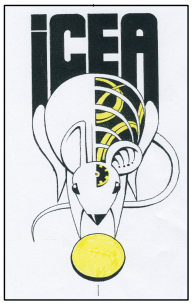


Integrating Cognition+Emotion+Autonomy

European Integrated Project IST-027819

6th Framework Programme - Cognitive Systems

Title

Ontology-based Software Engineering

Engineering Support for Autonomous Systems

Author

Julita Bermejo Alonso

Reference ASLab-ICEA-R-2006-016

Release 0.1 Draft

Date 2006-11-15

URL http://www.aslab.org/documents/ASLab-ICEA-2006-016.pdf

Clearance Consortium

Partners

University of Skövde, SE
Université Pierre et Marie Curie, FR
Centre National de la Recherche Scientifique, IT
Consiglio Nazionale delle Ricerche, FR
University of Sheffield, UK
University of the West of England, UK
BAE Systems, UK
Cyberbotics Ltd., CH
Hungarian Academy of Sciences, HU
Universidad Politécnica de Madrid, ES

Ontology-based Software Engineering

ASLab-ICEA-R-2006-016 v 0.1 Draft of 2006-11-15

Abstract

Keywords

Acknowledgements

We acknowledge the support of the European Commission Cognitive Systems Unit through grant IST-027819 ICEA.

Revisions

Release	Date	Content	Author
0.0	November 15, 2006	Initial release.	J.Bermejo

Contents

1 Introduction **8**

2 Software Engineering Definitions **9**

3 Roles of Ontologies for Software Development **11**

4 Categorization of Ontologies for Software Engineering **12**

5 Benefits on the Use of Ontologies **15**

6 Issues on Ontology–based Software Process **16**

7 Current Research on Ontologies for Software Engineering **18**

7.1 Ontologies for the Software Engineering Domain: Ontologies of Domain 18

7.2 Ontologies for the Software Engineering Process: Ontologies as Software Artifacts 31

8 Conclusions **35**

List of Figures

4.1 Categorization of usage of ontologies in Software Engineering 13

7.1 The SWEBOK ontology project phases 19
7.2 The SWEBOK ontology mapping 19
7.3 Application Configuration 20
7.4 Semantic Network of Modeling 21
7.5 Verification of Knowledge Based Systems by deploying Core Ontologies 23
7.6 Structure of the software maintenance projects ontology 24
7.7 Part of the Software Quality Ontology 24
7.8 Software Measurement Ontology 25
7.9 Ontology for Software Metrics and Indicators 26
7.10 Part of the Software Process Ontology 27
7.11 Software Process Metal–Level Model 28
7.12 Software Process Base Level Model 28
7.13 Architecture of Ontology for Software Development Methodologies and Endeavours 29
7.14 Onto-ActRE Problem Domain Ontology 30
7.15 EOSDE components 31
7.16 MANTIS Components 32
7.17 Ontology Modules for use cases supporting development . . . 34

Chapter 1

Introduction

Software development is more demanding nowadays as software applications have become increasingly more complex. New procedures and techniques are sought to simplify the software engineering processes with the aim of shortening development time and costs by re–using components. The development of software systems is a complex activity which may imply the participation of people and machines (distributed or not). Therefore, different stakeholders, heterogeneity and new software features make software development a heavily knowledge–based process [Force, 2001].

To reduce this complexity, the use of ontologies might prove useful. Ontologies allow for the definition of a common vocabulary and framework among users (either human or machines). Software development has benefited from this conceptual modeling, allowing a common understanding of the concepts involved in the software process. Ontologies also ease the integration issues that usually appear while developing software applications.

Chapter 2

Software Engineering Definitions

Software Engineering or *Software Process* is the set of activities, methods, practices, and transformations carried out to develop and maintain software and associated products, usually executed by a group of people who are organized according to a structure, with the support of technical tools [Acuña et al.,]

Software Process Modeling has been defined as the linguistic, diagrammatic or numerical representation of patterns of activities (processes) [Menzel and Grüninger, 2001]. The outcome of a software process modeling is a *Software Process Model*, also called *Process Model*. The processes are represented as models that represent, in an abstract way and at different levels of detail, the different processes that are involved in a finished, ongoing or proposed software process [Acuña and Ferre, 2001]. A robust foundation for process modeling should characterize both the general process structure described by a model as well as the class of possible instances of that structure.

The software process models could be either descriptive or prescriptive models[Acuña et al. A *Descriptive Model* pays attention to how the software was developed within the organization. A *Prescriptive Model*, on the other hand, focuses on the guidelines required to execute the software process, i.e, how the software must be developed. Within the later, manual and automated models could be considered. Manual prescriptive models are standards and methodologies to guide the software life cycle. Automated prescriptive models are computerized specifications of software process standards, which help agents involved in the software process to interpret the software process models in an automatic way.

Process models are used with several purposes: to improve process understanding and communications, process control, specify process development viewpoints or perspectives [Acuña and Juristo, 2005]. Furthermore, they can be analyzed, validated, simulated and executed. Therefore, process models provide a better way to define the development, maintenance and evolution of software processes, both regarding individual activities or the process as a whole [Acuña and Sánchez-Segura, 2006].

Model–driven software development (MDD) is based on models, modeling and model transformations. Models are used to reason about the problem do-

main and the solution domain. Modeling approaches have evolved from co-existence of model and code to a model–only one, where models are used in software processes as discussion tools. In between, the model–centric approach regards the models as code, if defined with enough detail. Transformation rules and patterns provide the tools to generate code from such models [Brown et al., 2005].

Domain–driven development is the development of software applications within a specific domain or application area [Hruby, 2005]. Also called *Domain engineering*, its goal is to identify, model and implement software models to be used in a particular application domain [Falbo et al., 2002].

Ontology–driven software development or engineering has been defined as an approach that, based on ontologies, takes into account semantic constraints, adapting in a dynamic way to new constraints [Tanasescu, 2005]. It could be considered a particular case of model–driven software, where models are based on ontologies at different levels of abstraction.

Chapter 3

Roles of Ontologies for Software Development

Ontologies, for software design and development, can be used with the following objectives [Ruiz and Hilera, 2006]:

- Specification: ontologies are used to specify either the requirements and components definitions (informal use) or the system´s functionality.
- Confidence: ontologies are used to check the system´s design.
- Reusability: ontologies could be organized in modules to define domains, subdomains and their related tasks, which could be later reused and/or adapted to other problems.
- Search: ontologies are used as information repositories.
- Reliability: ontologies could be used in (semi)–automatic consistency checking.
- Maintenance: ontologies improve documentation use and storage for system's maintenance.
- Knowledge acquisition: ontologies could be used as a guide for the knowledge acquisition process.

Within Software Engineering, two main roles for ontologies have been considered [Hesse, 2005]:

- Ontologies for the Software Engineering Process: the definition, re–use and integration of software components is aided by the use of ontologies as the conceptual basis.
- Ontologies for the *Software Engineering Domain*: the use of ontologies to describe the structure and terminology of the software engineering domain itself.

Chapter 4

Categorization of Ontologies for Software Engineering

To clarify the usage of ontologies within the software engineering and software technology, a taxonomy of ontologies is proposed in [Ruiz and Hilera, 2006]. Two different categories are considered and further detailed:

1. *Ontologies of Domain*: to represent (partial) knowledge of a particular domain within Software Engineering and Technology.
    - Software Engineering (SE)
        - Generic
        - Specific: requirements, design, construction, testing, maintenance, configuration management, quality, engineering tools and methods, engineering process, engineering management.
    - Software Technology (ST)
        - Software: programming techniques, programming languages, operating systems.
        - Data: data structure, data storage representations, data encryption, coding and information theory, files.
        - Information Technology and Systems: models and principles, database management, information storage and retrieval, information technology and systems applications, information interfaces and representation.
2. *Ontologies as Software Artifacts*: ontologies are used as some kind of artifact during a software process.
    - At Development Time
        - For Engineering Processes: when ontologies are used during development and maintenance processes
        - For Other Processes: when ontologies are used during complementary customer–supplier, support, management and organization processes.

- At Run Time
  - As Architectural Artifacts: when ontologies are part of the system software architecture.
  - As (Information) Resources: when ontologies are used as resource at run time by the system.

