Towards a general ontology of computer programs

Pascal Lando, Anne Lapujade, Gilles Kassel, Frédéric Fürst

MIS, Jules Verne University of Picardie, 33 rue Saint Leu, F-80039 Amiens, France
pascal.lando@u-picardie.fr, anne.lapujade@u-picardie.fr,
gilles.kassel@u-picardie.fr, frederic.furst@u-picardie.fr

Abstract. Over the past decade, ontology research has extended into the field of computer programs. The work has sought to define conceptual descriptions of the programs and thus master the latter's design and use. Unfortunately, these efforts have only been partially successful. Here, we present the basis of a Core Ontology of Programs and Software (COPS) which integrates the field's main concepts. But, above all, we emphasize the method used to build the ontology. Indeed, COPS specializes the DOLCE foundational ontology ("Descriptive Ontology for Linguistic and Cognitive Engineering", Masolo et al., 2003) as well as core ontologies of domains (e.g. artefacts, documents) situated on a higher abstraction level. This approach enables us to take into account the "dual nature" of computer programs, which can be considered as both syntactic entities (well-formed expressions in a programming language) and artefacts whose function is to enable computers to process information.

Keywords. Knowledge engineering, ontological engineering, foundational ontologies, core ontologies, program ontologies.

1 Introduction

Over the last ten years or so, the field of computing programs has witnessed an increasing number of ontological investigations in several disciplines and with different objectives: in the philosophy of computer science, the goal is to attain better knowledge of the nature (Eden & Turner, 2007) and semantics of computer programs (Turner & Eden, 2007); whereas in software engineering, formal descriptions seek to facilitate program maintenance (Welty, 1995, Oberle *et al.*, 2006). Such descriptions can also be used to orchestrate and automate the discovery of web services (Roman *et al.*, 2005).

Our current work is in line with these efforts. In a first step it seeks to build a general or "core" ontology (Gangemi & Borgo, 2004) of the domain of programs & software, and which will encompass the latter's main concepts and relations. This core ontology, (named COPS for "Core Ontology of Programs and Software"), will be used in a second step to conceptualize a sub-domain of computer programs, namely that of image processing tools. This step takes place within the project NeuroLOG (http://neurolog.polytech.unice.fr) which aims at developing a distributed software platform to help members of the neuroimaging community to share images and image processing programs. This platform relies on an ontology integrating COPS as a component (Temal *et al.*, 2006).

In this paper, we present not only our current content for COPS but more generally the methodological process that we used to build the ontology and the resulting structural features. The COPS ontology indeed specializes more abstract modules which strongly determine its structure, including the DOLCE foundational ontology ("Descriptive Ontology for Linguistic and Cognitive Engineering", Masolo *et al.*, 2003) and the I&DA core ontology ("Information and Discourse Acts", Fortier & Kassel, 2004). This type of design process is aimed at mastering two sorts of complexity (i) *conceptual* complexity, providing the ability to model complex objects (such as programs) at different abstraction levels and (ii) *modeling* complexity, providing the re-use of previously used & approved modules and also, the ability to design new modules by working in a distributed manner.

Two syntactic manifestations of COPS exist: one is coded in the semi-formal language from the OntoSpec method (Kassel, 2005) and the other is coded in the formal web ontology language OWL. Within the NeuroLOG project, this latter expression is implemented into software under development that includes a semantic research tool called CORESE (designed as part of the ACACIA project at the INRIA institute, http://www.inria.fr/acacia/corese). However, due to space restrictions, we shall disregard these syntactic aspects in this paper and focus on the ontology's content.

The remainder of this article is structured in the following way. In section 2, we present the overall ontological framework of reference used (and the DOLCE and I&DA ontologies in particular). Section 3 details the COPS ontology's content and our design choices. In section 4, our work is compared with other efforts to design ontologies in the domain of computer programs.

2 Our Ontological Framework

As shown in Figure 1, the COPS ontology is integrated into a larger ontology composed of sub-ontologies which are situated at different levels of abstraction: a descending link between two sub-ontologies O_1 and O_2 means that the conceptual entities (concepts and relations) of O_2 are defined by specialization of the conceptual entities of O_1. The DOLCE foundational ontology and the different core ontologies make up the resource used by the OntoSpec methodology (http://www.laria.u-picardie.fr/IC/site/) to help structure application ontologies (which include all the concepts necessary for a particular application), such as that developed within the NeuroLOG project.

