

Towards a Machine Enabled Semantic Framework for the Distributed Engineering Design

In One Volume

Vasile Ovidiu Chira B.Sc.

June 2004

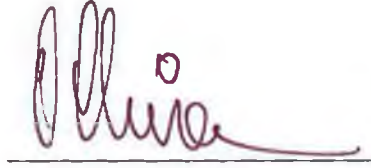
Submitted for the degree of
Doctor of Philosophy

Submitted to: Galway Mayo Institute of Technology
Research carried out at: The CIM Research Unit, National University of Ireland,
Galway, Ireland.
Galway Mayo Institute of Technology, Ireland

Research Director: Thomas Roche B.Eng, M.Eng., PhD

Declaration

I hereby declare that the work presented in this thesis is my own and that it has not been used to obtain a degree in this university or elsewhere.

A handwritten signature in red ink, appearing to read 'V. Chira', is written above a horizontal line.

Vasile Ovidiu Chira

Dedication

To my mother.....

Acknowledgments

There are many to whom I wish to thank for their impact on my scientific self and, consequently on this thesis. However, for objective reasons, only few of them will be mentioned here, since they are the ones that have directly influenced this research work. However, if I am to forget someone please accept *mea culpa*.

To my wife **Camelia** for quite **everything**. I am at this point of my thesis (i.e. writing acknowledgements, thus ending the thesis) more because of you than because of me. So, please accept my never-ending *appreciation* pal even when my moods were far from that want to leave. Thanks for reading, suggesting more essential, for listening and my quest to find the light of inspiration and during the last four years. Indeed, even if you the approach research section, you are an essential part of it. Moreover, I couldn't finish this work without your help in implementing the OSA prototype. I am grateful.



To **Thomas Roche** for his guidance and especially for the cultural and intellectual 'clashes' that formed and informed me as a researcher.

To **Elena Man** for the essential assistance in understanding the engineering design domain.

To **Attracta Brennan** for reading, re-reading, re-re-reading, ..., my chapters and suggesting such bright alternatives. I am impressed.

To me, Cami, Tom, Attracta, Eli, David and Valerie for being a fine having-a-good-craic research team ☺. To Laurentiu and Aurora for their friendship.

To all the lads from CIMRU.

Published Work Associated with this Thesis

Ovidiu Chira, Camelia Chira, David Tormey, Attracta Brennan, Thomas Roche, "An agent-based approach to knowledge management in distributed design", CE2003, Selected for Special Issue of Journal of Intelligent Manufacturing, 2004.

Ovidiu Chira, Camelia Chira, Thomas Roche, Attracta Brennan, "Semantic Tools for Knowledge Management in Distributed Engineering Design", 10th International Conference on Concurrent Enterprising Escuela Superior de Ingenieros, Seville, Spain, June 14-16, 2004.

Ovidiu Chira, Camelia Chira, David Tormey, Attracta Brennan, Thomas Roche, "A Multi-Agent Architecture for Distributed Design", International Conference on Applications of Holonic and Multi-Agent Systems HoloMAS 2003, Prague, September 1-3, 2003.

David Tormey, Camelia Chira, Ovidiu Chira, Thomas Roche, Attracta Brennan, "Development of Engineering Design Methodologies and Software Tools to Support the Creative Process of Design in a Distributed Environment", International Conference on Engineering Design ICED 03, Stockholm, August 19-21, 2003.

David Tormey, Ovidiu Chira, Camelia Chira, Attracta Brennan, Thomas Roche, "The Use of Ontologies for Defining Collaborative Design Processes", 32nd International Conference on Computers and Industrial Engineering, University of Limerick, August 11-12, 2003.

Ovidiu Chira, Camelia Chira, David Tormey, Attracta Brennan, Thomas Roche, "An agent-based approach to knowledge management in distributed design", 10th ISPE International Conference on Concurrent Engineering: Research and Applications, Madeira Island, Portugal, July 26-30, 2003.

Contents

List of Figures	viii
List of tables	ix
Chapter 1 Introduction	1
1.1 Research Background	2
1.2 Thesis Motivation	2
1.3 Aim and Objectives	3
1.4 Approach to Research	5
1.5 Thesis Structure	6
Chapter 2 The Distributed Engineering Design System	9
2.1 Introduction	10
2.2 Distributed Engineering Design	10
2.2.1 Engineering Design	11
2.2.2 The Distributed Engineering Design Organization	13
2.2.3 A Distributed Engineering Design Model	17
2.2.4 Information Setbacks in Distributed Engineering Design	19
2.2.5 Changing the perspective	21
2.3 A Systemic Approach to Distributed Engineering Design	23
2.3.1 Systems Theory and Cybernetics	24
2.3.2 The Distributed Engineering Design System (DEDS)	26
2.3.3 A Distributed Engineering Design System Model	34
2.3.4 Information Setbacks in Distributed Engineering Design System	37
2.4 Distributed Engineering Design System Requirements	38
2.5 Conclusions	41
Chapter 3 Distributed Technologies	43
3.1 Introduction	44
3.2 Ontologies	44
3.2.1 Background	45
3.2.2 Definition	47
3.2.3 Typologies	54
3.2.4 Methodologies for Building Ontologies	56
3.3 Software Agents	65
3.3.1 Background	65
3.3.2 Definition	67
3.3.3 Typologies	70
3.3.4 Multi-Agent Systems	73
3.3.5 Final Remarks	78
3.4 Conclusions	79
Chapter 4 Semantic Framework for the Distributed Engineering Design System	80
4.1 Introduction	81
4.2 Thinking the Distributed Engineering Design System	82
4.2.1 Prerequisites	82
4.2.2 The Cooperation Process	83
4.2.3 Cooperation Processes as the Main Information Flow Patterns	88
4.3 The Need for Semantic Support	91
4.3.1 A DEDS Requirements Analysis	92