The categories in this taxonomy could be mapped to the two main roles defined beforehand (Sec. 3) :

Roles	*TaxonomyCategories*
Ontologies for Software Engineering Domain	Ontologies of Domain
Ontologies for Software Engineering Process	Ontologies as Software Artifacts

Table 4.1: Mapping of Roles and Taxonomy Categories

An additional categorization of ontologies for software engineering is provided in [Happel and Seedorf, 2006]. Two dimensions for comparison are considered: the role of ontologies (at run time vs. at development time) and the kind of knowledge (domain vs. infrastructure). Combining this two dimensions, four different areas are defined:

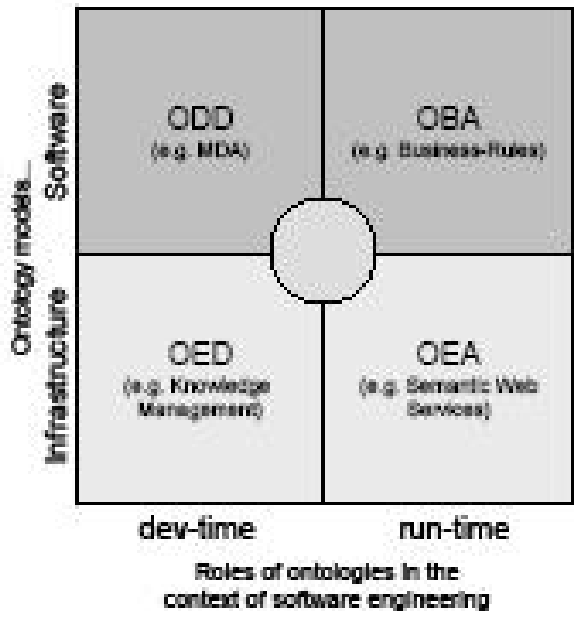


Figure 4.1: Categorization of usage of ontologies in Software Engineering

- Ontology–driven development (ODD): ontologies are used at development time to describe the domain.

- Ontology–enabled development (OED): ontologies are used at development time to support developers with their activities.
- Ontology–based architectures (OBA): ontologies are used at run time as part of the system architecture.
- Ontology–enabled architectures (OEA): ontologies are used at run time to provide support to the users.

This categorization provides a similar viewpoint about the use of ontologies corresponding to the category "Ontologies as software artifacts" previously described. Both consider the same temporal dimension (at run time v.s at development time), as described in [Guarino, 1998]. Paying attention to the definitions (not the names) used in the categorization, the other dimension about kind of knowledge (infrastructure vs. software) is underlying in the subcategories considered by the taxonomy

Chapter 5

Benefits on the Use of Ontologies

Ontologies provide benefits regarding the process of software development, which could be summarized as follows , [Falbo et al., 1998], [Falbo et al., 2002], [Liao et al., 2005], [Ruy et al., 2004], [Abran et al., 2006], [Ruiz and Hilera, 2006]:

- Ontologies provide a *representation vocabulary* specialized for the software process, eliminating conceptual and terminological mismatches.
- The use of ontologies and alignment techniques allow *to solve compatibility problems* without having to change existing models.
- Ontologies might help *to develop benchmarks* of software process by collecting data on the Internet and the use of the Semantic Web.
- Ontologies allow both *to transfer knowledge* and *to simplify the development cycle* from project to project.
- Ontologies promote common understanding among software developers, as well as being used as domain models.
- Ontologies allow for an *easier knowledge acquisition process*, by sharing a same conceptualization for different software applications.
- Ontologies allow *to reduce terminological and conceptual mismatches*, by forcing to share understanding and communications among different users during the ontological analysis.
- Ontologies also provide for *a refined communication* between tools forming part of an environment.
- Ontologies, when as *machine–understandable representations*, help in the development of tools for software engineering activities.

Chapter 6

Issues on Ontology–based Software Process

Although ontologies are considered a useful element within software engineering activities, some issues should still be born in mind when developing ontology–based software development projects [Hesse, 2005]:

- The ontology–based approach is adequate for those software development projects that belong to a set of projects within the same domain.
- The ontology–based approach allow to extend the notion of reusability to the modeling phase, not only the usual implementation one. Therefore, ontologies could be considered reusable model components in the system.
- Model–Driven Developments can benefit from the use of ontologies as model re–use mechanisms.
- The ontology–based approach affects all the software development process phases, from requirement analysis and domain analysis to integration, deployment and use of the developed software.
- The ontology–based approach allow ontologies to be used to facilitate software development in the long term, as well as addressing interoperability and re–use issues.

Furthermore, ontologies should exhibit some specific properties to facilitate their use within the software engineering community [González-Pérez and Henderson-Seller

- Completeness: to assure that all areas of software development are covered. It could be achieved by paying attention to the different activities carried out by software development enterprises.
- Unambiguity: to avoid misinterpretations. Ambiguity could be avoided by using both concise definitions of concepts and semi–formal models.
- Intuitive: to specify concepts familiar to users' domain.

- Genericity: to allow the ontology to be used in different contexts. It could be done by keeping the ontology as small as possible, to achieve maximum expressiveness while being minimal.
- Extendability: to facilitate the addition of new concepts. It could be achieved by providing appropriate mechanisms defining how to extend the ontology.

Chapter 7

Current Research on Ontologies for Software Engineering

This section attempts at providing a literature review of current research on the use of ontologies within software engineering. The examples provided have attempted to cover most of the current research under different topics. The review is as comprehensive as it could be up to date, since the use and development of ontologies for software engineering are ongoing topics under continuous research and development.

To achieve some level of organization within the several extant ontologies, environments and projects, they have been classified according to the mapping of roles and the taxonomy described in Table 4.1.

7.1 Ontologies for the Software Engineering Domain: Ontologies of Domain

Ontologies are used, in general, to represent (partial) knowledge of a particular domain within Software Engineering.

1. Software Engineering

   - *Generic*

     Ontologies based on the SWEBOK guide: The Guide to the Software Engineering Body of Knowledge (SWEBOK) [Society, 2004] seeks to identify, describe and organize a portion of the body of knowledge of the discipline of software engineering that is generally accepted (i.e. knowledge and practices described are applicable to most projects most of the time, with an existing consensus about their value and utility).

     Based on the SWEBOK guide, several attempts to develop a generic ontology for software engineering have taken place. As a first approach, several ontology development methodologies were

analyzed to specify the main activities to develop such ontology [Mendes and Abran, 2005b]. Three phases were finally defined: proto–ontology construction, internal validation cycle and finally, external validation and possible extension Fig. (7.1). The SWEBOK proto–ontology contains over 6,000 concepts and 1,200 facts or instances of concepts [Mendes and Abran, 2005a].

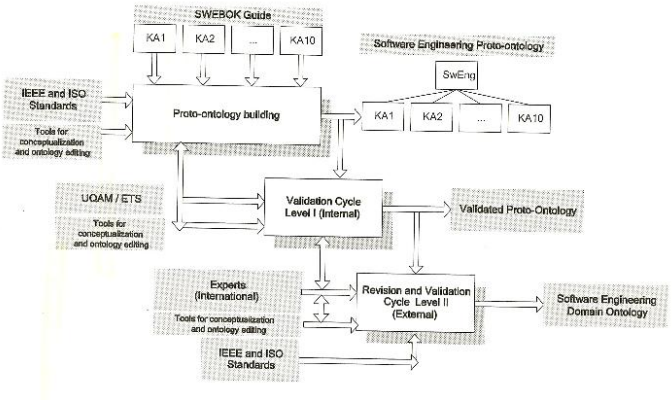


Figure 7.1: The SWEBOK ontology project phases

Another approach for an ontology based on the SWEBOK guide falls within the Onto–SWEBOK project [Sicilia et al., 2005], where essential software engineering concepts are mapped into OpenCyc (the open source version of Cyc (MISSING REF!))definitions. Such concepts encompasses the idea that a software engineering deals with *artifacts* created by *agents* as a result of *activities* guided by *rules*. As a result, some propositions for the general mapping have been proposed Fig. (7.2).

Proposition #1 `SoftwareEngineers` are a class of `oc IntelligentAgents` (excluding collectives). Software engineering activities will require individuals of this class.

Proposition #2 Actual software engineering activities, as enacted in software projects, are a specific class of `PurposefulAction` situated in the context of a project that has as its final outcome the creation or modification of a software program.

Proposition #3 The elements used, created and modified in software engineering activities are specific kinds of `Artifacts`.

Figure 7.2: The SWEBOK ontology mapping

A summary of both approaches could be found in [Abran et al., 2006].