As a consequence of this overall structure, COPS's conceptualization depends on modeling choices made upstream, i.e. in components situated at a higher level of abstraction. In this section, we present these key modeling choices by successively introducing the DOLCE ontology (2.1), the modeling of (participant) roles & artefacts (2.2) and the I&DA ontology (2.3).

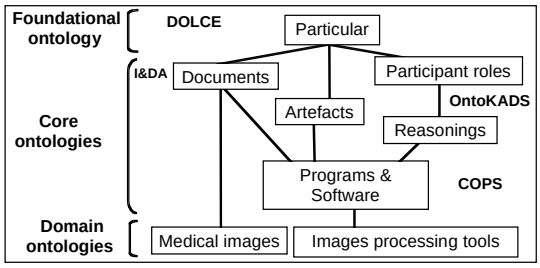

Fig. 1. Structure of the application ontology in the NeuroLOG project.

2.1 DOLCE (Particulars)

DOLCE is a "foundational" ontology, which means that it comprises abstract concepts aimed at generalizing the set of concepts that we may encounter in the different domains of knowledge. In accordance with philosophically-grounded principles, DOLCE's domain – that of *Particulars* – is partitioned into four sub domains (*cf*. Fig. 2):

- *Endurants* are entities "enduring in time" (e.g. the present article). Within *Endurants*, *Physical Objects* are distinguished from *Non-Physical Objects*, since only the former possess direct spatial *Qualities*. The domain of *Non-Physical Objects* covers social entities (e.g. the French community of researchers in knowledge engineering) and cognitive entities (e.g. your notion of knowledge engineering). To take plural entities into account (a persons in a community or the proceedings of a conference), the notion of *Collection* was recently introduced under *Non-Physical Objects* (Bottazzi *et al.*, 2006).
- *Perdurants* are entities which "happen in time" (e.g. your reading of this article) and in which *Endurants participate*. Among *Perdurants*, one defines *Actions* that are intentionally accomplished (*Accomplishments*), i.e. controlled by an *Agent* (further defined in 2.2).
- *Endurants* and *Perdurants* have inherent properties (*Qualities*) that we perceive and/or measure (e.g. the weight of the paper copy of the article you may be holding or how long it takes you to read this article).
- These *Qualities* take a value (*Quale*) within regions of values which are *Abstracts* (e.g. 20 grams, 15 minutes).

These concepts are defined in DOLCE by means of rich axiomatization, which space restrictions prevent us from presenting. In particular, *Endurants* and *Perdurants* can be differentiated in terms of the dissimilar temporal behaviors of their parts. The interested reader is invited to refer to (Masolo *et al.*, 2003).

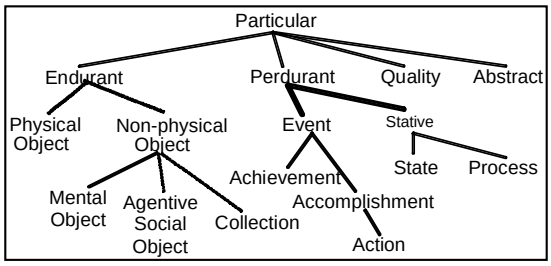

Fig. 2. An excerpt from DOLCE's hierarchy of concepts.

Comment: in the following sections, names of conceptual entities will be noted in italics, with first *Capital Letters* for concepts and in a *javaLikeNotation* for relations.

2.2 Roles and Functions

In this section, we introduce two important notions linking *Endurants* to *Perdurants* (namely *roles* and *functions*) and we provide a brief reminder of the underlying modeling choices (Bruaux *et al.*, 2005).

A *role* (or more exactly a "participant role" or "thematic role") accounts for the the behaviour of *Endurants* when *participatingIn* (as defined by DOLCE) *Perdurants*. By way of an example, during the writing of an article, several entities participate in this *Action*: a person as an *Agent*, a pencil or a pen as an *Instrument* and the article itself as a *Result*. Here, the term " participant role" designates a category of concepts (e.g. *Agent, Instrument, Result*) which constitute a sub-ontology of *Endurants*, since the nature of the *participation* relation of DOLCE constraints participants to be *Endurants* (cf. Fig. 3a).