4.3.2 Cooperation Process – the Semantic Enablers	96
4.3.3 Ontologies and Software Agents – the Technological Enablers	98
4.4 Proposed Framework for Enabling Semantics within the DEES	100
4.4.1 The Architectural Framework	101
4.4.2 The Architectural Layers	103
4.4.3 The Architectural Planes	105
4.5 Conclusions	111
Chapter 5 An Instantiation of the Proposed Architectural Framework	113
5.1 Introduction	114
5.2 An Ontology-based Software Agent (OSA) System	114
5.2.1 The Proposed OSA System	116
5.3 An OSA Prototype	120
5.3.1 Prototype Characterization	121
5.3.2 The Ontological Plane	122
5.3.3 The Software Agents Plane	127
5.4 Conclusions	133
Chapter 6 Conclusions and Future Work	134
6.1 Research Summary	135
6.2 Research Results	136
6.3 Further Development and Recommendations for Future Work	139
6.4 Final Remarks	141
References	144
Appendix 1 The Semantic Web	154

List of Figures

Figure 1.1 Approach to research.....	6
Figure 1.2 Thesis layout.....	7
Figure 2.1 The role of the computer in the design project space (MacGregor 2002).....	16
Figure 2.2 A Distributed Engineering Design model.....	18
Figure 2.3 The main concepts of an open system (Bennett, McRobb et al. 1999).....	29
Figure 2.4 A high-level model of the DEDS.....	34
Figure 2.5 A systemic approach to distributed engineering design.....	39
Figure 3.1 Possible interpretations of the term “ontology” after (Guarino and Giaretta 1995).....	48
Figure 3.2 Conceptualization-language-ontology relation (Guarino 1998).....	49
Figure 3.3 Kinds of ontologies, according to their level of dependence on a particular task or point of view (Guarino 1997; Guarino 1998).....	53
Figure 3.4 Uses for Ontologies (Uschold 1996).....	54
Figure 3.5 Ontology life cycle in the Methontology approach (Fernandez-Lopez 2001).....	62
Figure 3.6 The Ontological Reengineering process.....	63
Figure 3.7 Scope of intelligent agents (adapted from Gilbert et al. by (Bradshaw 1997))..	68
Figure 3.8 Nwana’s agent typology (Nwana 1996).....	69
Figure 3.9 The taxonomy of agents proposed by Franklin and Graesser (Franklin and Graesser 1996).....	71
Figure 4.1 The fundamental model of communication (VanCuilenburg, Scholten et al. 1991).....	82
Figure 4.2 A black-box view on DEDS as an information transformation system.....	86
Figure 4.3 High-level DEDS – Environment interaction.....	87
Figure 4.4 The transformation of information from I_1 to I_2 is performed through the cooperation of Human System, Engineering Design Model System, and Infrastructure System.....	88
Figure 4.5 The proposed DEDS Architectural Framework.....	99
Figure 4.6 The bi-plane model view of the DEDS architectural framework.....	103
Figure 5.1 An ontological model of the <i>service</i> concept.....	114
Figure 5.2 The proposed OSA system to support the access to design information.....	116
Figure 5.3 Prototype’s Ontology Library.....	120
Figure 5.4 The Protégé 2000 view of the Product Ontology.....	121
Figure 5.5 The Protégé 2000 view of the Material Ontology.....	122
Figure 5.6 A part (two subassemblies and two components) of a Smoke Alarm design information.....	123
Figure 5.7 The GUI of cami:MyAgent agent.....	125
Figure 5.8 Cami has requested the Material Browse service.....	126
Figure 5.9 The material_browse Service Provider’s GUI presented to the requester.....	127
Figure 5.10 The providing of the SearchProduct service by the OSA Prototype.....	128
Figure 5.11 The GUIs presented to the user cami by the ProductSearch Query Builder agent and RDQL agent.....	129
Figure 6.1 Research progression from implicit and heterogeneous knowledge towards explicit and homogeneous knowledge.....	137

List of tables

Table 3.1 Example of an ontology from (Benjamins, Fensel et al. 1998).....	51
Table 3.2 The Ontology Development Process.....	60
Table 3.3 Various definitions of an agent.....	65

Chapter 1

Introduction

1.1 Research Background

1.2 Thesis Motivation

1.3 Aim and Objectives

1.4 Approach to Research

1.5 Thesis Structure

1.1. Research Background

This thesis is the outcome of the research work the author has carried out as part of a team (i.e. CODE) within two projects funded by Enterprise Ireland, i.e. Environmental Enterprise Services Ireland (EESI) and Intelligent Agent Based Collaborative Design Information Management and Support Tools (IDIMS).