Ontology–Based Domain–Driven Design [Hruby, 2005]: the research focuses on the development of software applications for a specific domain, based on the use of ontologies. A method consisting of

four steps is proposed:

(a) Determination of the domain: the first step is to determine the scope of the application.

(b) Election of an ontology for the domain: this step requires either to choose an existing ontology or to develop one suitable for it. A metamodel can aid to develop a domain ontology.

(c) Consideration of users requirements: this step considers the requirements for functionality that are not part of the ontology, as they are not needed in all applications belonging to the chosen domain.

(d) Construction of the application model: this step implies configuring objects from ontological categories (metaclasses) with aspects (concepts) originated from specific user requirements.

Hruby distinguishes clearly between **domain ontology** (which specifies the structure of **concepts to be applied to all systems** in the domain, with the compromise of minimal possible conceptualization) and **application functionality** (which specifies the functionalities and user requirements which might **differ** from application to application, and cannot be fully considered at the time of ontology creation), The former is addressed by using two dimensions. The first one, Object dimension, implemented as Ontological Categories, whose instantiation are the Application Objects. The second one, Aspect dimension, to reflect behavior by considering Aspect Categories, whose instantiation produce Application Aspects.

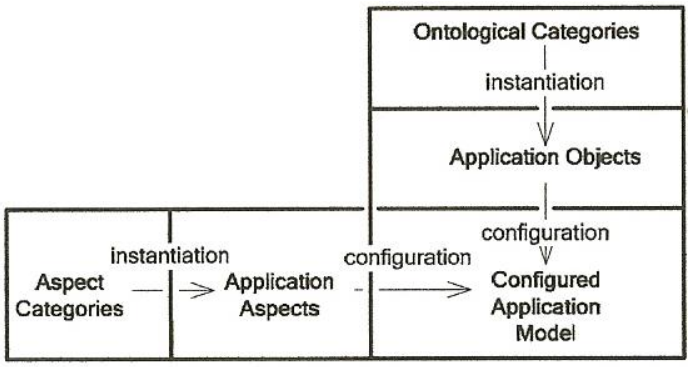


Figure 7.3: Application Configuration

- *Specific*

Software Requirements

Generic Requirement Analysis Method based on Ontologies (GRAMO) [Girardi et al., 2004]: it is an ontology–based technique for the spe-

cification of domain models in the analysis phase of Multi–Agent Domain Engineering . Different modeling tasks are considered:

- *Goal modeling*: both the general goal of the system as well as specific goals are considered. Additionally, the responsibilities needed to reach a specific goal are described. The outcome is a goal model as a three level chart (general goal as first level; specific goals as second level; responsibilities as third level).
- *Role modeling*: the responsibilities identified in the former task are assigned to an internal or external role. Later, the activities needed for each responsibility are defined, as well as the inputs, resources and outputs of each activity. The outcome is a three level organizational chart (responsibility in the first level; activity in the second one and, resources in the third level).
- *Variability modeling*: Several rules have been defined to allow the refinement of goals, roles, responsibilities, activities and resources. The outcome is the definition of fixed features (existing in all subsystems of a family of systems in the domain) or variable features (specific characteristics of a particular system in the family).
- *Interaction modeling*: the interaction among internal and external roles are analyzed by considering their activities, inputs, and outputs. The outcomes is an interaction model for each goal, similar to the collaboration diagram in UML.
- *Concept modeling*: the concepts of the domain and their relations are considered as part of the modeling task. The outcome is a concept model as a semantic network (nodes as concepts; links as relationships).

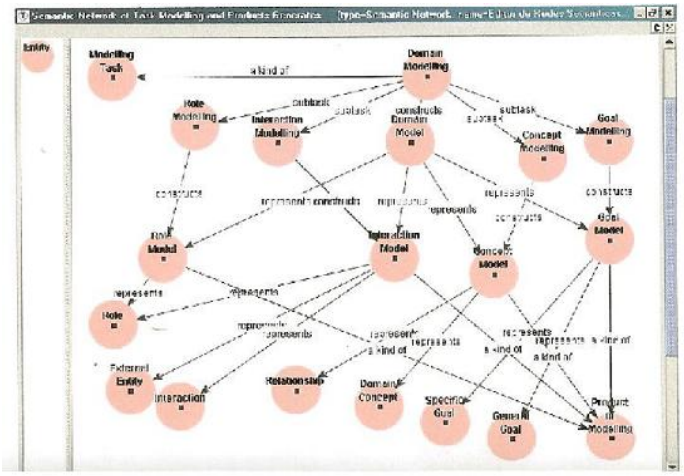


Figure 7.4: Semantic Network of Modeling

Domain Modeling, within this context, refers either to the formulation of a problem (by using the Goal, Role, Variability and Inter-

action modeling tasks) or the representation of a knowledge area (by carrying out the Concept and Variability modeling tasks). The knowledge of the GRAMO technique is represented by an ontology ONTODM [Girardi and Faria, 2003], which is an instantiation of the Domain Model class (and therefore, of the Concept, Goal, Role, Variability, and Interaction Models).

Software Design

Ontologies in Software Design [Kalfoglou, 2000]: Ontologies are used as domain knowledge to check the initial phases of software systems design, with the aim of detecting conceptual errors within the domain knowledge (Fig. (7.5)).

The reliability of software systems, with respect to consistency checking, is tested in different domains such as business process modeling or systems dynamics.

Software Maintenance

Software Maintenance Ontologies: Software maintenance is a knowledge–intensive process, where knowledge of activities, legacy software, system architecture, problem requirements coming from different sources is handled to maintain the software [Anquetil et al., 2006]. Therefore, ontologies provide a solid basis to conceptualize such knowledge. Ontologies for software maintenance have been developed by [Dias et al., 2003] and [Ruiz et al., 2004], where the ontology consists of sub–ontologies Fig. (7.6). Due to some similarities among the two of them, an attempt to merge them is described in [Vizcaíno et al., 2005].
A detailed version of the aforementioned efforts is summarized in a Maintenance Ontology [Anquetil et al., 2006]. The knowledge required to maintain a software system is organized as five different sub–ontologies: system, computer science skills, maintenance process, organizational structure and application domain.

Software Quality

The importance of quality in software is a topic under study within software engineering. However, what it is understood by software quality is not fully defined.
Software Quality Ontology [Falbo et al., 2002]: To tackle this variability, a Software Quality Ontology was developed as part of Ontology–based Domain Engineering (ODE). Different competency questions, regarding quality characteristics, relevance, metrics and paradigms were considered. The concepts and relationships defined as a result are shown in Fig. (7.7).