A *function* can be defined as the ability – assigned by agents to *Endurants* – to facilitate the performance of an *Action*, i.e. the ability of playing the role of *Instrument* in a *Perdurant*; in turn, this notion enables definition of the concept of an *Artefact* - an *Endurant* to which a function is assigned. According to the type of *Action* (the sub-ontology of *Actions* on which COPS relies is discussed in 3.2), different types of *Artefacts* can be distinguished (cf. Fig. 3b): *Tools* are distinguished from *Cognitive Artefacts* according to whether the *Action* they can perform corresponds to modification of the physical world or the non-physical world. Of the latter, *Artefacts of Communication* enable communication of information to agents, whereas *Artefacts of Computation* allow computers to perform *Actions* as *Agents*.

In Figure 3, one can note that the concepts of *Author* and *Scientific Publication* encapsulate the type of entity and, respectively, the role and function assigned to the entity. This modeling choice, which is consistent with the most common paradigm for role modeling (Steimann, 2000), leads to a tangled taxonomy.

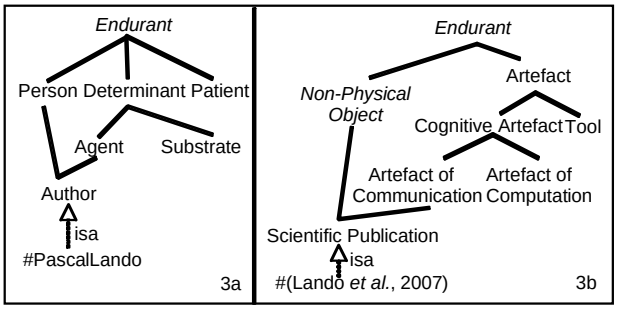

Fig. 3. Modeling of roles (a) and functions-artefacts (b).

2.3 I&DA (*Inscriptions, Expressions and Conceptualizations*)

I&DA is a core ontology in the domain of semiotics, and was initially built to classify documents according to their contents.

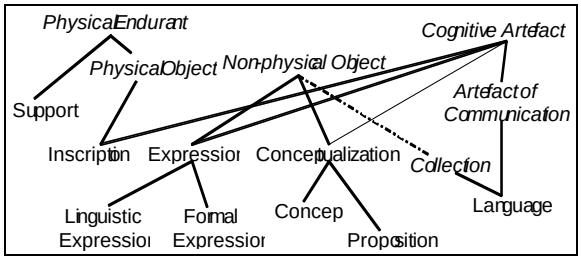

Fig. 4. The top level of I&DA's hierarchy of concepts.

As shown in Figure 4, I&DA extends DOLCE by introducing three main concepts:

- *Inscriptions* are knowledge forms materialized by a substance and inscribed on a physical *Support* (e.g. a written text materialized by some ink on a sheet of paper). Furthermore, these forms are intentional objects, which hold for other entities: *Inscriptions realize Expressions*.
- *Expressions* are non-physical knowledge forms *orderedBy* a *Language*. In their turn, *Expressions* hold for other entities, namely contents that agents attribute to them: *Expressions express Conceptualizations*.
- Lastly, *Conceptualizations* are means by which agents can reason about a world. Within *Conceptualizations*, a functional distinction is made between *Propositions* (which are descriptions of situations) and *Concepts* (which serve to classify entities in a world).

The reader will note that, in order to account for documents, I&DA chooses to consider three distinct entities rather than three different views of the same entity. We shall see in the next section that this modeling choice has important repercussions on the structure of COPS.

3 COPS : a Core Ontology of Programs and Software

The COPS ontology indeed classifies a program as a document whose main characteristic is to allow a computer to perform information processing. In section 3.1, we first show how the ontological framework presented so far contributes to the definition of these particular documents. In sections 3.2 to 3.5, we then present several sub-ontologies dedicated to the different aspects of the notion of "program".

