Premiers pas vers une ontologie générale des programmes informatiques[1]

Pascal Lando, Frédéric Fürst, Gilles Kassel et Anne Lapujade

Laboratoire de Recherche en Informatique d'Amiens,
Université de Picardie Jules Verne
33 rue Saint Leu, 80039 Amiens
{pascal.lando,frederic.furst,
gilles.kassel,anne.lapujade}@u-picardie.fr

Résumé : Le domaine des programmes informatiques fait l'objet depuis plusieurs années d'investigations ontologiques, l'enjeu étant de disposer de descriptions conceptuelles des programmes pour mieux maîtriser leur conception et leur utilisation. Il s'avère toutefois que les efforts entrepris jusqu'à présent n'ont permis d'aboutir qu'à des résultats partiels. Dans cet article, nous présentons les bases d'une ontologie générale des programmes (COPS) intégrant des concepts et relations centraux de ce domaine. Au-delà du contenu lui-même de l'ontologie, l'article met l'accent sur la méthode retenue pour sa construction. L'ontologie spécialise ainsi l'ontologie formelle de haut niveau DOLCE ainsi que des ontologies générales d'autre domaines (ex. : documents, artefacts) situées à des niveaux d'abstraction plus élevés. Cette démarche nous permet de rendre compte de la nature duale des programmes en les assimilant tout à la fois à des entités *syntaxiques*, des expressions formulées dans un langage de programmation, et à des *artefacts* dont la fonction est de permettre à des ordinateurs de réaliser des traitements de l'information.

Mots-clés : Ontologies formelles et de haut niveau (foundational ontologies), Ontologies « noyaux » (core ontologies), Ontologies de programmes.

1 Introduction

Depuis une dizaine d'années, le domaine des programmes informatiques fait l'objet d'un nombre croissant d'investigations ontologiques visant, selon les disciplines, différents objectifs : en philosophie de l'informatique, l'objectif est d'accéder à une meilleure connaissance de la nature des programmes (Eden & Turner, 2006) et de leur sémantique (Turner & Eden, sous presse) ; en génie logiciel, la description formelle des programmes vise à aider leurs développeurs à les concevoir et à les maintenir (Welty, 1995)(Oberle *et al.*, 2006) ou, pour les services sur le Web, à automatiser leur découverte et leur combinaison (Roman *et al.*, 2005).

Le travail présenté dans cet article se situe dans la lignée de ces efforts. Il vise, dans un premier temps, à concevoir une ontologie générale, ou « noyau » (Gangemi &

[1] Ce travail est en partie financé dans le cadre du projet NeuroLOG (ANR-06-TLOG-024) du programme Technologies Logicielles de l'Agence Nationale de la Recherche : http://neurolog.polytech.unice.fr.

Borgo, 2004), du domaine des programmes et des logiciels, rassemblant des concepts et relations centraux de ce domaine. Cette ontologie noyau, nommée COPS (pour : Core Ontology of Programs and Software), doit par la suite être utilisée pour conceptualiser un sous-domaine des programmes informatiques, celui des outils de traitement d'images. Cette seconde étape s'inscrit dans le cadre du projet NeuroLOG[1] visant à développer une plate-forme logicielle distribuée pour aider la communauté des chercheurs et cliniciens en neuro-imagerie à mutualiser des images et des programmes de traitement d'images. Cette plate-forme repose sur une ontologie intégrant COPS comme composant (Temal *et al.*, 2006).

Dans cet article, nous exposons le contenu courant de COPS mais, surtout, nous nous plaçons sur un plan méthodologique pour présenter la démarche de construction de COPS et la structuration de l'ontologie qui en résulte. L'ontologie COPS spécialise ainsi des modules plus abstraits qui conditionnent sa structuration, notamment l'ontologie de haut niveau DOLCE – Descriptive Ontology for Linguistic and Cognitive Engineering – (Masolo *et al.*, 2003) et l'ontologie générale de documents I&DA – Information and Discourse Acts – (Fortier & Kassel, 2004). Une telle démarche de conception vise à maîtriser deux complexités : i) une complexité *conceptuelle*, en permettant de modéliser des objets complexes comme le sont les programmes informatiques à différents niveaux d'abstraction ; ii) une complexité de *conception*, en permettant de réutiliser des modules déjà utilisés et évalués dans le cadre d'autres ontologies d'application, et, également de travailler de façon distribuée à la conception de nouveaux modules.