Likewise, within software quality, several standards and propos-

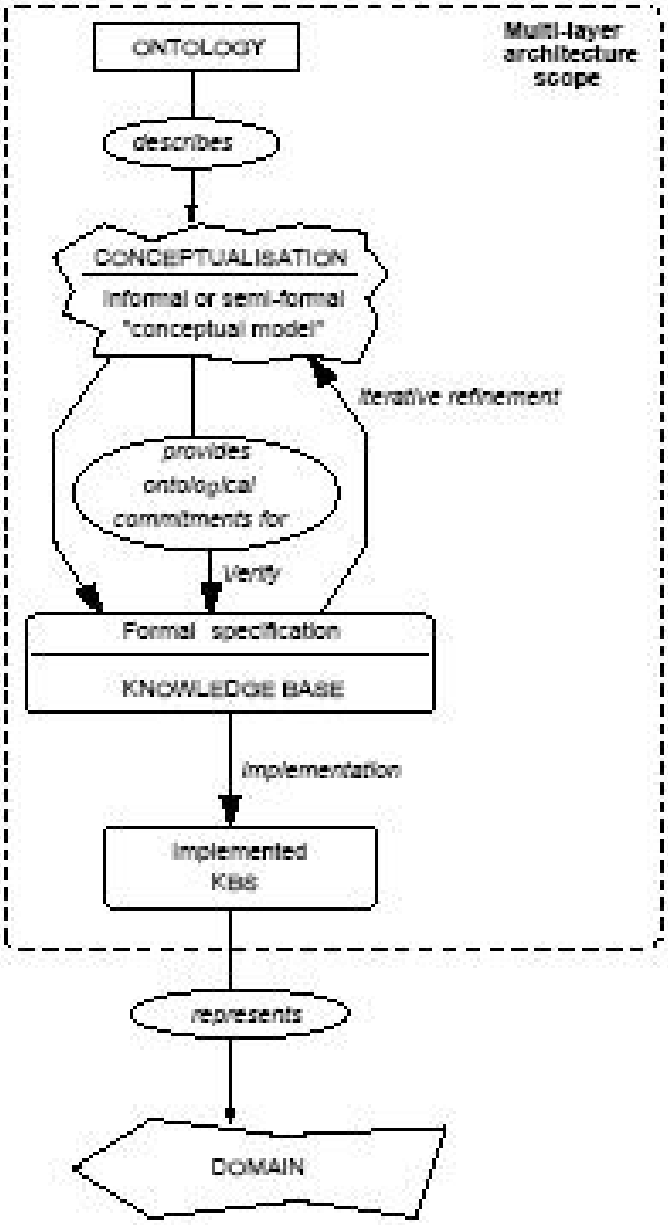


Figure 7.5: Verification of Knowledge Based Systems by deploying Core Ontologies

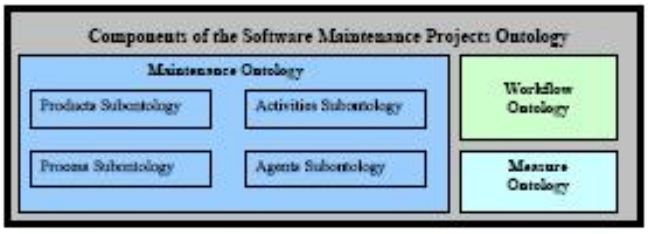


Figure 7.6: Structure of the software maintenance projects ontology

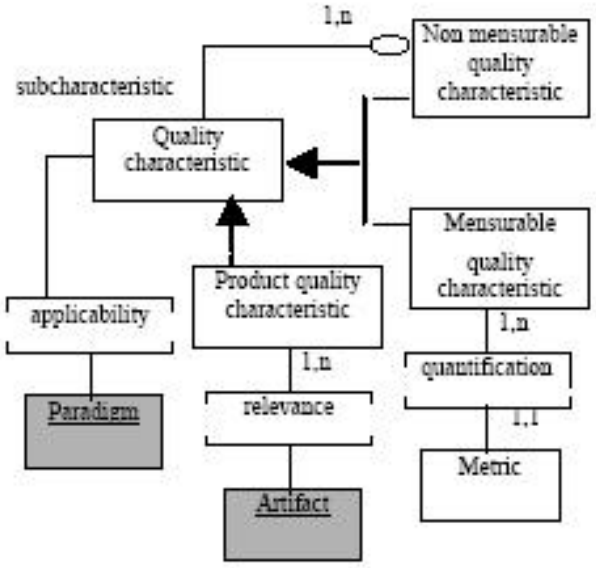


Figure 7.7: Part of the Software Quality Ontology

als exist when it comes to software measurement terminology. In an effort to address this issue, ontologies have been used as means to come up with a common vocabulary and terminology. Two examples of such effort are provided. usually utilized in software engineering.

Software Measurement Ontology (SMO) [Bertoa et al., 2006]: An ontology to provide a common vocabulary for software measurement was developed by using different standards. Concepts were identified, their definitions were provided and the relationships among them were described. The ontology was organized in four sub–ontologies: software measurement characterization and objectives, software measures, measurement approaches and finally, measurement Fig. (7.8) using REFSENO [Tautz and Wangenheim, 1998] as representation mechanism and development guide.

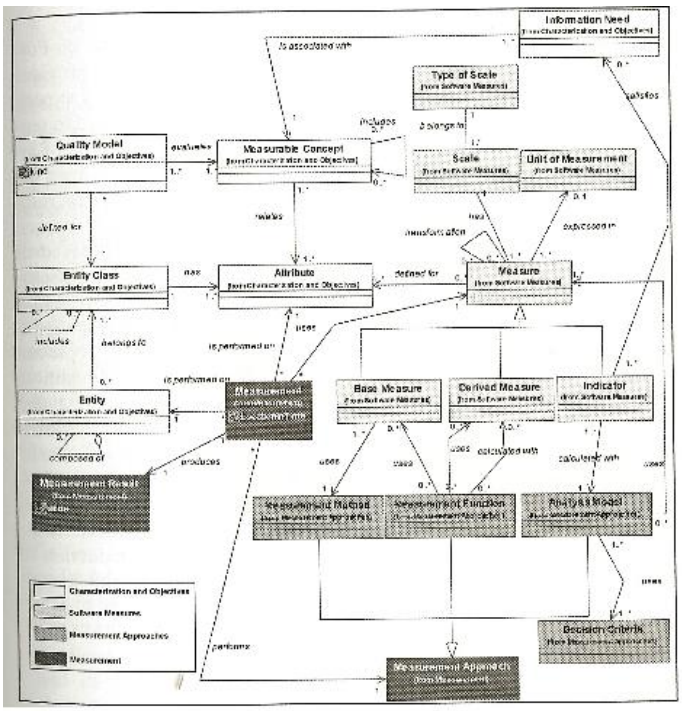


Figure 7.8: Software Measurement Ontology

Ontology for Software Metrics and Indicators [Martín and Olsina, 2004]: An ontology was developed to aid in reaching a consensus on quality, performance and measurement related concepts when it comes to software metrics and indicators, as seen in Fig. (7.9).

Software Engineering Process

Ontology–based Development Environment (ODE) : ODE is an environment to develop software based on ontologies for the software

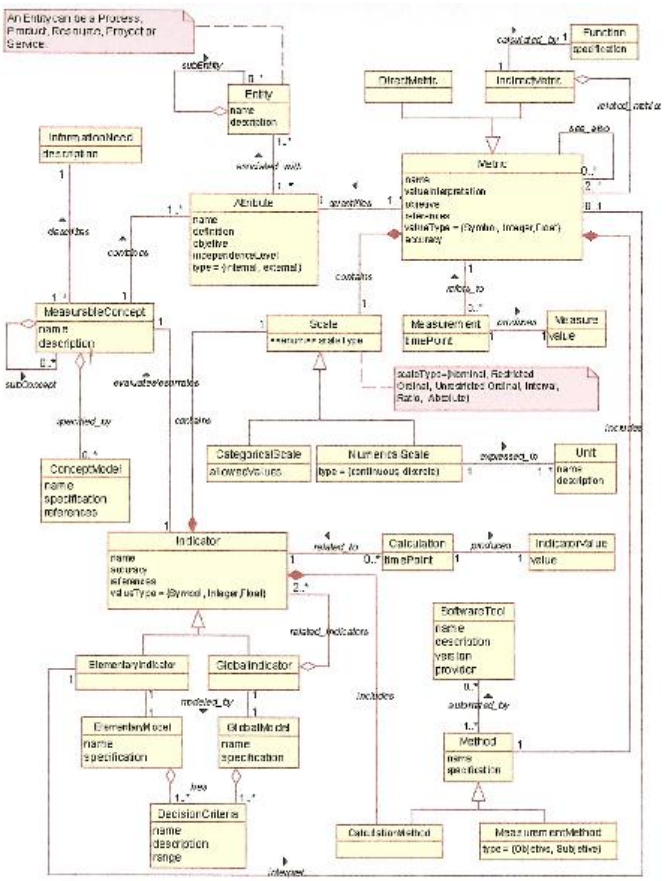


Figure 7.9: Ontology for Software Metrics and Indicators

development domain. The ontological approach to domain engineering considers three main activities: domain analysis in the form of ontology development with a systematic approach [Falbo et al., 1998], infrastructure specification made by mapping the ontology into object models and finally, infrastructure implementation by developing Java components [Falbo et al., 2002].

The ODE's architecture consists of different levels, defined and clarified throughout the research [Falbo et al., 2003], [Falbo et al., 2004b], [Falbo et al., 2005]:

(a) Ontological level: description of the ontologies themselves by using such as concepts, properties, relationships, axioms and competence questions. It corresponds to the ODE's meta–ontology.

(b) Meta level: description of the classes related to the knowledge of the software engineering domain. Such classes are instances of the former level elements.

(c) Base level: definition of the classes that implement ODE's applications. The classes can be derived from the two former levels, but for particular applications, new classes, associations and attributes could be defined as needed.