3.1 The Dual Nature of Programs

Firstly, the distinction made in DOLCE between *Endurants* and *Perdurants* prompts us to distinguish between the program (as an *Endurant*) and its executions, which are *Perdurants*. Secondly, when focusing on the program as an *Endurant*, the distinctions in I&DA between *Inscriptions*, *Expressions* and *Conceptualizations* prompt us to consider three categories of entities commonly referred to as "programs" (*cf.* Fig. 5):

- *Files*, which are *Inscriptions* written on an electronic support (e.g. CDs, computer memory, magnetic tape, etc.). Furthermore, these files constitute only one type of program *Inscription*; a paper listing or an on-screen display of a program are also program *Inscriptions*.
- *Computer Language Expressions*, which are well-formed formulas (*isAWellFormedFormula Of* is a sub-relation of *isOrderedBy*) in a *Computer Language*. These expressions include *Programs*.
- *DataTypes* and *Algorithms*, which are *Conceptualizations* that represent the semantics of *Programs*. *DataTypes* are *Concepts* on which rely programming languages (e.g. variable, class, structure) and which are reflect to diversity of programming languages (Turner & Eden, 2007). *Algorithms* describe calculus steps in terms of these *DataTypes* (e.g. affecting a constant to the value of a variable, then adding another value, etc.).

This approach boils down to considering programs as *Expressions*, which is a consensual point of view in both computer science and philosophy. However, we consider that this purely syntactical description of a program is not enough to fully capture the nature of programs.

Indeed, programs have also a functional dimension, in that they allow computers to perform *Actions* (*Computations*). This functional dimension is present in expressions such as "sort program", "program for calculating the greatest common divisor of two numbers" or "image processing program". Programs are therefore also *Artefacts of Computation* (cf. Fig. 5). As commonly proposed in philosophy for the characterization of artefacts (Kroes & Meijers, 2002), we therefore end at a dual characterization of programs, considered to be both *Computer Language Expressions* and *Artefacts of Computation.*

In order to account for these dimensions of programs (and refine them), COPS proposes a sub-ontology of *Actions* (*cf.* 3.2) and a sub-ontology of *Languages* (*cf.* 3.3). We shall see in 3.4 that COPS's concept of *Program* integrates complementary constraints with regard to this first characterization.

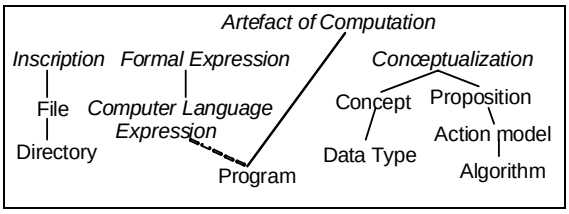

Fig. 5. The general structure of COPS.

3.2 A Sub-ontology of Actions

In order to refine the functional dimension of *Programs* (and *Computer Language Expressions* in general) and specify what these latter allow a person or a computer to perform, COPS is endowed with an ontology of *Actions* (*cf.* Fig. 6).

On one level, *Actions* are distinguished according to (i) the world in which the transformation-*Action* occurs (physical (*Doing*) or non-physical (*Non-physical Action*)) and (ii) the *Agent* performing the *Action* (a human (*Human Action*) or a computer (*Computational Action*)).

The first semantic axis relies on a strong hypothesis dealing with the identity criteria of *Actions*, namely that *Actions* performed in separate worlds of entities are themselves distinct *Actions*. The worlds of entities considered in COPS converge with the common hierarchy of computer description levels, to which we add the "knowledge level" postulated by Newell (1982). This hypothesis prompts specialization of *Non-physical Actions* into *Symbolic Actions* (which, at the symbolic level, consist in transforming *Expressions* - e.g. *Executing a Program*, *Compiling a Program*) and *Conceptual Actions* (which, at the knowledge level, consist in transforming *Conceptualizations*). Of the latter, *Actions* involving *knowledge models* and which are taken into account by the CommonKADS methodology (Bruaux *et al.*, 2005) (e.g. *Diagnosing a car's failure*, *Monitoring a patient*) are distinguished from *Actions* performed on *Data* and *Data Types* constituting the paradigms generated by the different programming languages (e.g. *Incrementing a Variable*, *Creating an Object*). The reader should note (cf. Fig. 6) that *Discourse Acts* are *Non-physical Actions*. The latter (considered as *Actions* which lead to a change in the state of knowledge of the addressee of the discourse) are used in COPS to account for *Actions* such as *Requests* (e.g. querying databases) or *Orders* for executing *Programs*.