Deux manifestations syntaxiques de COPS existent, correspondant à la spécification de l'ontologie respectivement dans le langage semi-informel de la méthode OntoSpec (Kassel, 2005) et dans le langage RDF Schema[2]. Dans cet article, toutefois, pour des raisons de place, nous faisons abstraction de ces aspects syntaxiques pour nous focaliser sur le contenu de l'ontologie.

La suite de l'article est structurée comme suit. La section 2 présente le cadre ontologique de référence utilisé, en introduisant notamment les deux ontologies DOLCE et I&DA. La section 3 détaille le contenu de l'ontologie COPS et les choix de conceptualisation effectués. Dans la section 4, nous comparons notre travail à d'autres efforts (évoqués en début de section) visant à concevoir des ontologies dans le domaine des programmes.

2 Le cadre ontologique des travaux

Comme présenté en figure 1, l'ontologie COPS est intégrée dans une ontologie plus large, composée de sous-ontologies situées à différents niveaux d'abstraction : un lien descendant reliant deux sous-ontologies O1 et O2 signifie que les entités conceptuelles (concepts et relations) de O2 sont définies par spécialisation des entités

[2] Dans le cadre du projet NeuroLOG, cette dernière manifestation est exploitée par un logiciel, en cours de développement, qui intègre le moteur de recherche sémantique CORESE conçu dans le cadre du projet ACACIA de l'INRIA (*http://www.inria.fr/acacia/corese*).

conceptuelles de O1. L'ontologie fondationnelle DOLCE et les différentes ontologies noyaux constituent la ressource utilisée par la méthode OntoSpec[3] pour aider à structurer des ontologies d'application[4], comme l'ontologie que nous développons dans le cadre du projet NeuroLOG.

En conséquence de cette structuration globale, la conceptualisation de COPS dépend de choix de modélisation effectués en amont, c'est-à-dire dans des composants situés à un niveau d'abstraction supérieur. Dans cette section, nous présentons ces principaux choix de modélisation en introduisant successivement l'ontologie DOLCE (2.1), la modélisation des rôles (de participation) et des artefacts (2.2) et l'ontologie I&DA (2.3).

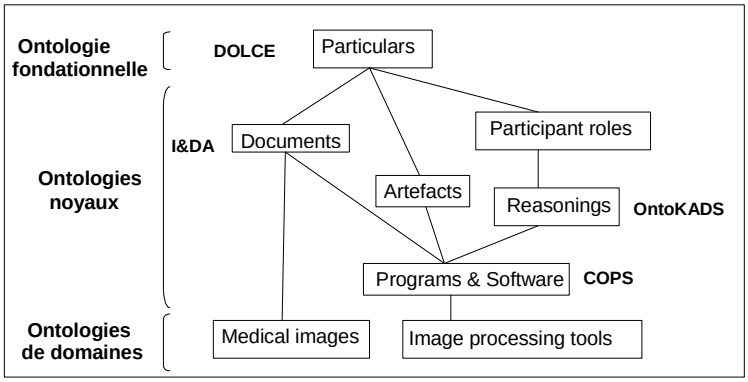

Fig. 1 – Structure de l'ontologie d'application du projet NeuroLOG

2.1 DOLCE

DOLCE est une ontologie « fondationnelle », en ce sens qu'elle comporte des concepts abstraits ayant vocation à généraliser l'ensemble des concepts que l'on peut rencontrer dans les différents domaines de connaissances. Suivant des principes philosophiquement fondés, le domaine de DOLCE – les *Particulars*[5] – est partitionné en quatre sous-domaines (cf. Fig. 2).

– Les *Endurants* sont des entités « endurant dans le temps » (ex. : le présent article). Parmi les *Endurants* sont distingués les *Physical Objects* et les *Non-Physical Objects*, les premiers étant les seuls à posséder des qualités (*Quality*) spatiales directes. Le domaine des *Non-Physical Objects* recouvre le domaine des entités sociales (ex. : la communauté française des chercheurs en IC) et des entités cognitives (ex. : votre notion de l'IC). Pour rendre compte d'entités plurielles comme une communauté ou les actes d'une conférence, une notion de *Collection* a récemment été introduite sous *Non-Physical Objects* (Bottazzi *et al.*, 2006).

[3] Cette ressource est disponible à l'adresse *http://www.laria.u-picardie.fr/IC/site/*

[4] Une « ontologie d'application » contient tous les concepts nécessaires à une application particulière.