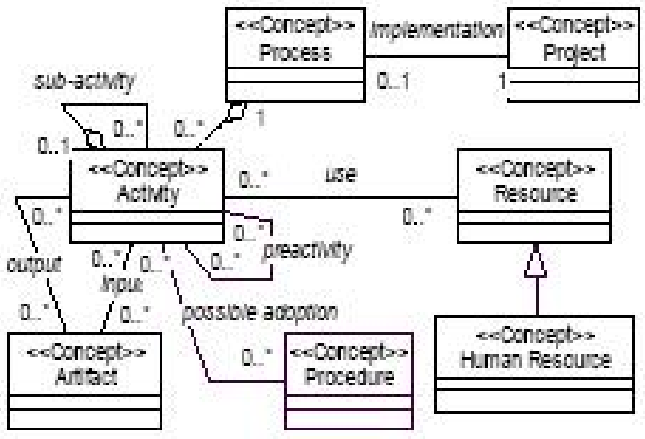


Figure 7.10: Part of the Software Process Ontology

The environment also considers an Organizational Memory to store, handle and update the knowledge acquired, both from formal and informal sources [Falbo et al., 2004a]. ODE has been extended with both an ontology (ODEd) and axioms (AxE) editors [Souza and Falbo, 2005].

Their experiences, when it comes to strong points and weaknesses of ODE have been summarized in [Falbo, 2004]. Additionally, the terms used in ODE have been mapped to well–established software process quality standards, methodologies and models [Falbo and Bertollo, 2

Ontology for Software Development by Method Engineers and Software

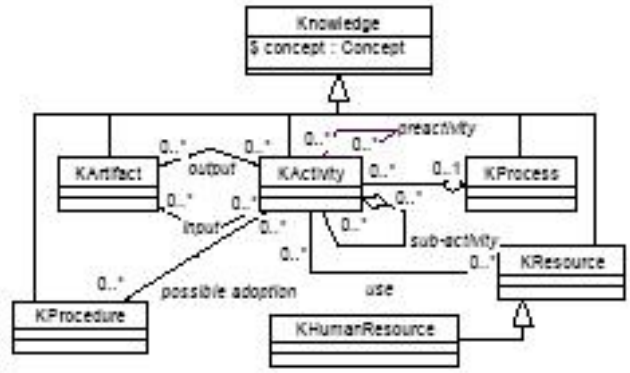


Figure 7.11: Software Process Metal–Level Model

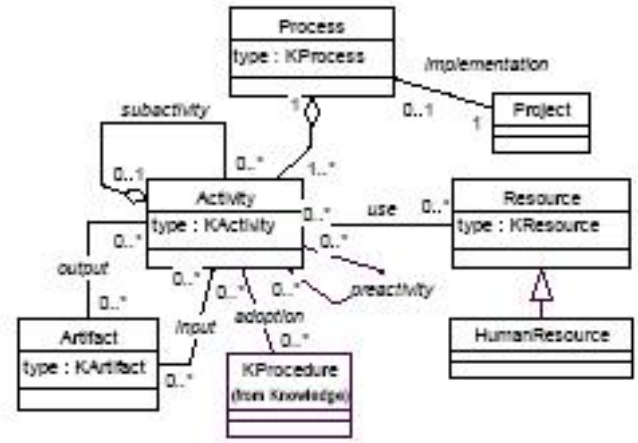


Figure 7.12: Software Process Base Level Model

Developers [González-Pérez and Henderson-Sellers, 2006]: The focus is to consider concepts and methodologies to be used both by method engineers and software developers when developing software. The ontology considers a three–layer architecture based on domains (metamodel, method and endeavor) Fig. (7.13).

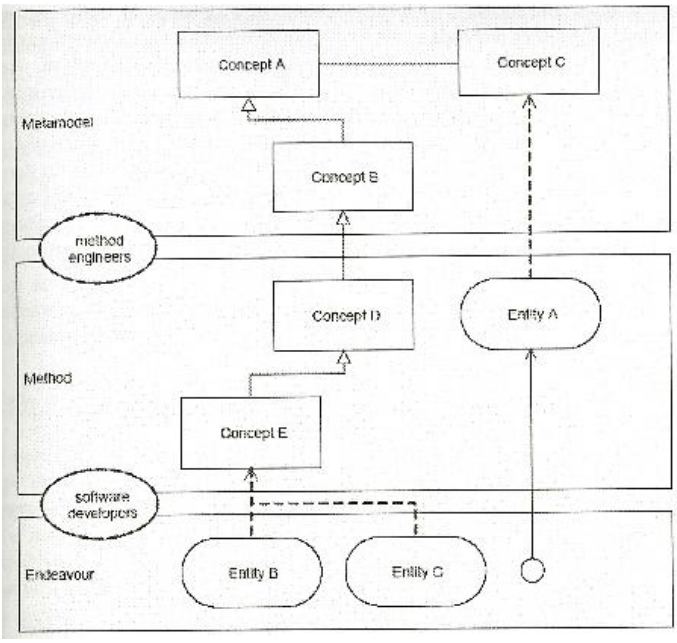


Figure 7.13: Architecture of Ontology for Software Development Methodologies and Endeavours

A concept is placed in a layer not based on structural relationships, but on *semantics* depending on the concept being a generic one, used by method engineers and by software developers.

Ontology–based Active Requirements Engineering Framework (Onto–ActRE framework)[Lee and Gandhi, 2005]:

A framework called Onto–ActRE was developed to elicit, represent and analyze the myriad of elements and factors involved in the development of software–intensive systems. Such systems' capabilities depend strongly on the interweaved relationships with sub–systems and environment.

The framework includes different models (based on well–establish requirements engineering techniques): goal–driven scenario composition (to model goals and objectives at different level of abstraction); requirements domain model (to model requirements from top–level to sub–domains); viewpoint hierarchy (to model viewpoints from different stakeholders, system's and environment's concerns); other domain taxonomies (to classify domain concepts, properties and relationships). As a result, the Problem Domain Ontology (PDO) is obtained (Fig.7.14).

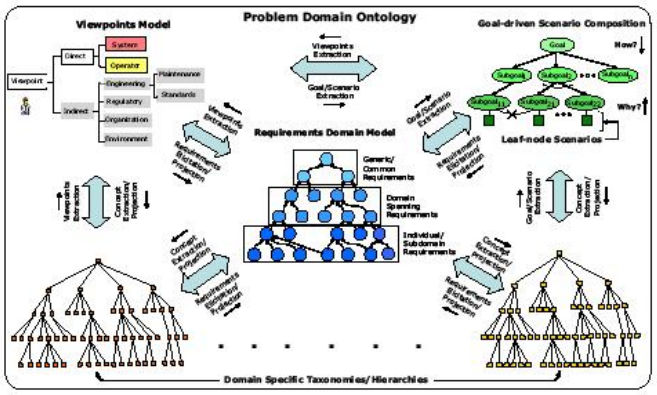


Figure 7.14: Onto-ActRE Problem Domain Ontology

The engineering processes to develop the different models are supported by the GENeric Object Model (GenOM) tool, which allows to create and handle knowledge bases associated with such processes.

The framework and its support tool has been used to the automation of the Department of Defense Information Technology Security Certification and Accreditations Process.

Software Process Ontology (SPO) [Liao et al., 2005]: the authors analyze two different software process models such as Capability Maturity Model (CMM) and ISO/IEC 15504 to define key concepts in software process (which are later mapped from one to another). The analysis allows for the definition of an ontology–based software process model framework which encompasses a Software Process Ontology (SPO), which defines the process model at a conceptual level. The SPO is used to build two different ontologies to fulfil for the two analyzed models. The use of the SPO and its extensions consists in allowing an user to check the description of the processes and practices recommended by the reference model.
Ontology based Requirements Elicitation (ORE) [Saeki, 2004], [Kaiya and Saeki, 2006]: the research proposes a requirement elicitation method based on a domain ontology. The artifacts used to elicit requirements are a requirement list provided by users in natural language, a domain ontology and finally, both a mapping and inference rules between the requirements list and the ontology concepts and relationships.

2. Software Technology This subcategory has not been included, as it was considered to fall apart from the research focus of this dissertation. It mainly concerns ontologies for software techniques, languages, data and information related technology and systems.

7.2 Ontologies for the Software Engineering Process: Ontologies as Software Artifacts

Within this category, ontologies are used as some kind of artifact during a software process.