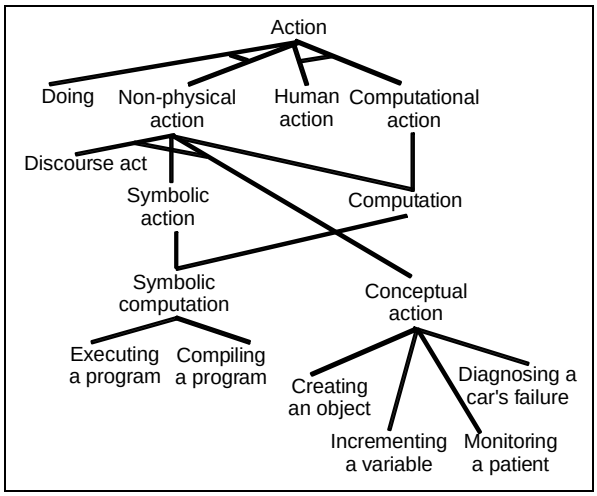

Fig. 6. The sub-ontology of *Actions* in COPS.

3.3 A Sub-ontology of Computer Languages

In order to refine the syntactical dimension of *Expressions*, COPS includes an ontology of computer languages. Classically, one distinguishes between natural languages and formal languages. The formal languages of interest here are C*omputer Languages*, *i.e.* those designed for interpretation by a computer (microprocessor) or a program. Our conceptualization of *Computer Languages* (*cf.* Fig. 7) is based on the functions (the artefactual dimension) of the *Expressions* that they can *order*. The first category of computer languages is that of *General Purpose Computer Languages* (*GPCLs*), *i.e.* Turing-complete languages dedicated to the writing of all kinds of programs. The second category is that of *Domain-Specific Computer Languages* (*DSCLs*), *i.e.* non-Turing-complete languages limited to the writing of particular types of expressions (database queries, operating system commands, etc.). *Programming languages* are all *GPCLs* that are understandable by humans. *GPCLs* that are only understandable by computers or programs are *Low-level Computer Languages* (or low-level programming languages): *Machine Languages* (understandable by a processor) and *Byte-code Languages* (understandable by a virtual machine).

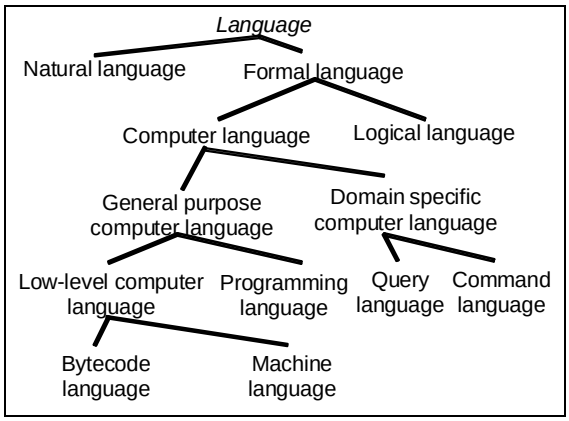

Fig. 7. The sub-ontology of computer languages in COPS.

3.4 A Sub-ontology of Computer Language Expressions and Programs

The sub-ontology of *Expressions* (all of which are considered here to be well-formed *Expressions*) mirrors the sub-ontology of *Languages* that *order* these *Expressions*. The structure of these two taxonomies is similar (*cf.* Fig. 8) and is based on the functional dimension. An *Expression* in a *GPCL* (*General Purpose Computer Language Expression*) allows a computer to perform an arbitrary *Computation* (e.g. declaring a variable, calculating the greatest common divisor of two numbers). By contrast, an *Expression* in a *Query Language* (*Query Expression*) or an *Expression* in a *Command Language* (*Command Expression*) are functionally different: they do not allow computers to performing a *Computation* but allow (human) users to *Ordering* (which is a kind of *Discourse Act*) the performance of particular *Computations*, such as, for example, querying or modifying a set of data. These *Expressions* are therefore *Artefacts of Communication* and this functional distinction has repercussions for the definition of COPS's concept of *Program*.

On one side, we consider that a *Program* syntactically corresponds only to a particular type of *Expressions orderedBy* a *Programming Language*. Indeed, the peculiarity of *Expressions* qualified as *Programs* is that they can be either directly executed by a computer (after a compilation) or taken in charge by an interpreter. As an example, a program in the language C is composed of one or more functions, one of these functions being necessarily called "main". By contrast, *Expressions* such as a function or an instruction do not possess this entry point rendering the *Expression* executable or interpretable.