[5] Les ontologies présentées dans l'article n'existant qu'en langue anglaise, nous conservons dans la suite pour étiqueter les concepts les termes anglais, qui apparaîtront en *italique* et dans une *notationALaJAVA*.

- Les *Perdurants* sont des entités se « déroulant dans le temps » (ex. : votre lecture de l'article), auxquelles participent des *Endurants*.
- *Endurants* et *Perdurants* ont des propriétés (*Qualities*) inhérentes, que nous percevons et/ou mesurons (ex. : le poids de la copie papier de l'article entre vos mains, la durée de votre lecture de l'article). Parmi les *Perdurants* sont définies les *Actions* qui sont accomplies (*Accomplishment*) intentionnellement, c'est-à-dire contrôlées, par un *Agent* (ce concept est défini en §2.2).
- Ces *Qualities* prennent une valeur (*Quale*) dans des régions de valeurs qui sont des *Abstracts* (ex. : 20 grammes, 15 minutes).

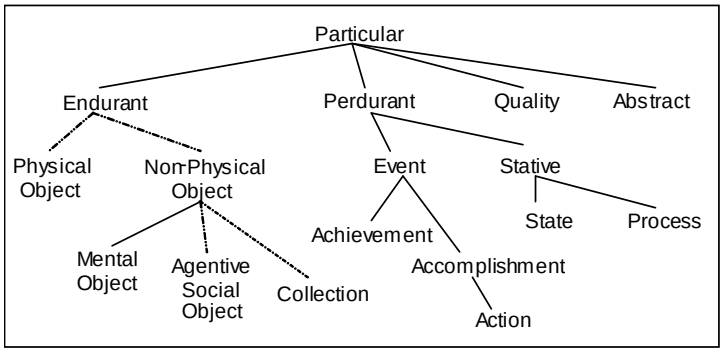

Fig. 2 – Extrait de la hiérarchie de concepts de DOLCE[6].

2.2 Rôles et fonctions

Dans cette section, nous introduisons deux notions importantes liant les *Endurants* aux *Perdurants* – les *rôles* et les *fonctions* – et rappelons les choix de modélisation les concernant (Bruaux *et al.*, 2006).

Un *rôle*, plus exactement un « rôle de participation » ou « rôle thématique », rend compte d'une manière pour un *Endurant* de participer à un *Perdurant*. Par exemple, lors de l'écriture manuscrite d'un article, plusieurs entités participent à cette *Action* : une personne en tant qu'*Agent*, un crayon ou stylo en tant qu'*Instrument* et l'article lui-même en tant que *Résultat*.

Une *fonction* est la capacité à faciliter la réalisation d'une *Action*, et cette notion permet à son tour de définir le concept d'artefact : un *Artefact* est un *Endurant* auquel est attribuée une fonction. Selon le type d'*Action*, différents types d'*Artefacts* peuvent être distingués (cf. fig. 3b). Ainsi, dans OntoKADS, les outils (*Tools*) sont distingués des *Cognitive Artefacts* selon que l'*Action* qu'ils permettent de réaliser correspond à une modification du monde physique ou non physique. Parmi ces derniers, les *Artefacts of Communication* permettent de transmettre des informations à des agents tandis que les *Artefacts of Computation* permettent à des ordinateurs de réaliser des *Actions* en tant qu'*Agents*.

[6] Ces concepts sont définis dans DOLCE au moyen d'une axiomatisation riche, qu'il est impossible de présenter dans l'article, faute de place. Notamment, *Endurants* et *Perdurants* se distinguent par le comportement temporel différent de leurs parties. Le lecteur pourra se référer à (Masolo *et al.*, 2003).

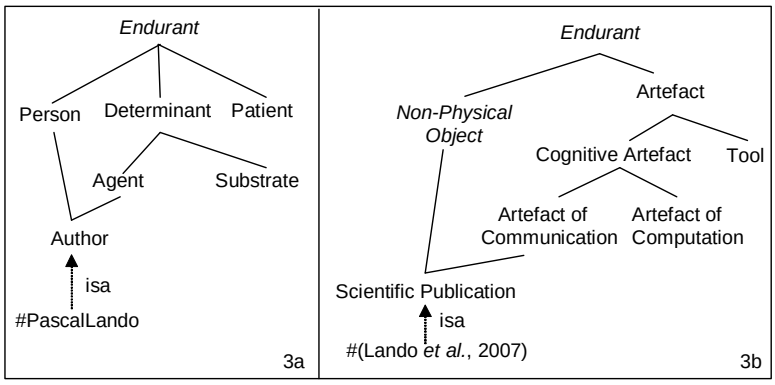

Fig. 3 – Modélisation des rôles (a) et des fonctions-artefacts (b).