The goal of the EESI project (<http://pan.nuigalway.ie/eesi/>) is the formation of an Internet based environmental services company for small and medium enterprises in domains such as Design for Environment, Life Cycle Costing, Reverse Logistics, PDM Systems, Standards and Legislation. The research area covers data and project management, electronic communication, e-learning and the European environmental legislation.

The IDIMS project (<http://pan.nuigalway.ie/idims/>) investigates the use of Software Agents, Ontologies and Semantic Web to support the synthesis and presentation of information for distributed teams for the purposes of enhancing design, learning, creativity, communication and productivity.

While both projects required the investigation of the engineering design domain, (i) the EESI project focused on the development and the *integration* of engineering design software tools into a Web-based environment and (ii) the IDIMS project proposes software tools to support the designer's decision-making process, in order to facilitate the *interoperation* among distributed design environment (DDE) participants during the various design activities and to *manage* design knowledge.

1.2 Thesis Motivation

Once tests and validations were carried out, some limitations of the EESI project became clear. These limitations mainly derived from the lack of appropriate ICT tools to support the collaboration and knowledge exchange within distributed environments. Limitations of the existing ICT tools for distributed environments can be classified, from author's experience within the EESI project, under three headings, i.e. *Data Management*, *User Distribution* and *System Architecture*, as follows:

1. *Data Management* issues included:

- *Fragmentation*. It has become vital for modern distributed environments to be able to deal with structured data and data in a specific context (i.e. information and knowledge). For this to happen, a knowledge management system and not a data management system should be designed and developed.
- *Data repository is not truly distributed*. Even if modern database systems such as Oracle can deal with clusters of database servers, the administration of such

systems requires high skills and as a result can be resource demanding (i.e. time, specialists, IT infrastructure).

- *Information and Knowledge Reusability*. Because data is taken out of its context when it is saved (e.g. data *in* a CAD application makes a visual sense, but, when saved, it represents strings of signs without meaning *outside* the CAD tool), it becomes almost impossible to implement a reusability policy (such as for object oriented programming) for complex information structures in a distributed environment.
- *Semantic Search Engines*. There is no way for a search engine *to know* what the user truly wants and as a result can only perform retrievals based on key words and simple queries.

2. *Users Distribution* issues included:

- The *semantic distribution* of users could be difficult to be implemented because of the impossibility of translating data from one specialised language into another (for example engineering specific language versus administrative specific language or 'jargon'). Such translation can be only carried out with a semantic translation mechanism (which is not available with existing systems).
- Generally, users perform a variety of tasks that do not require any special skills (e.g. filling all sorts of reports with information that is already available within the environment, translating information form visual representations into written words), tasks that in the existing distributed collaboration environments are not automated (mainly because the distributed systems are typically highly heterogeneous).
- The users are required to perform time-consuming activities in order to synthesize information and knowledge from data (mainly because they can only obtain simple data and not direct information and knowledge).

3. *System Architecture* issues included:

- *Inherent distribution*: The data, information, knowledge, users and technology in a distributed environment are highly heterogeneous and there is no standard *representation* with which to work.
- *Inherent complexity*: The distributed environments tend to become too large to be solved (in some cases it is even impossible) by a single, centralised system because of hardware and software limitations, difficulty, and the resulting increase in time and cost. Distributed decision making is a particularly complex

task to implement in distributed environments, particularly for the design and development of environmentally superior products where a diverse expertise is required to inform the design process.

Closely related to the above problems, the author considers that one of the main difficulties encountered within the EESI project was the integration of various engineering design tools and users at levels that go beyond the exchanges of simple strings of data. In order to achieve this integration goal, the author found that it was essential to enable exchanges/connections of complex information structures among the concerned engineering design actors. Moreover, the author also found that, in order to fulfill their purpose, the services offered through the Web portal had to be tailored to their particular users both at presentational and content levels. Therefore, the same service had different layout appearances and information content for different users (i.e. user with different needs). This situation further complicated the development of the Web framework that would amalgamate software tools having different approaches (usually concealed) for representing their working information to users with/having a wide variety of profiles.

The investigations carried out for finding solutions to the above-mentioned problems, while contributing in part to the concretization of the IDIMS project, form the motivation behind the present research work. It is intended to discover and to adopt an holistic view of the distributed engineering design organization, in order to reveal the intimate mechanisms capable of controlling and regulating information related functionalities. Moreover, the author intends to uncover how these mechanisms can be steered by means of software applications in order to improve the performance of the engineering design actors.