On another side, we consider that there exist executable or interpretable *Expressions* which are not *Programs*. Indeed, in the same lines as Eden and Turner (Eden & Turner, 2007), we only speak of *Programs* as *Expressions* being *orderedBy* Turing-complete languages (or *General Purpose Programming Languages*). Hence COPS does not consider a SQL query or a shell command, which are yet interpretable or executable, to be *Programs*. To sum up, we define a *Program* as an *Expression* in a Turing-complete language which can be interpreted or compiled and executed by an *Operating System* (*Executable Program*) or a *Virtual Machine* (*Byte-code Program*).

In addition, crosscutting relations link the different types of *Programs*: a *Source Code hasForExecutable* (or "can be compiled into several") *Executable programs*, and conversely an *Executable Program* or a *Byte-code Program hasForSourceCode* a *Source Code*.

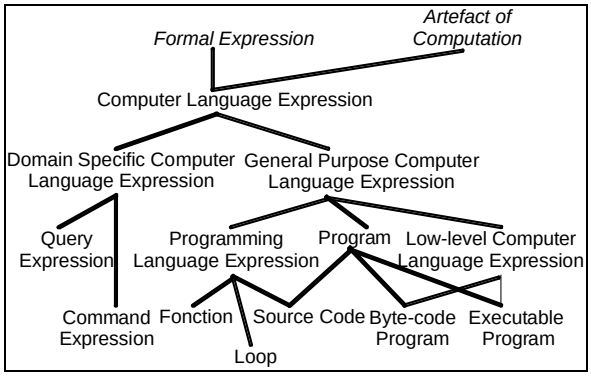

Fig. 8. A sub-ontology of *Programs* in COPS.

Note that *Operating Systems* or *Virtual Machines* do not appear in COPS's sub-ontology of *Programs*. The reason is that they are not *Programs*, but they are rather made up of a set of *Programs*. They are therefore related to software, which are defined in COPS as collections of *Programs*, as presented in the next section.

3.5 A Sub-ontology of Software and Platforms

This sub-ontology of COPS models entities that are collections of *Programs* rather than single *Programs*. The concept *Library of Programs* (*cf.* Fig. 9) designates a *Collection* of *Programs* and, potentially, other documents (such as manuals).

By analogy with *Program*, *Software* is defined as both a *Library of Programs* and an *Artefact of Computation*. Since it must be executable, S*oftware* includes at least one *Executable* (or *Interpretable*) *Program*. *Software* includes *Compilers* (whose function is to allow a computer to translate a *Source Code* into an *Executable program)*, *Interpreters* (whose function is to allow a computer to execute a *Source Code*) and *Operating Systems* (whose function is to allow a computer to execute *Executable Programs*). This function defines another class of *Artefacts* - the *Platforms*.

A *Platform* can be a purely material entity (*Hardware Platform*) or an entity that is partially made up of *Software* (*Software Platform*). *Software Platforms* include *Operating Systems* and *Computers* on which *Operating Systems* run.

Crosscutting relations link *Programs* and particular types of *Software*: a *Source Code isCompilableBy* particular *Compilers* and/or *isInterpretableBy* particular *Interpreters*; an *Executable Program runsOn* a particular *Operating System*; a *Byte-code Program runsOn* a particular V*irtual Machine*.

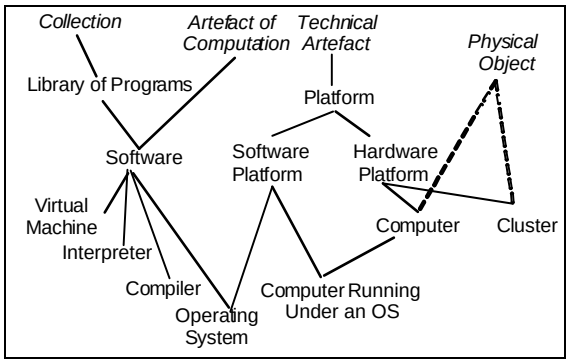

Fig. 9. The sub-ontology of *Software* and *Platforms* in COPS.