On peut noter en figure 3 que les concepts *Author* et *Scientific Publication* encapsulent le type d'une entité et, respectivement, le rôle et la fonction attribuée à l'entité. Ce choix de modélisation, conforme au paradigme le plus courant de modélisation des rôles (Steimann, 2000), conduit à considérer une taxinomie ayant une structure de treillis.

2.3 I&DA

I&DA est une ontologie noyau du domaine de la sémiotique, initialement développée pour classer les documents par leur contenu. Comme le montre la figure 4, I&DA étend DOLCE en introduisant trois concepts principaux.

– Les *Inscriptions* sont des formes de connaissances matérialisées par une substance et inscrites sur un support physique (ex. : un texte imprimé matérialisé par de l'encre sur du papier). En outre, ce sont des objets intentionnels, tenant pour d'autres entités : les *Inscriptions* réalisent (*realize*) des *Expressions*.
– Les *Expressions* sont des formes non-physiques de connaissances organisées par (*isOrderedBy*) un *Language*. À leur tour, les *Expressions* tiennent pour d'autres entités, à savoir les contenus que leur assignent des agents : les *Expressions* expriment (*express*) des *Conceptualizations*.
– Les *Conceptualizations* sont finalement les moyens utilisés par des agents pour raisonner sur le monde. Parmi les *Conceptualizations*, une distinction fonctionnelle est faite entre les *Propositions*, qui sont des descriptions de situations, et les *Concepts*, qui servent à classer des entités.

On notera que, pour rendre compte des documents, I&DA choisit de considérer trois entités distinctes plutôt que trois points de vue différents sur une seule entité. Nous allons voir dans la section suivante que ce choix de modélisation a des répercussions importantes sur la structuration de COPS.

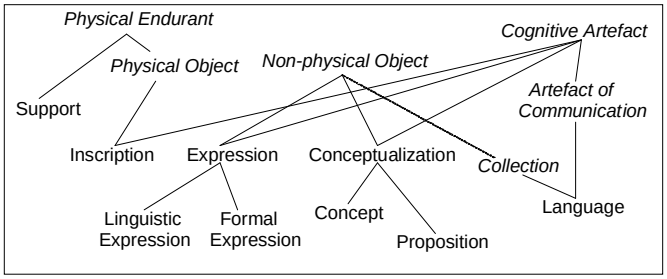

Fig. 4 – Haut niveau de la hiérarchie des concepts de I&DA.

3 COPS : une ontologie noyau des programmes et des logiciels

L'ontologie COPS assimile en effet un programme à un document dont la particularité principale est de permettre à des ordinateurs de réaliser des traitements de l'information. En section 3.1, nous montrons tout d'abord l'apport des ontologies DOLCE et I&DA à la définition de ces documents particuliers. Nous présentons ensuite, en sections 3.2 à 3.4, différentes sous-ontologies permettant de rendre compte de la diversité des programmes.

3.1 Nature duale des programmes

En premier lieu, la distinction introduite dans DOLCE entre *Endurants* et *Perdurants* permet de distinguer le programme-*Endurant* de ses exécutions, qui sont autant de *Perdurants*. En second lieu, en se focalisant sur le programme-*Endurant*, les distinctions établies dans I&DA entre *Inscriptions*, *Expressions* et *Conceptualizations* conduisent à distinguer trois catégories d'entités, communément désignées sous le terme « programme » (cf. Fig. 5) :

– Des fichiers (*Files*), qui sont des *Inscriptions* inscrites sur un support informatique (ex. : disque optique, mémoire centrale, bande magnétique). Ces fichiers ne sont du reste qu'un type d'*Inscription* de programme : une impression sur un listing ou une visualisation sur écran sont également des *Inscriptions* de programmes.
– Des expressions spécifiées dans un langage informatique (*Computer Language Expression*), qui sont des formules bien formées (*isAWellFormedFormulaOf* est une sous-relation de *isOrderedBy*) d'un langage informatique (*Computer Language*). Parmi ces expressions figurent les *Programs*.
– Des *DataTypes* et des *Algorithms*, qui sont des *Conceptualizations* rendant compte de la sémantique des *Programs*. Les *DataTypes* sont des *Concepts* sur lesquels reposent les langages de programmation (ex. : classe, enregistrement), et dont la diversité accompagne la variété des langages de programmation (Turner & Eden, sous presse). Les *Algorithms* décrivent des étapes de calcul en termes de ces structures de données (ex. : affecter une constante à la valeur d'une variable, puis…).