1. At Development Time
   - *For Engineering Processes*

     Development process

     Domain Oriented Software Development Environment (DOSDE) [de Oliveira et al., 20
     the aim was to develop an environment to help software developers in domains which are not familiar to them. They considered two different types of knowledge: related to the application domain and related to tasks. The first kind of knowledge was implemented as a *domain ontology*, divided into sub–ontologies to model the main concepts of the domain. The second kind as a *task ontology* combined with a *Problem Solving Method (PSM)* into a single model (PST) that considered three different levels to describe both the tasks and the way to carry them out (verbal description using natural language; conceptual description as an intermediate stage describing both concepts and an algorithm to solve the task; and a formal description of the former using Prolog language to formalize both the concepts and inferences to solve the task). Additionally, the sub–ontologies of the domain ontology were mapped to the tasks of the PST. The environment was tested with knowledge domain regarding software requirements within a software development process for cardiology and acoustic propagation domains. An extension of DOSDE, to support also organizational knowledge has been implemented as an Enterprise–Oriented Software Development Environment (EOSDE) [Oliveira et al., 2006] whose components can be seen in Fig. (7.15).

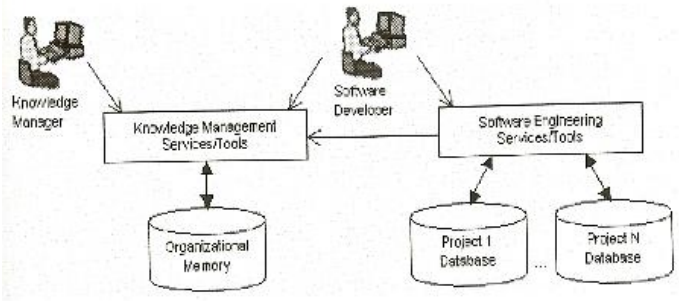


Figure 7.15: EOSDE components

Part of the Organizational Memory, it is the Enterprise Ontology, fundamental to support common vocabulary that represents use-

ful knowledge for software developers within an organization. It consists of different subontologies (intellectual capital, behavior, artifacts, structure and general strategy) to address different aspects of the enterprise.

Maintenance process

Software Maintenance Environment MANTIS [Ruiz et al., 2002]: In addition to define software maintenance concepts in the form of an ontology, the authors realized the necessity of an environment to handle software maintenance projects. They used the Maintenance Ontologies [Ruiz et al., 2004] as common conceptual framework to be used in MANTIS (Fig. (7.16). The ontology allows to share knowledge among those taking part in a software maintenance project. The ontology has also been used in a knowledge management system (KM–MANTIS).

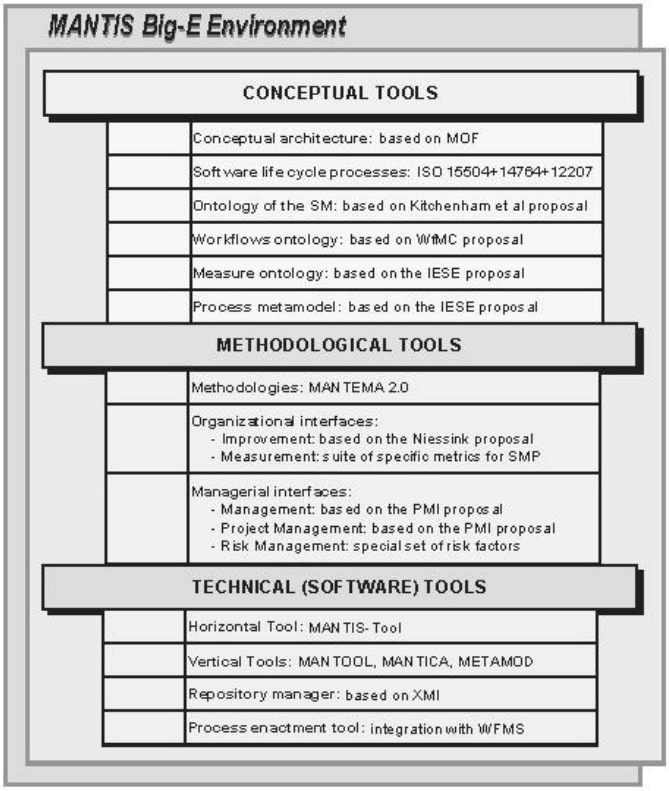


Figure 7.16: MANTIS Components

- *For Other Processes*

  Ontology–based Software Engineering for Multi–Site Software Development [Wongthongtham et al., 2005b], [Wongthongtham et al., 2005a]: the development of multi–site software projects involves the development teams to reside in different geographically dispersed

sites. Therefore, this kind of projects are prone to misunderstandings within teams communication. This research has focused on this kind of problems by using ontologies as both *common conceptualization and communication mechanism* among teams. The proposed solution considers two different ontologies: Generic ontology (to define terms, vocabulary, semantic interconnections, and rules of inference) and Application–specific ontology (to specify object–oriented development for a particular project). Benefits of this approach are the use of an explicit, formalized, machine–readable and consistent concepts among multi–site developers.

2. At Run Time

   - *As Architectural Artifacts (ontology–driven software)*

     Ontology Driven Architecture (ODA) [Force, 2001]:
     The W3C's Software Engineering Task Force has proposed the Ontology Driven Architecture (ODA), where an ontology describes the properties, relationships and behaviors of the components required in a software development process.
     A particular case of the above, is the scenario described in [Knublauch,] for the *tourism domain within the Semantic Web*. Ontologies are used for input (e.g. core travel and geography ontologies) and output (e.g. instances of them by service providers) data structures. Ontologies are furthermore used to represent background knowledge needed by an application to fulfil its task. It is worth mentioning it, that "domain models are not only used for code generation, but they are used as executable artifacts at run–time".
     Similarly, an ontology–based approach has been used to aid in the development of *software components in an application server* [Oberle et al., 2004]. The ontology is used as a conceptual model both for development and run–time of software components. The ontology is divided into generic modules, to model both semantic and syntactic metadata. Additionally, domain modules have been implemented to formalize knowledge specific to a certain application.
     Another example of the use of ODAs within the Semantic Web is the Telelearning Operating System (TELOS) [Magnan and Paquette,] devised to tackle learning and knowledge management problems within the LORNET pancanadian eLearning project [Project,]. Ontologies are used as conceptual models not only for the system requirements but as part of the system itself. Inference engines are used to carry out queries and reason with the core ontology.

   - *As Information Resources (ontology–aware software)*

     Ontology–based Infrastructure for Intelligent Systems [Eberhart, 2003]: ontologies are used twofold: for data integration from several sources and for intelligent systems operations. The use of ontologies is

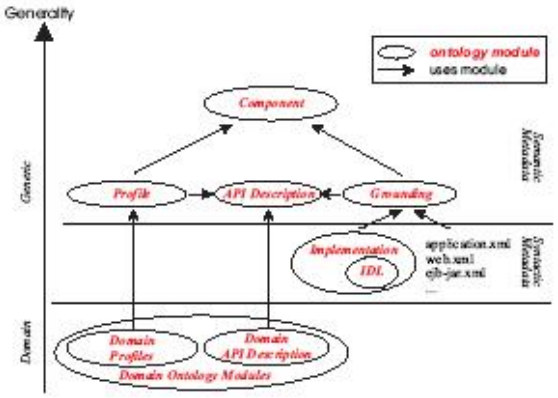


Figure 7.17: Ontology Modules for use cases supporting development

tested as part of a help system for online learning. *Ontology Driven Web Information System Environment (WISE)* [Tang et al., 2006]: ontologies are used as domain models to develop Web Information System. Two ontologies are considered: the domain ontology and the behavior ontology. The first one models static entities such as system structures and stored data [Tang et al., 2005a]. The second one describes dynamic processes such as system activities and interactions [Tang et al., 2005b].

Chapter 8

Conclusions

Software development processes are becoming increasingly complex, with shorter development times at lower cost. Therefore, the described research has pinpointed ontologies as a core element for software engineering.

On the one hand, ontologies allow for a common vocabulary both of the domain and different subdomains (requirements, design, maintenance, etc.). Ontologies provide a backbone for software projects as conceptualization of fundamental concepts and their relationships for a (sub)domain. Being too complex a task, several ontology–based software engineering research projects have tackled this complexity twofold:

Firstly, by considering different levels within the ontology. Despite the myriad of names, the underlying idea is to distinguish different levels to address from general to detail. In general, researchers have considered a general *meta–level* to describe the ontologies themselves. This level can be instantiated into a *knowledge level* which comprises the domain knowledge in a general and unspecified way, always with a minimal ontological commitment. When applied to specific applications, such knowledge is instantiated into, generally called, objects as an *application or base level*. How the instantiation has been made from level to level is worth considering it. Not all entities or classes from level to level will be instantiated.