1.3 Aim and Objectives

The overall aim of this thesis is to identify and propose a suitable architectural framework for supporting cooperation processes and therefore enabling semantics within the distributed engineering design environment. The proposed architecture is intended to characterize a software-based management of design related data, information and knowledge flows in the distributed engineering design organization. The aim is to provide a computational context for implementing ICT tools that would:

- (i) *Minimise* the effect of user and resource dispersion (particularly temporal and geographical dispersion), the misunderstandings that might be generated by the (otherwise beneficial) functional and semantic distribution, the time spent for searching and retrieval of information, the effort of information translation

between different tools and the administrative and organisational efforts not directly related to the design process (e.g. revision control)

- (ii) *Maximise* the quality of information (i.e. relevant information at relevant and appropriate times), knowledge sharing and reuse among distributed design actors, the flexibility of the user interfaces and the designer's time spent in the actual designing process.

In order to achieve the overall aim, the research work supporting this thesis was carried out along the following objectives:

1. To investigate and characterize the engineering design process performed in a distributed environment and its problematic aspects;
2. To research and study alternative theories for thinking and modelling the distributed engineering design process;
3. To investigate current research in information and knowledge management for identifying supporting technologies for a possible solution to the identified problematic aspects (from point 1);
4. To analyze the requirement needs for a solution according to the findings from previous objectives, i.e. the driving problems (from point 1), the research and therefore the thinking approach (from point 2), and available supporting technologies (from point 3);
5. To synthesize the architectural framework along the identified supporting technologies (from point 3);
6. To instantiate a software system along the underlying computational context as described by the architectural framework (from point 5).

1.4 Approach to Research

The identification of an architectural framework to fulfil the aim and objectives of this research requires a good understanding of the application domain (i.e. the distributed engineering design), its key stakeholders and its shortcomings. Moreover, given the complexity and the extent/vastness of the distributed engineering design organization, the author has identified and focused his research on those aspects concerning the information flow dynamics within the distributed design environment (DDE).

Given that the proposed research work is the result of the authors thinking and interpretations steered by existing scientific theories (i.e. Systems Theory, Cybernetics, Systems Thinking, Ontologies and Agent-based Systems), the approach to this research is presented in figure 1.1.

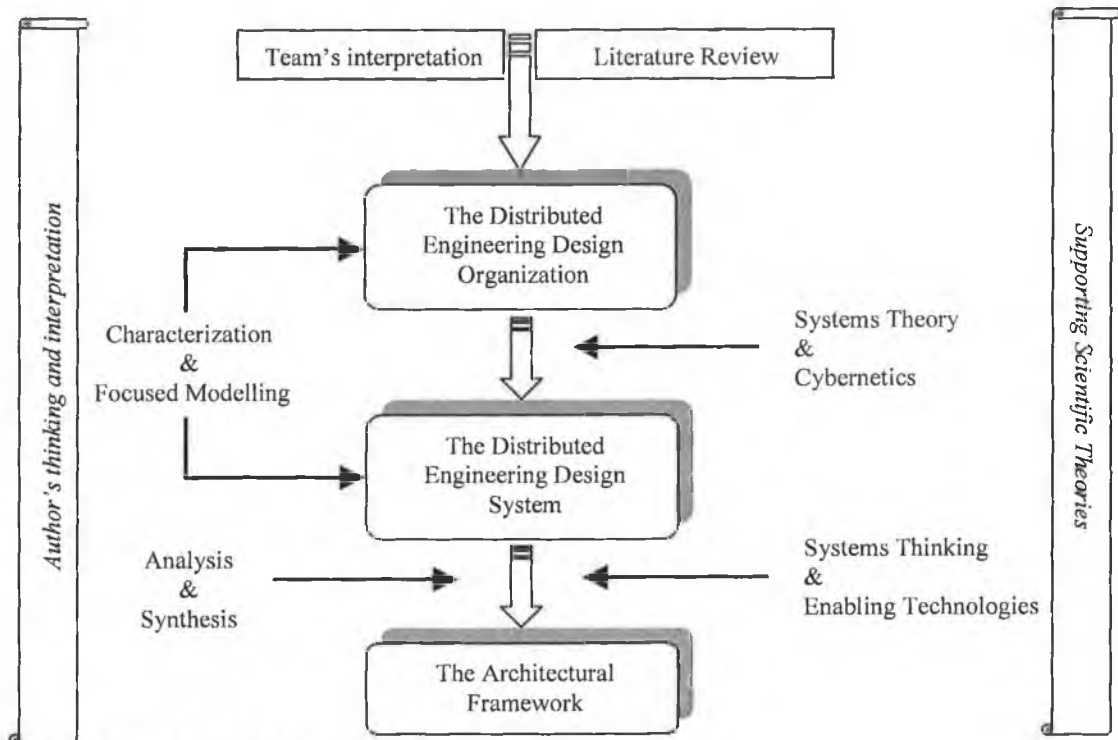


Figure 1.1 Approach to research.