Cette conceptualisation revient à assimiler les programmes à des expressions, ce qui correspond à un point de vue largement consensuel, aussi bien en informatique qu'en philosophie de l'informatique. Pour autant, nous considérons que cette seule dimension syntaxique est insuffisante pour rendre compte complètement de la nature des programmes.

En effet, les programmes possèdent également une dimension fonctionnelle, dans le sens où ils permettent à des ordinateurs de réaliser des *Actions* (*Computations*). Cette dimension fonctionnelle se retrouve dans des expressions comme : « programme de calcul de PGCD » ou « programme de traitement d'images ». Nous aboutissons ainsi à une caractérisation duale des programmes, considérés comme étant à la fois des *Computer Language Expressions* et des *Artefacts of Computation* (cf. Fig. 5). Nous verrons en section 3.3 que le concept *Program* de COPS intègre des contraintes complémentaires par rapport à cette première caractérisation.

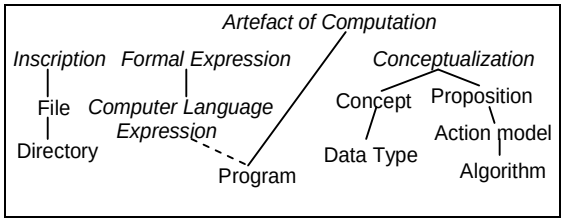

Fig. 5 – Structuration générale de COPS.

3.2 Sous-ontologie des langages informatiques

Pour affiner la dimension syntaxique des programmes, COPS comporte une ontologie des langages. Classiquement, parmi les langages, on distingue les langues naturelles des langages formels. Parmi ces derniers, ceux qui nous intéressent ici sont les langages informatiques (*Computer Language*), c'est-à-dire les langages destinés à une interprétation par un ordinateur (physique) ou un programme. Notre conceptualisation des langages informatiques (cf. Fig. 6) repose sur les fonctions (dimension artefactuelle) des expressions qu'ils permettent d'ordonner. La première catégorie de langages informatiques est celle des langages informatiques généralistes (*General purpose computer language*), c'est-à-dire les langages Turing-complets permettant l'écriture de tous types de programmes. La deuxième catégorie est celle des langages dédiés (*Domain specific computer language*), non Turing-complets, restreints à l'expression de certains types d'instructions (requêtes sur une base de données, commandes adressées à un système d'exploitation, etc).

Les langages de programmation (*Programming Languages*), ou langages de haut niveau, sont tous les langages généralistes compréhensibles par un être humain. Les langages généralistes qui ne le sont pas sont les langages de bas-niveau (*Low-level computer language*) : langages machines (interprétables par un microprocesseur) et byte-codes (interprétables par une machine virtuelle).

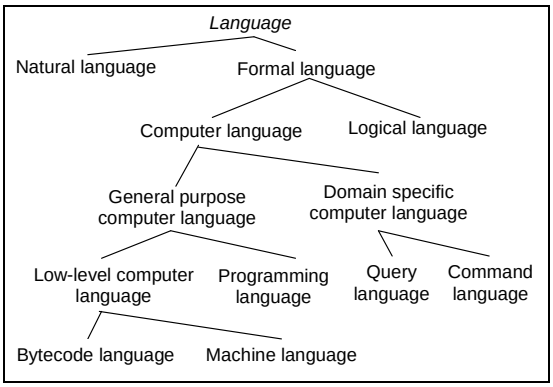

Fig. 6 – Sous-ontologie des langages informatiques dans COPS.

3.3 Sous-ontologie des programmes

La sous-ontologie des langages induit une sous-ontologie des *Expressions* ordonnées par ces langages. On retrouve ainsi, dans la taxonomie des *Expressions*, la structure de l'ontologie des langages (cf. Fig. 7), caractérisant la dimension fonctionnelle des *Expressions*. Ainsi, une *Expression* dans un langage généraliste (*General Purpose Computer Language Expression*) permet de réaliser n'importe quelle *Computation* (ex : déclarer une variable, calculer le PGCD de deux nombres). En revanche, une *Expression* dans un langage de requêtes (*Query Expression*) ou une expression dans un langage de commandes (*Command Expression*) permettent de réaliser des *Computations* particulières, requêtes (interrogation ou modification de données) ou commandes (ordres de réaliser d'autres *Computations*). La notion de Program joue cependant un rôle particulier dans cette taxinomie.

