med Research Group, University of Leipzig.
- 2001 - 2003 Business Consultant at Anica System S. A. Lublin, Poland: software engineering, system analysis and data modeling.
- 1997 – 2002 Master's Degree. Chair of Fundamentals of Computer Science, Department of Philosophy, Catholic University of Lublin, Poland.
Thesis: *Application of the Oracle CASE* Method for the analysis of the information system of the Income Evidence Department at the Catholic University of Lublin: Modeling of functional and informational requirements*.
- 2000 – 2002 Institute of Marketing and Management, Department of Economics, Catholic University of Lublin, Poland (5 semesters).
- 1997 University-entrance diploma at Zamoyski High school, Lublin, Poland.
Main subjects: Physics and Mathematics.

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Dissertation selbstständig und ohne unzulässige fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die zugelassenen Quellen und Hilfsmittel benutzt und sämtliche Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurde als solche kenntlich gemacht. Ebenfalls sind alle von anderen Personen bereitgestellten Materialien oder erbrachten Dienstleistungen als solche gekennzeichnet.

Leipzig, 20.7.2006

Patryk Burek