A comprehensive literature review, carried out by the author as part of a research team, accounts for this thesis application domain, i.e. the Distributed Engineering Design. However, specific characteristics of the distributed engineering design (such as complexity, distribution, multi-disciplinarity, highly reliance on information exchanges, finality) resulted in the author's adoption of General Systems Theory and Cybernetics as conceptualising and modelling frameworks. The goal of using such approaches is to cognitively model the perception of the distributed engineering design organization, i.e. the Distributed Engineering Design System, so that deeper insights are attainable. In this way, a characterization and a model of the Distributed Engineering Design System are inferred. Furthermore, a Systems Thinking guided analysis of the achieved results is synthesized in the proposed Architectural Framework.

1.5 Thesis Structure

The present thesis is structured into six chapters. A diagrammed map, presented in figure 1.2, summarizes this thesis layout.

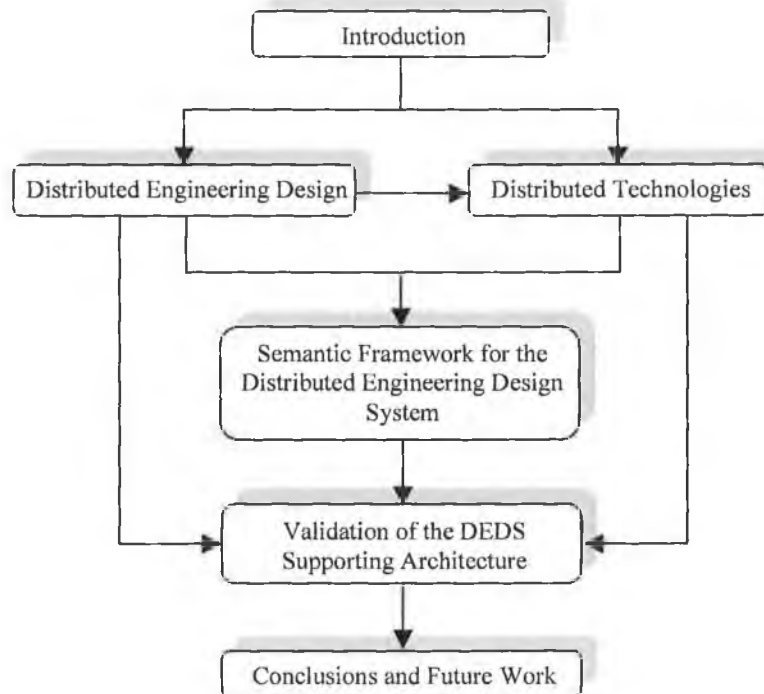


Figure 1.2 Thesis layout.

Chapter 2. Distributed Engineering Design

Chapter two introduces and characterizes the application domain of the research work, i.e. the engineering design process performed within a distributed environment. It states the research focus, i.e. the information flow dynamics, and identifies the driving problematic aspects.

Subsequent to identified limitations of the conventional results, the author proposes a novel approach to dealing with the distributed engineering design organization, i.e. the systemic approach. Following the principles of Systems Theory and Cybernetics, it is argued and demonstrated that actually, the distributed engineering design organization is an open cybernetic system (called by the author DEDS) 'kept together' by information structures and cooperation processes dynamics. Such a rethinking, in this new context, of the distributed engineering design organization and its shortcomings, results in a set of requirements for a computer-based solution to the identified problems.

Chapter 3. Distributed Technologies

This chapter presents two of the most promising research areas for implementing semantically enabled environments, i.e. Ontologies and Software Agents. These two technologies are envisioned, throughout literature, to form the next distributed

computational environment, capable of managing inherent complex and inherent distributed systems.

Chapter 4. Semantic Framework for the Distributed Engineering Design System

This chapter argues that a hierarchy of cooperation process forms the directorial pattern of the DEDS functionality, and therefore determines its performance. It is found that the cooperation processes form (by means of communication processes), enable (by means of co-location processes), regulate (by means of coordination processes) and support (by means of collaboration processes) intra-system information-mediated *interactions*. Consequently, it is shown that, while the semantics can enrich the cooperation processes, the cooperation processes, in turn, can support the preservation of semantics among the DEDS components.

Given that the poor semantic integration of the information structures into the whole has been identified as a key problem of the DEDS, the author propose an architectural framework to define the computational context for enabling semantics for the cooperation processes within a distributed engineering design environment.

Chapter 5. An Instantiation of the Proposed Architectural Framework

This chapter proposes an ontology-based software agents system (called OSA) as a conceptual instantiation of the architectural framework, with the purpose of testing and validating the results of the current research, i.e. the proposed architecture. In turn, the OSA system will be validated by means of implementation of an OSA prototype.

Chapter 6. Conclusions and Future Work

This is the last chapter of this thesis. The main conclusions of the research are discussed and recommendations are given for further research in this area. A special attention is given to the potential of the Semantic Web, subject to which the author gave an entire annex (see annex one).

Chapter 2

The Distributed Engineering Design System

2.1 Introduction

2.2 Distributed Engineering Design

2.3 A Systemic Approach to Distributed Engineering Design

2.4 Distributed Engineering Design System Requirements

2.5 Conclusions

2.1. Introduction

This chapter offers insights into applying a holistic approach to the research in distributed engineering design, i.e. the systemic approach, which “unifies and concentrates on the interaction between elements, studies the effects of interactions and emphasizes a global perception” (Rosnay 1979; Rosnay 1997).