En premier lieu, nous considérons qu'un *Program* ne correspond syntaxiquement qu'à un type particulier d'*Expressions* définies par un *Langage de programmation*. En effet, une *Expression* qualifiée de *Program* a ceci de particulier qu'elle peut, soit être directement exécutée par un ordinateur (éventuellement après une compilation), soit être prise en charge par un interpréteur[7]. Ce n'est pas le cas d'une fonction ou d'une déclaration de variable, par exemple, qui sont pourtant des expressions en *Langage de Programmation*. En second lieu, nous considérons qu'il existe des *Expressions* exécutables ou interprétables qui ne sont pas des *Programs*. En effet, à l'instar d'Eden et Turner (Eden & Turner, 2006), nous ne parlons de *Programs* que dans le cas d'*Expressions* en langages Turing-complets (ou *General purpose programming language)*. Ainsi, une requête SQL ou une commande shell, qui sont pourtant interprétables ou exécutables, ne sont pas considérées dans COPS comme des *Programs*.

En résumé, nous définissons un *Program* comme une *Expression* dans un langage Turing-Complet susceptible d'être exécutable par une machine physique (programme

[7] Par exemple, un programme en langage C est composé d'une ou plusieurs fonctions, dont une en particulier est impérativement baptisée « main ». Par contre, des *Expressions* comme une fonction ou une instruction ne possèdent pas ce point d'entrée rendant l'*Expression* exécutable ou interprétable.

exécutable) ou virtuelle (programme interprétable), éventuellement après traduction (compilation).

En complément, des relations « transversales » relient certaines catégories de *Programs* : ainsi, un *Source code* peut avoir des versions exécutables (relation *hasForExecutable*) et un exécutable ou un byte-code a un *Source Code* (relation *hasForSource-code*). Des relations lient les programmes et des catégories particulières de programmes : un source-code est compilable (*isCompilableBy*) par certains compilateurs et/ou interpretable (*isInterpretableBy*) par certains interpréteurs ; un exécutable tourne sur (*runsOn*) un certain système d'exploitation ; un byte-code tourne sur (*runsOn*) une certaine machine virtuelle. Les systèmes d'exploitation ou les machines virtuelles n'apparaissent en fait pas dans cette partie de COPS car, constitués d'un ensemble de programmes, ils s'apparentent à des logiciels.

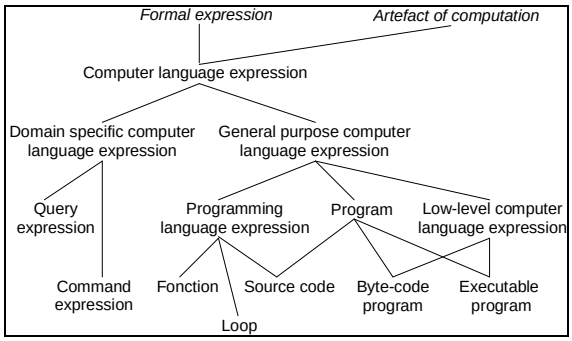

Fig. 7 – Sous-ontologie des programmes de COPS.

3.4 Sous-ontologie des logiciels et des plateformes

Cette sous-ontologie de COPS rend compte du fait que bon nombre de traitements sont réalisés, non pas par un programme isolé, mais par une collection de programmes. Le concept *Library of Programs* (cf. fig. 8) désigne une collection de programmes et de documents qui ne sont pas des programmes, comme de la documentation.

Un logiciel (*Software*) se définit alors (de façon analogue à un programme) comme étant à la fois une *Library of Programs* et un *Artefact of computation*. Un *Software* comporte cependant au moins un programme exécutable ou interprétable. Parmi les logiciels figurent les compilateurs (*Compiler*) dont la fonction est de permettre de traduire des code-sources en programmes exécutables, les interpréteurs (*Interpreter*) dont la fonction est de permettre d'exécuter des code-sources et les systèmes d'exploitation (*Operating Systems*) dont la fonction est de permettre d'exécuter des programmes exécutables. Cette fonction définit une classe d'artefacts, les *Platforms*. Cette fonction est habituellement attribuée, soit à des entités exclusivement matérielles (*Hardware Pltaforms*), soit à des entités logicielles, ou comportant du logiciel, (*Software Platforms*). Parmi ces dernières, on peut distinguer les systèmes d'exploitation mais aussi des architectures matérielles sur lesquelles tourne un système d'exploitation.