Secondly, by dividing the knowledge (sub)domain under consideration in different subontologies. This allows to split the definition efforts, focusing on a topic (agents, structure, organization, etc.) at each time. Furthermore, subontologies also allow a more organized of the domain knowledge.

Another topic of concern, when ontologies are used to describe knowledge domain, has been the *semantic* issue. In other words, the problem on how to confer a meaning to the concepts encompassed in the ontologies. Axioms and rules are the usual means to provide for such meaning. Inference based on such axioms is used to expand and keep up to date the ontologies with new terms.

On the second hand, to support software development processes. Related to it, the underlying idea is that of ontologies being used at development time (as artifacts for software processes) or at run time (ontologies either as

component of the software system or as information resource). Possibly, the use of ontologies at run time as system's components open a wide range of possibilities for software re–use.

Despite the obvious benefits on the usage of ontologies, some issues still remain. The research projects described in former sections seem to show common features, as aforementioned. Nevertheless, except for some efforts that are based on both existing software engineering standards or the body of knowledge, the rest are developed ad-hoc according to the requirements of a specific software project. Methodologies and tools originally for ontology engineering, but not developed to fulfil software engineering requirements, have been used (except for [Tautz and Wangenheim, 1998]). Possibly, to reach a consensus among ontology developers within the software engineering community is too time and effort consuming. As a consequence, we will witness to a myriad of local developments which do not benefit from each other. *NOTE: this conclusion made when OMG efforts with ODM and MDA are not fully grasped. It might change in the future*

Bibliography

[Abran et al., 2006] Abran, A., Cuadrado, J., García-Barriocanal, E., Mendes, O., Sánchez-Alonso, S., and Sicilia, M. (2006). *Ontologies for Software Engineering and Software Technology*, chapter Engineering the Ontology for the SWEBOK: Issues and Techniques, pages 103–121. Springer-Verlag Berlin Heidelberg.

[Acuña and Juristo, 2005] Acuña, S. and Juristo, N. (2005). *Software Process Modeling*, volume 10 of *International Series in Software Engineering*, chapter Software Process Modeling: A Preface, pages xiii–xxiv. Springer NY.

[Acuña and Sánchez-Segura, 2006] Acuña, S. and Sánchez-Segura, M. (2006). *New Trends in Software Process Modeling*, volume 18 of *Series on Software Engineering and Knowledge Engineering*, chapter Preface, pages v–ix. World Scientific.

[Acuña et al.,] Acuña, S. T., Antonio, A. D., Ferre, X., Lopez, M., and Mate, L. The software process: Modelling, evaluation and improvement.

[Acuña and Ferre, 2001] Acuña, S. T. and Ferre, X. (2001). Software process modelling. In *Proceedings of ISAS-SCI (1)*, pages 237–242.

[Anquetil et al., 2006] Anquetil, N., de Oliveira, K., and Dias, M. (2006). *Ontologies for Software Engineering and Software Technology*, chapter Software Maintenance Ontology, pages 153–173. Springer-Verlag Berlin Heidelberg.

[Bertoa et al., 2006] Bertoa, M., Vallecillo, A., and García, F. (2006). *Ontologies for Software Engineering and Software Technology*, chapter An Ontology for Software Measurement, pages 175–196. Springer-Verlag Berlin Heidelberg.

[Brown et al., 2005] Brown, A., Conallen, J., and Tropeano, D. (2005). *Model-Driven Software Development*, chapter Introduction: Models, Modeling, and Model-Driven Architecture (MDA), pages 1–16. Springer-Verlag Berlin Heidelberg.

[de Oliveira et al., 2004] de Oliveira, K., Zlot, F., Rocha, A., Travassos, G., Galotta, C., and de Menezes, C. (2004). Domain–oriented software development environment. *Journal of Systems and Software*, (72):145–161.

[Dias et al., 2003] Dias, M., Anquetil, N., and de Oliveira, K. (2003). Organizing the knowledge used in software maintenance. *Journal of Universal Computer Science*, 9(7):641–658.

[Eberhart, 2003] Eberhart, A. (2003). *Ontology-Based Infrastructure for Intelligent Applications*. Phd thesis, University of Saarbrücken.

[Falbo, 2004] Falbo, R. (2004). Experiences in using a method for building domain ontologies. In *Proceedings of Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2004)*, pages 474–477, Alberta, Canda.

[Falbo et al., 2004a] Falbo, R., Arantes, D., and Natali, A. (2004a). Integrating knowledge management and groupware in a software development environment. In Karagiannis, D. and Reimer, U., editors, *Proceedings of 5th International Conference on Practical Aspects of Knowledge Management (PAKM' 2004)*, number LNAI 3336, pages 94–105, Vienna, Austria. Springer–Verlag Berlin Heidelberg.

[Falbo and Bertollo, 2005] Falbo, R. and Bertollo, G. (2005). Establishing a common vocabulary for software organizations understand software processes. In *Proceedings of EDOC International Workshop on Vocabularies, Ontologies and Rules for the Enterprise (VORTE'2005)*, Enschede, The Netherlands.

[Falbo et al., 2002] Falbo, R., Guizzardi, G., and Duarte, K. (2002). An ontological approach to domain engineering. In *Proceedings of 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 351–358, Ischia, Italy.

[Falbo et al., 1998] Falbo, R., Menezes, C., and Rocha, A. (1998). A systematic approach for building ontologies. In Heidelberg, S.-V. B., editor, *Proceedings of 6th Ibero–American Conference on AI*, number LNCS1484 in Lecture Notes in Artificial Intelligence, pages 349–360, Lisbon, Portugal.

[Falbo et al., 2003] Falbo, R., Natali, A., Mian, P., Bertollo, G., and Ruy, F. B. (2003). ODE: Ontology–based software development environment. In *Proceedings of IX Congreso Argentino de Ciencias de la Computación*, pages 1124–1135, La Plata, Argentina.

[Falbo et al., 2005] Falbo, R., Ruy, F., and Moro, R. (2005). Using ontologies to add semantics to a software engineering environment. In *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'2005)*, pages 151–156, Taipei, China.

[Falbo et al., 2004b] Falbo, R., Ruy, F., Pezzin, J., and Moro, R. D. (2004b). Ontologias e ambientes de desenvolvimento de software semânticos. In *Proceedings of IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'2004)*, volume I, pages 277–292, Madrid, Spain.

[Force, 2001] Force, S. E. T. (2001). Ontology driven architectures and potential uses of the semantic web in systems and software engineering.

[Girardi et al., 2004] Girardi, R., de Faria, C. G., and Balby, L. (2004). Ontology–based domain modeling of multi–agent systems. In *Proceedings*

of Third International Workshop on Agent–Oriented Methodologies (OOPSLA 2004), Vancouver, Canada.

[Girardi and Faria, 2003] Girardi, R. and Faria, C. (2003). A generic ontology for the specification of domain models. In Overhage, S. and Turowski, K., editors, *Proceedings of 1st International Workshop on Component Engineering Methodology (WCEM'03)*, pages 41–50, Erfurt, Germany.

[González-Pérez and Henderson-Sellers, 2006] González-Pérez, C. and Henderson-Sellers, B. (2006). *Ontologies for Software Engineering and Software Technology*, chapter An Ontology for Software Development Methodologies and Endeavours, pages 123–151. Springer-Verlag Berlin Heidelberg.

[Guarino, 1998] Guarino, N. (1998). Formal ontology in information systems. In *Proceedings of FOIS'98*, pages 3–15, Trento, Italy. IOS Press, Amsterdam.

[Happel and Seedorf, 2006] Happel, H. and Seedorf, S. (2006). Applications of ontologies in software engineering. In *Proceedings of 2nd International Workshop on Semantic Web Enabled Software Engineering (SEWE 2006)*, Athens, GA, U.S.A.

[Hesse, 2005] Hesse, W. (2005). Ontologies in the software engineering process. In Lenz, R., editor, *Proceedings of Tagungsband Workshop on Enterprise Application Integration (EAI2005)*, Berlin, Germany. GITO–Verlag.

[Hruby, 2005] Hruby, P. (2005). Ontology–based domain–driven design. In *Proceedings of Object-Oriented Programming, Systems, Languages And Applications (OOPSLA'05)*, San Diego, California, U.S.A.