4 Discussion

In this section, we compare COPS with other attempts to elaborate ontologies of programs.

In the philosophy of computer science, Eden & Turner (2007) recently began creating an ontology of programs in order to answer questions like: how can we display differences between hardware and software entities, and how can we distinguish a program from a program specification? Even though the ontological tools used in the present work are different from Eden & Turner's, it is interesting to compare the respective conceptualizations. For example, Eden & Turner (2007) define a program as a "well-formed expression in a Turing-complete programming language" - emphasizing the syntactical dimension of programs but setting aside their functional dimension (which exists in the COPS concept of *Program*). The fact that these two notions differ suggests that it would be useful to extend the COPS core ontology to other concepts.

The "web services" community has generated a variety of initiatives – METEOR-S, OWL-S, WSMO (Roman *et al.*, 2005) – which seek to formally describe the discovery, evocation and orchestration (at different levels of automation) of such services. These efforts are currently far removed from COPS' aims because (i) the work emphasizes the operational nature of the descriptions and (ii) these descriptions concerning the function (the *Action* in COPS) realized by the service (e.g. booking a travel ticket) are situated on a meta level, which allows definition of the prerequisites for operation of the service (*e.g.* information about the travel has to be given) and the effects resulting from its execution (*e.g.* the ticket price is debited from a bank account). Within the framework of the NeuroLOG project, the functionalities targeted in terms of the evocation and orchestration of software tools are similar, which is why we plan to extend COPS to consider this level of description.

In the software engineering domain, Welty (2005) has suggested developing Comprehensive Software Information Systems (for software maintenance) by using an ontology which enables a detailed, conceptual description of software. This ontology could be considered as an extension of the COPS *Expression* sub-ontology, as it enables description at the code level and consideration of all the syntactical

constructions available in programming languages. On the other hand, it supposes (strangely) that the entities playing *data* and *result* roles are real world entities (*e.g.* persons) and not conceptualizations modeling the real world. In COPS, we chose to follow (Turner & Eden's, 2007) idea whereby program semantics are based on data and data types which model real world entities - for example (in the object paradigm), an *instance* which models an individual person or a *class* which models a set of persons.

Other work in the software engineering domain (Oberle *et al.*, 2005) led to publication of the Core Software Ontology (CSO) in order to better develop, administer and maintain complex software systems. The ontology-building approach is similar to ours, with re-use of the DOLCE high level ontology and core ontologies such as DnS ("Descriptions & Situations", Gangemi & Mika, 2003). COPS and CSO also share some modeling choices, such as the distinction between three entities (called *Inscriptions*, *Expressions* and *Conceptualizations* in COPS). However, we can note some different modeling choices. For example, in CSO (and assuming that every program can be a data item for another program), the *Data* concept subsumes the *Software* concept. In contrast, COPS assimilates the *Data* concept to a participant role (*cf.* 2.2) which can be played by arbitrary entities - *Programs*, for example. In fact, whereas CSO considers only one type of *Action* (namely "computational activities" whose participants are necessarily *Inscriptions* (in the sense of COPS) inscribed on some sort of hardware), COPS distinguishes several categories of *Actions* according to the nature of the participant entities (*cf.* 3.2). COPS' richer framework allows it to define a *Program Compilation* as an *Action* in which at least two *Programs* participate. Lastly, we can note that the functional dimension of programs is absent in CSO.

Those comparisons show that other core ontology proposals for the software domain do exist but that (i) the various efforts have not been coordinated and (ii) the existing ontologies display some important differences in terms of both range and structure.

5 Conclusion

In the present paper, we have presented the foundations of a core ontology of programs and software (COPS) derived by specializing the DOLCE foundational ontology and whose goal is to help structure more specific programming domains. In this connection, the next application of COPS within the NeuroLOG project, to help conceptualizing the domain of image processing tools, will provide an opportunity for evaluating the modeling choices made for the building of the ontology.

COPS' current conceptualization reveals a domain populated by entities having various nature. Indeed, there are temporal entities (program executions), physical entities (program inscriptions), plural entities (program collections), functional entities (program execution platforms) and, lastly, dual-nature (syntactic and functional) entities - the programs themselves. COPS' model-building feedback confirms the fact that ontological resource re-use (enabling modeling choices at several abstraction levels) is necessary for controlling the complexity of such domains.