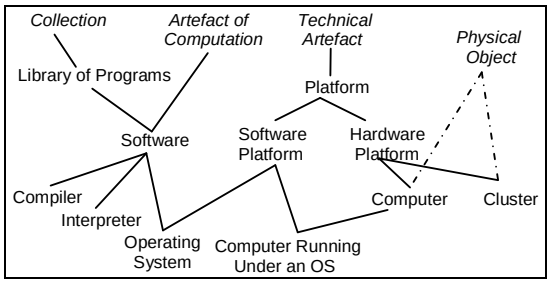

Fig. 8 – Sous-ontologie des librairies de programmes et des plateformes de COPS.

4 Discussion

Dans cette section, nous positionnons COPS vis-à-vis d'autres travaux visant à élaborer des ontologies de programmes. En philosophie de l'informatique, (Eden & Turner, 2006) ont récemment entrepris d'élaborer une ontologie des programmes pour tâcher de répondre à des questions comme : peut-on exhiber des différences entre entités matérielles et entités logicielles, ou comment distinguer un programme d'une spécification de programme ? Bien que les outils ontologiques utilisés dans ce travail soient différents, il est intéressant de comparer les conceptualisations proposées. Ainsi, par exemple, ces auteurs définissent un programme comme une « expression bien formée d'un langage de programmation Turing-complet », mettant en avant des propriétés des langages de programmation mais occultant la dimension fonctionnelle présente dans le concept *Program* de COPS. Ces deux notions étant distinctes, ce constat suggère qu'il soit utile de compléter l'ontologie « noyau » COPS pour prendre en compte d'autres concepts.

La communauté des Services Web est à l'origine de différentes initiatives – METEOR-S, OWL-S, WSMO (Roman *et al.*, 2005) – visant à décrire formellement des services, l'objectif de ces descriptions étant de permettre, de manière plus ou moins automatique, la découverte, l'évocation et l'orchestration de services. Ces efforts sont actuellement éloignés de COPS car, d'une part, l'accent est mis sur le caractère opérationnel des descriptions et, d'autre part, ces descriptions, lorsqu'elles concernent la fonction (l'action en COPS) réalisée par le service (ex. : réserver un billet de train), se situent à un niveau méta pour définir les pré-conditions permettant au service de s'exécuter (ex. : des informations sur un trajet doivent être données) et les effets qui résultent de son exécution (ex. : le prix du billet est débité d'un compte en banque). Dans le cadre du projet NeuroLOG, les fonctionnalités visées en termes d'évocation et d'orchestration d'outils logiciels étant proches, nous nous apprêtons à étendre COPS pour prendre en compte ce niveau de description.

Dans le domaine du génie logiciel, (Welty, 2005) a proposé de développer des systèmes d'aide à la maintenance de logiciels (Comprehensive Software Information Systems) exploitant une ontologie permettant de décrire conceptuellement, et dans les moindres détails, les logiciels. Cette ontologie peut être considérée comme une extension de la sous-ontologie des expressions de COPS, car elle permet de descendre au

niveau des lignes de code et de prendre en compte les constructions proposées par les langages de programmation. Par contre, elle suppose (étrangement) que les entités jouant le rôle de donnée et de résultat soient des entités du monde réel (ex. : des personnes) et non des conceptualisations modélisant le monde réel. Dans COPS, au contraire, nous suivons la thèse défendue par (Turner & Eden, sous presse) selon laquelle la sémantique des programmes repose sur des données et des types de données modélisant les entités du monde réel, par exemple, dans le paradigme objet, une *instance* modélisant une personne individuelle ou une *classe* modélisant un ensemble de personnes.

Toujours dans le domaine du génie logiciel, (Oberle *et al.*, 2005) proposent l'ontologie CSO (Core Software Ontology) destinée à mieux développer, administrer et maintenir les systèmes informatiques complexes. La démarche de construction de l'ontologie est proche de la notre : réutilisation de l'ontologie de haut niveau DOLCE et d'ontologies noyaux comme DnS (Descriptions & Situations). COPS et CSO partagent des choix de modélisation, comme le fait de distinguer trois entités équivalentes aux *Inscriptions*, *Expressions* et *Conceptualizations* de COPS. Par contre, on peut noter des choix de modélisation différents. Par exemple, dans CSO, prenant prétexte du fait que tout programme peut être la donnée d'un autre programme, le concept *Data* subsume le concept *Software*. En outre, pour ce qui concerne les programmes considérés en tant que des expressions, CSO est moins détaillée que COPS : les notions de programme et de logiciel ne sont pas distinguées et la dimension fonctionnelle des programmes est absente.