[Kaiya and Saeki, 2006] Kaiya, H. and Saeki, M. (2006). Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *Proceedings of 14th IEEE International Requirements Engineering Conference,(RE'06)*, pages 189–198, Minneapolis/St. Paul, Minnesota, USA. IEEE CS.

[Kalfoglou, 2000] Kalfoglou, Y. (2000). *Deploying Ontologies in Software Design*. Phd thesis, University of Edinburgh.

[Knublauch,] Knublauch, H. Ontology–driven software development in the context of the semantic web: An example scenario with protègè/owl. In *Proceedings of International Workshop on the Model–Driven Semantic Web (MDSW2004*.

[Lee and Gandhi, 2005] Lee, S. and Gandhi, R. (2005). Ontology-based active requirements engineering framework. In *Proceedings of 12th Asia-Pacific Software Engineering Conference (ASPEC'05)*, Taipei, Taiwan. IEEE.

[Liao et al., 2005] Liao, L., Qu, Y., and Leung, H. (2005). A software process ontology and its application. In *Proceedings of Workshop on Sematic Web Enable Software Engineering (SWESE)*, Galway, Ireland.

[Magnan and Paquette,] Magnan, F. and Paquette, G. Telos: An ontology driven elearning os. In *Proceedings of 4th International Conference on Adaptive Hypermedia and Adaptive Web–based Systems (AH2006)*.

[Martín and Olsina, 2004] Martín, M. and Olsina, L. (2004). Towards an ontology for software metrics and indicators as the foundation for a cataloging web system. In *Proceedings of 4th International Conference Web Enginering (ICWE 2004)*, volume 3140 of *Lecture Notes in Computer Science*, Münich, Germany. Springer Berlin-Heidelberg.

[Mendes and Abran, 2005a] Mendes, O. and Abran, A. (2005a). Issues in the development of an ontology for an emerging engineering discipline. In *Proceedings of 1st Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering (ONTOSE)*, volume 163, Alcalá de Henares, Madrid, Spain. CEUR Workshop Proceedings.

[Mendes and Abran, 2005b] Mendes, O. and Abran, A. (2005b). Software engineering ontology: A development methodology. *METRICS NEWS*, (9):68–76.

[Menzel and Grüninger, 2001] Menzel, C. and Grüninger, M. (2001). *Formal Ontology and Information Systems*, chapter A Formal Foundation for Process Modelling, pages 256–259. New York, ACM Press.

[Oberle et al., 2004] Oberle, D., Eberhart, A., Staab, S., and Volz, R. (2004). Developing and managing software components in an ontology–based application server. In Jacobsen, H., editor, *Proceedings of Middleware 2004, ACM/IFIP/USENIX 5th International Middleware Conference*, volume 3231 of *LNCS*, pages 459–477, Toronto, Ontario, Canada.

[Oliveira et al., 2006] Oliveira, K., Villela, K., Rocha, A., and Travassos, G. (2006). *Ontologies for Software Engineering and Software Technology*, chapter Use of Ontologies in Software Development Environments, pages 275–309. Springer-Verlag Berlin Heidelberg.

[Project,] Project, L.

[Ruiz et al., 2002] Ruiz, F., García, F., Piattini, M., and Polo, M. (2002). *Advances in Software Maintenance Management: Technologies and Solutions*, chapter Environment for Managing Software Maintenance Projects, pages 255–290. Idea Group Inc.

[Ruiz and Hilera, 2006] Ruiz, F. and Hilera, J. (2006). *Ontologies for Software Engineering and Software Technology*, chapter Using Ontologies in Software Engineering and Technology, pages 49–102. Springer-Verlag Berlin Heidelberg.

[Ruiz et al., 2004] Ruiz, F., Vizcaíno, A., Piattini, M., and García, F. (2004). An ontology for the management of software maintenance projects. *International Journal of Software Engineering*, 14(3):323–349.

[Ruy et al., 2004] Ruy, F., Bertollo, G., and Falbo, R. (2004). Knowledge-based support to process integration in ODE. *CLEI Electronic Journal*, 7(1).

[Saeki, 2004] Saeki, M. (2004). Ontology-based software development techniques. *ERCIM News*, (58):14–15.

[Sicilia et al., 2005] Sicilia, M., Cuadrado, J., and Rodríguez, D. (2005). Ontologies of software artifacts and activities: Resource annotation and application to learning technologies. In *Proceedings of 17th Software Engineering and Knowledge Engineering Conference (SEKE'05)*, pages 145–150, Taipei, Taiwan.

[Society, 2004] Society, I. C. (2004). Guide to the software engineering: Body of knowledge 2004 version.

[Souza and Falbo, 2005] Souza, V. and Falbo, R. (2005). Supporting ontology axiomatization and evaluation in oded. In *Proceedings of VIII Workshop Iberoamericano de Ingeniería de Requisitos y Ambientes de Software (IDEAS'2005)*, pages 59–70, Valaparaíso, Chile.

[Tanasescu, 2005] Tanasescu, V. (2005). An ontology–driven life–event portal. Master, Computer Science.

[Tang et al., 2005a] Tang, L., Li, H., Pan, Z., Tan, S., Qiu, B., Tang, S., and Wang, J. (2005a). Podwis: A personalized tool for ontology developing in domain specific web information systems. In *Proceedings of 7th Asia Pacific Web Conference (APWeb 2005)*, Shanghai, China.

[Tang et al., 2005b] Tang, L., Li, H., Pan, Z., Yang, D., Li, M., Tang, S., and Ying, Y. (2005b). An ontology based approach to construct behaviors in web information systems. In *Proceedings of 6th International Conference on Web-Age Information Management (WAIM 2005)*, Hangzhou, China.

[Tang et al., 2006] Tang, L., Li, H., Qiu, B., Li, M., Wang, J., Wang, L., Zhou, B., Yang, D., and Tang, S. (2006). Wise: a prototype for ontology driven development of web information systems. In et al., X. Z., editor, *Proceedings of 8th Asia Pacific Web Conference (APWeb 2006)*, number 3841 in LNCS, pages 1163–1167, Harbin, China. Springer-Verlag Berlin Heidelberg.

[Tautz and Wangenheim, 1998] Tautz, C. and Wangenheim, C. V. (1998). REFSENO: A representation formalism for software engineering ontologies. Technical Report IESE-Report 015.98/E version 1.1, Fraunhofer Institute for Experimental Software Engineering.

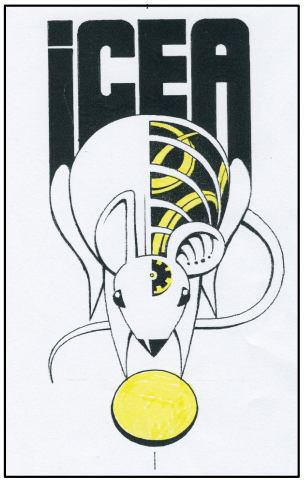
[Vizcaíno et al., 2005] Vizcaíno, A., Anquetil, N., Oliveira, K., Ruiz, F., and Piattini, M. (2005). Merging software maintenance ontologies: Our experience. In *Proceedings of 1st Workshop on Ontology, Conceptualization and Epistemology for Software and Systems Engineering (ONTOSE)*, volume 163, Alcalá de Henares, Madrid, Spain. CEUR Workshop Proceedings.

[Wongthongtham et al., 2005a] Wongthongtham, P., Chang, E., and Cheah, C. (2005a). Software engineering sub-ontology for specific software development. In *Proceedings of 29th IEEE/NASA Software Engineering Workshop (SEW'05)*, pages 27–33, Greenbelt, Maryland.

[Wongthongtham et al., 2005b] Wongthongtham, P., Chang, E., and Dillon, T. (2005b). Towards ontology-based software engineering for multi-site software development. In *Proceedings of 3rd IEEE International Conference on Industrial Informatics (INDIN)*, pages 362–365, Perth, Australia.

Document information

Reference: ASLab-ICEA-R-2006-016 v 0.1 Draft
Title: Ontology-based Software Engineering
Subtitle: Engineering Support for Autonomous Systems
URL: http://www.aslab.org/documents/ASLab-ICEA-2006-016.pdf
Date: 2006-11-15



EUROPEAN INTEGRATED PROJECT IST-027819
Integrating Cognition, Emotion and Autonomy

www.ist-icea.org