The chapter opens with a characterization of the application domain of this thesis, i.e. the distributed engineering design organization, focusing on the importance of information structures. Firstly, the key assumptions and characteristics, based on the appropriate literature, about the concept of *engineering design* are illustrated and described. Next, the distributed design activity is introduced and portrayed, and its significant aspects are identified and analyzed. Based on the specific findings, a working model of the distributed engineering design organization is proposed and investigated. Furthermore, some of the critical information related problems are identified.

Based on the conclusions of the literature review, the need for a new and innovative approach orients the research towards a systemic perspective. The systemic viewpoints offer the means of describing and demonstrating that the distributed engineering design organization behaves as and therefore is an open cybernetic system, called DEDES. A systemic adaptation and reorganization of the proposed distributed engineering design model results in a high-level model of the DEDES. Furthermore, once the portrait of the DEDES has been drawn, an interpretation of the information related problems (as identified from literature) is performed and the appropriate conclusions are presented.

Based on the knowledge gained, the chapter concludes with a set of preliminary requirements necessary to enhance distributed engineering design organization functionality and to reduce its negative aspects. Technologies to implement the specifications are also anticipated and documented.

A set of final remarks concludes the present chapter.

2.2 Distributed Engineering Design

In the context of today’s business environment, the engineering design activity involves multiple clusters of users requiring concurrent access to multiple system resources and collaborating in a distributed design environment in order to achieve global optima (Chira, Chira et al. 2003). One of the reasons for this is that “complex design problems generally

require more knowledge than any one single person possesses because the knowledge relevant to a problem is usually distributed among stakeholders” (Arias, Eden et al. 2000). The above state of affairs offers a preliminary insight into the complexity of the application domain of the research that underlines the present thesis. Because of this and because of the various scientific positions and points of view found in literature regarding the *engineering design* concept and its derivatives, the author considers helpful and necessary to state and characterize his understanding of the field.

2.2.1 Engineering Design

Before characterizing the concept of engineering design as understood throughout the thesis (i.e. as a process), a terminological clarification is first performed. *Engineering design* as a *science* is defined as “a body of intellectually tough, analytic, partially empirical, teachable doctrine about the design process” (Simon 1996). Hubka and Eder proposed a more formal definition, as follows: “Design science comprises a collection (a system) of logically connected knowledge in the area of design and of design methodology...Design science addresses the problem of determining and categorizing all regular phenomena of the system to be designed, and of the design process. Design science is also concerned with deriving from the applied knowledge of the natural sciences appropriate information in the form suitable for the designer’s use” (Hubka and Eder 1987). In summary, the science of design is in fact “a system of knowledge” (Eder 1998) that investigates the design as its object.

Engineering design as a *discipline* (Cross 2000; Gero 2000; Love 2002) incorporates the specific interdisciplinary features of the design science in an accurate learning environment. Hence, the discipline of design defines rules, regulations and methodologies that form the framework for studying and teaching design.

While the meaning of *engineering design* as a *process* is somehow elusive in spite of its widespread use, attempts have been made to define it. Even though a universally accepted definition of the design process has not yet been agreed upon, nevertheless, a large number of proposed definitions exist (Feilden 1963; Finkelstein and Finkelstein 1983; Luckman 1984) that cover or focus on different aspects of the engineering design process domain. The literature mainly deals with the following aspects:

1. The process of design itself (Luckman 1984; Pugh 1991; Hubka and Eder 1996; Roche 1999; Gero 2000),

2. The designer as the main agent of the process (Roche 1999; Gero 2000),
3. The finality of the design process or the objectives of design activity (Feilden 1963; Pugh 1991; Lang, Dickinson et al. 2002),
4. The life-cycle information aspect that integrates the design in a more holistic view of the product realization process (Eder 1998; Roche 1999; Lang, Dickinson et al. 2002).

1. A generic *process* is understood as “a series of actions or operations conducing to an end” (Merriam-Webster 2003). For the process of design in particular, the actions can be sequential or parallel, or combined and the end purpose is a solution to a design problem (Smith and Morrow 1999). From the various definitions expressed in literature (Luckman 1984; Pugh 1991; Hubka and Eder 1996; Gero 2000), explicitly stated or implicitly supposed, this process emerges to be three-fold. Firstly, engineering design is an information transformation process from initial formal or informal requirements and constraints towards a final artifact or product that fulfils these requirements in varying degrees of performance characteristics (ideally the final artifact will fulfill all requirements under the given constraints). Secondly, a problem solving process assists the transformation of information process by employing specific methods that help establish a path from the initial conditions to an acceptable optimal solution. Thirdly, a decision making process is involved in making the right choices at the appropriate time in the problem solving process (Boer 1989).

2. The designer, and mainly his/her cognitive activities, is a critical element of any engineering design process, as, the designer is the decision-maker. Throughout literature (Roche 1999; Gero 2000), characteristics such as skills, experience, knowledge, imagination, originality and creativity are mentioned as key words associated with the designer. The designer, usually collaborating with other designers, is the one who carries out the design process and the design evolves through designers’ negotiation strategies with each other (Brereton, Cannon et al. 1994).