In its current version, COPS only covers a part of this domain. Work in process is extending the ontology in several directions. A first goal is to extend the programs

semantics: links with processing (functions) only give an account of the "what", so it lacks the "how" - requiring us to take into account algorithms and data types which have only been positioned (*cf.* Fig. 5) and not precisely analyzed. A second goal is to enlarge COPS to program specifications: we plan to re-use the "problem resolution model" notion in OntoKADS (Bruaux et al., 2005) to extend COPS to the more general class of action models performed by computers using programs.

Acknowledgements

This work was funded in part by the NeuroLOG project (ANR-06-TLOG-024) under the French National Research Agency's Software Technologies program (http://neurolog.polytech.unice.fr).

References

Bottazzi E., Catenacci C., Gangemi A. & Lehmann J.: From Collective Intentionality to Intentional Collectives: an Ontological Perspective. In: Cognitive Systems Research, Special Issue on Cognition, Joint Action and Collective Intentionality, 7(2-3), p. 192-208 , Elsevier (2006).

Bruaux S., Kassel G. & Morel G.: An ontological approach to the construction of problem-solving models. In P. Clark and G. Schreiber (eds), In: 3rd International Conference on Knowledge Capture (K-CAP 2005), p. 181-182, ACM (2005). A longer version is published as LaRIA's Research Report 2005-03. Available at: http://hal.ccsd.cnrs.fr/ccsd-00005019 .

Eden A. H. & Turner R.: Problems in the Ontology of Computer Programs. In : *Applied Ontology* 2(1), p. 13-36 (2007).

Fortier J.-Y. & Kassel G.: Managing Knowledge at the Information Level: an Ontological Approach. In: Proceedings of the ECAI'2004 Workshop on Knowledge Management and Organizational Memories, p. 39-45, Valencia, Spain (2004).

Gangemi A. & Borgos S. (eds): Proceedings of the EKAW'04 Workshop on Core Ontologies in Ontology Engineering, Northamptonshire (UK). From http://ceur-ws.org (Vol-118) (2004).

Gangemi A. & Mika P.: Understanding the Semantic Web through Descriptions and Situations. In: R. Meersman et al. (eds), Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2003), Catania (Italy), (2003).

Kroes P. & Meijers A.: The Dual Nature of Thechnical Artifacts – presentation of a new research programme. In: Techné, 6(2), p. 4-8 (2002).

Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A. & Schneider L.: The WonderWeb Library of Foundational Ontologies and the DOLCE ontology. In: WonderWeb Deliverable D18, final report (vr. 1.0, 31-12-2003) (2003).

Newell A.: The Knowledge Level. In: Artificial Intelligence 18, p. 87-127 (1982).

Niles I., Pease A.: Towards a standard upper ontology. In: Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'2001). ACM Press, p 2-9 (2001).

Oberle D., Lamparter S., Grimm S., Vrandecic D., Staab S. & Gangemi A.: Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems. In: Applied Ontology, 1(2), p. 163-202 (2006).

Roman D., Keller U., Lausen H., de Bruijn J., Lara R., Stollberg M., Polleres A., Feier C., Bussler C. & Fensel D.: Web Service Modeling Ontology. In: Applied Ontology 1, p. 77-106 (2005).

Steimann F.: On the representation of roles in object-oriented and conceptual modelling. In: Data and Knowledge Engineering, 35, p. 83-106 (2000).

Temal L., Lando P., Gibaud B., Dojat M., Kassel G. & Lapujade A.: OntoNeuroBase: a multi-layered application ontology in neuroimaging. In: Proceedings of the 2nd Workshop: Formal Ontologies Meet Industry: FOMI 2006, Trento (Italy) (2006).

Turner R. & Eden A.H.: Towards a Programming Language Ontology. In: G. Dodig-Crnkovic & S. Stuart (eds.), Computation, Information, Cognition – The Nexus and the Liminal, Cambridge, UK: Cambridge Scholars Press, chapter 10, p. 147-159 (2007).

Welty C.: An Integrated Representation for Software Development and Discovery. Ph.D. Thesis, RPI Computer Science Dept. July 1995, From *http://www.cs.vassar.edu/faculty/welty/papers/phd/* (1995)