Ces comparaisons montrent qu'à ce jour, des propositions d'ontologies noyaux existent dans le domaine des programmes informatiques, mais que les efforts restent non coordonnés et que les ontologies proposées présentent des différences importantes, tant dans leur portée que dans leur structuration.

5 Conclusion

Nous avons présenté dans cet article les bases d'une ontologie « noyau » des programmes et logiciels (COPS), élaborée par spécialisation de l'ontologie fondationnelle DOLCE, et amenée à son tour à structurer des domaines plus spécifiques de programmes, notamment celui des outils de traitements d'images.

La conceptualisation actuelle de COPS révèle un domaine peuplé d'entités de natures très diverses. On y trouve en effet des entités temporelles (des exécutions de programmes), des entités physiques (des inscriptions de programmes), des entités plurielles (des collections de programmes), des entités fonctionnelles (des plate-formes d'exécution de programmes) et, enfin, des entités comportant une double nature, syntaxique et fonctionnelle : les programmes eux-mêmes. Le retour d'expérience de construction de COPS nous conforte dans l'idée que la réutilisation de ressources ontologiques, apportant des choix de modélisation à différents niveaux d'abstraction, s'avère indispensable pour maîtriser la complexité de tels domaines.

Dans sa version actuelle, COPS ne couvre qu'une partie de son domaine. Des travaux sont en cours pour étendre l'ontologie dans plusieurs directions. Un premier

objectif est de compléter la sémantique des programmes : les liens avec les traitements réalisés (la fonction) ne rendent compte que du **quoi**, il manque donc le **comment**, ce qui nécessite de prendre en compte les algorithmes et types de données ; or ceux-ci ont seulement été positionnés (cf. fig. 5) sans être étudiés précisément. Un second objectif est d'élargir COPS aux spécifications de programmes : nous comptons pour cela réutiliser la notion de « modèle de résolution de problèmes » proposée dans l'ontologie OntoKADS (Bruaux *et al.*, 2006) pour l'étendre à la classe plus générale des modèles d'actions réalisées par des ordinateurs au moyen de programmes.

Références

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

BRUAUX S., KASSEL G. & MOREL G. (2006). OntoKADS : une ontologie générale de la résolution de problèmes. In *Actes des 17èmes journées francophones d'Ingénierie des Connaissances : IC'2006*, Nantes. http://www.sdc2006.org/cdrom/

EDEN A. H. & TURNER R. (2006). Problems in the Ontology of Computer Programs. *Technical Report CSM-461, ISSN 1744-8050, Department of Computer Science, University of Essex.* *www.eden-study.org/articles/2006/problems-ontology_programs_csm461.pdf*

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

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

MASOLO C., BORGO S., GANGEMI A., GUARINO N., OLTRAMARI A. & SCHNEIDER L. (2003). The WonderWeb Library of Foundational Ontologies and the DOLCE ontology. *WonderWeb Deliverable D18, final report* (vr. 1.0, 31-12-2003).

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

OBERLE D., LAMPARTER S., GRIMM S., VRANDECIC D., STAAB S. & GANGEMI A. (2006). Towards Ontologies for Formalizing Modularization and Communication in Large Software Systems. In *Applied Ontology*, 1(2), p. 163-202.

ROMAN D., KELLER U., LAUSEN H., DE BRUIJN J., LARA R., STOLLBERG M., POLLERES A., FEIER C., BUSSLER C. & FENSEL D. (2005). Web Service Modeling Ontology. In *Applied Ontology* 1, p. 77-106.

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

TEMAL L., LANDO P., GIBAUD B., DOJAT M., KASSEL G. & LAPUJADE A. (2006). OntoNeuroBase: a multi-layered application ontology in neuroimaging. In *Proceedings of the 2nd Workshop: Formal Ontologies Meet Industry: FOMI 2006*, Trento (Italy).

TURNER R. & EDEN A.H. (sous presse). Towards a Programming Language Ontology. In G. Dodig-Crnkovic and S. Stuart (eds.), *Computing, Philosophy, and Cognitive Science*, Cambridge, UK: Cambridge Scholars Press.

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