3. The objectives of engineering design converge in a “structure, machine or system to perform pre-specified functions with the maximum economy and efficiency” (Feilden 1963). Besides the accomplishment of the initial requirements under the initial constraints, any end product generally has some significant other characteristics such as: it responds to consumer demand, it is economically manufactured, it fulfills a human need, and so on

(Feilden 1963; Pugh 1991; Lang, Dickinson et al. 2002). This final artifact is the pragmatic result of the design process exertion, an activity that is supported by the designer's intellectual competences, experience and knowledge.

4. Life-cycle information represents a novel perspective of the design process that integrates it into the wider picture of product design (Eder 1998; Lang, Dickinson et al. 2002). As products become more and more complex, the need to bring them to market quickly and at low cost requires that expertise be shared along the supply chain. Following this life-cycle line, engineering design is the starting phase of a process consisting of a set of interrelated phases that also include manufacturing, supply, use, maintenance and disposal. Therefore, engineering design is no longer a process on its own, disconnected from other processes. It has been fused to support and to be assisted by the broader process of product realization by means of needed and available knowledge (from the other phases).

In summary, the author views engineering design as an information transformation process performed by qualified human designers. The input consists of abstract statements of design requirements and the output represents detailed information that specifies the product (Hubka and Eder 1996; Chira, Chira et al. 2003). The successful accomplishment of the engineering design process depends on the designers' problem solving skills and on the decisions they make at the various junctures of design, based on the available life-cycle information. Furthermore, the author identified that a key actor in engineering design is *information*. Designing means to alter some initial information structures, through a series of stages generally identified as requirement definition, functional specifications, conceptual design and detailed design (Roche 1999), until a suitable structure is achieved and outputted. The engineering designers alter the information based on their expertise (which in turn is based on their personal knowledge), on the available methodologies, methods and tools, and even more importantly, based on the information they can straightforwardly access. Moreover, because of the various functionalities which today's product has to cover (e.g. assembly, disassembly, usage, manufacturing, disposal), the outputted information needs to incorporate informative structures from and for its different life stages.

2.2.2 The Distributed Engineering Design Organization

Competitive pressures and the constraints caused by the complex demands of today's markets such as quick time to market, low cost, high quality, low environmental impact

and increased customization trigger the necessity of transition to better-suited forms of design (Tomiyaama 1994; Kimura 1997; Hirsch 2000; Thoben 2002).

Therefore, even if the engineering design process remains the same in its substance, the way it is carried out and the resources involved needed to adapt to the changing characteristics of the business environment.

Globalization is probably one of the main forms of adaptation to these market and legislative pressures. It launched the concept of a *distributed organization* that describes “an organization which distributes its work to the best locations for their execution based on the criteria of people skills, costs and resources” (Gammack and Poon 1999).

Following this trend, better forms of carrying out the engineering design activity have been developed. This globalization and reorganization of the engineering design activity is called *distributed engineering design* (Pahng, Senin et al. 1997; Gammack and Poon 1999; Lang, Dickinson et al. 2002; MacGregor 2002). Hence, distributed engineering design is seen as a strategy of organizing the engineering design activity for “exploiting the knowledge and expertise of all parties involved, including marketing, engineering, design, management, suppliers, production, [...], in the design team, no matter how these parties are distributed geographically and organizationally” (Lang, Dickinson et al. 2002).

Some of the key defining characteristics of distributed engineering design, identified in literature, are as follows (Olsen, Cutkosky et al. 1994; Cross and Cross 1995; Harvey and Koubek 1998; Ahn, Roundy et al. 1999; Siemieniuch and Sinclair 1999; Chen and Lee 2002; Lang, Dickinson et al. 2002; MacGregor 2002):

1. *Distribution*
2. *Teamwork*
3. *Cooperation*
4. *Computer's role*

1. *Distribution.*

Design projects have become increasingly larger and more complex and have started to require a multidisciplinary approach (Cutkosky, Englemore et al. 1997). Therefore, more designers dispersed in different geographic locations and coming from different disciplines can become involved in the same project (Olsen, Cutkosky et al. 1994; Siemieniuch and Sinclair 1999). Moreover, together with the designer, design data, design information and design knowledge are generally highly dispersed (Cross 1994; Pahl and Beitz 1996) (Bertola and Teixeira 2003). Using Weiss' (Weiss 1999) typology, the author identifies

four kinds of *distribution*, each at two levels (i.e. the human level and the information resources level), as follows:

- *Geographical distribution*: the users and the information resources are dispersed in different geographical locations;
- *Temporal distribution*: the users participate within a distributed environment at different zones of time (e.g. 1 p.m. in Ireland means 10 p.m. in Japan), while the information resources can arise (e.g. become available) at different moments of time;
- *Functional distribution*: the users and the information resources are structured in clusters defined by specific perceptual, effectual and intellectual capabilities.
- *Semantic distribution*: the users and the information resources are structured in clusters defined by specific languages and conceptual realities.

2. *Teamwork*

To manage distribution towards a common, unified goal, effective *teamwork* is essential (Olsen, Cutkosky et al. 1994; Cross and Cross 1995) (Siemieniuch and Sinclair 1999). As problems become more complex, design is not anymore an individual activity, but a collective effort (Patel, D'Cruz et al. 1997). Designers need to be aware of each other and must work together and collaborate in order to meet the design's objectives.

3. *Cooperation*

The implementation of a viable organization consisting of dispersed human designers clustered in virtual teams with access to distributed information resources requires a robust collaboration process (Lawson 1990; Brereton, Cannon et al. 1994; Olsen, Cutkosky et al. 1994; Harvey and Koubek 1998; MacGregor 2002). In the literature, collaboration is the main concept used to describe the process of bringing and linking together humans and information resources in a functional distributed organization. Following the line of Pena-Mora et al (Pena-Mora, Hussein et al. 2000), the author believes that the concept of *cooperation* should be used instead, the reason being that *cooperation* refers to the different processes of co-operation between humans in collaborative environments and consists of the following sub-processes (Pena-Mora, Hussein et al. 2000):

- *Communication*: exchange of information, events and activities between participants;

- *Co-location*: infrastructure to provide effective communication among distributed participants;
- *Coordination*: management of the workflow, resources, information and communication process;
- *Collaboration*: the process of creation of a shared understanding in a distributed environment, enabling in this way the communication process.

Therefore, while the collaboration process may implicitly subsume the other three processes, for terminological clarity, this thesis adopts the notion of cooperation that explicitly defines the meaning of ‘coming together’ of design actors (human and non-human) in a distributed design environment.

4. Computer's role

Besides a meaningful cooperation of distributed teams, an important aspect of a modern engineering design activity is the *role of the computer*. Since its early introduction, the computer has acted as an advanced tool for designer usage. In the same time with the evolution of the engineering design process, the computer has acquired more and more important roles (see figure 2.1).

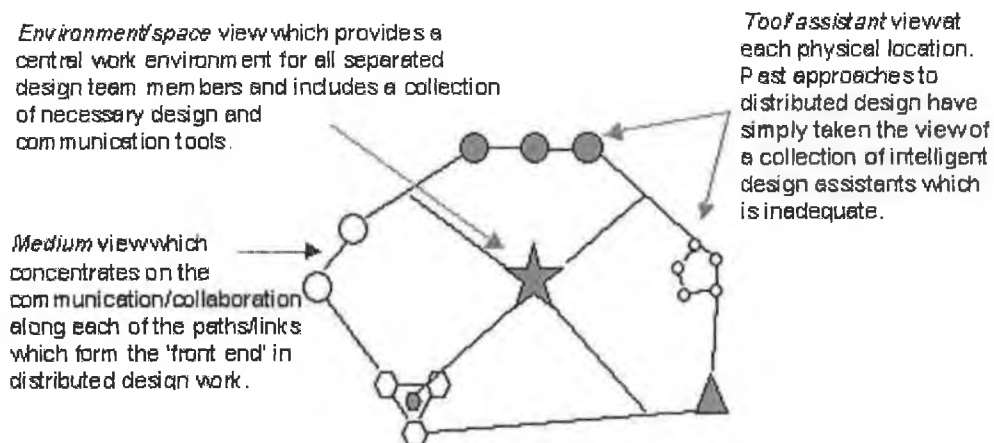


Figure 2.1 The role of the computer in the design project space (MacGregor 2002).

Today, the computer (or computer networks) acts as a medium or workplace: “a suite of tools, necessary to support the human designer, both for actual design work and communication” (MacGregor 2002).

The environment in which distributed engineering design takes place can be called a distributed design environment (DDE). Therefore, a DDE consists of distributed designers and distributed information resources networked by computers and software tools. A

cooperation process is employed to support the geographical, temporal, functional and semantical integration of all engineering resources (e.g. designers, information, tools, methods and methodologies).

An important characteristic of such an environment, caused by the distribution of designers and design resources, is its heterogeneity (Pahng, Senin et al. 1997). Designers are heterogeneous (diverse disciplines, experience, expertise and knowledge), computers are heterogeneous (different hardware and software platforms) and the software tools used by designers are heterogeneous (different software tools developers).

2.2.3 A Distributed Engineering Design Model

Design models are schematic, simplified representations of the patterns (or structures) and functionalities of the design process and are used to represent the design process.

Generally, a classic engineering design model (e.g. descriptive, prescriptive, mathematical, computational, life-cycle) does not include any references to designers and design infrastructure. This is because a generic designer is implicitly assumed and the infrastructure (e.g. computers) is not seen to have a critical impact on a supposed collocated design process (after all, computers are considered only when developing software tools for designers).

For clarity and working reasons, an adaptation of such a generic model to *distributed* engineering design is required. In order to represent the distributed engineering design activity (i.e. how design takes place) in accordance with the findings from the previous section, the author contends that a modified model should explicitly include references to designers and infrastructure. Moreover the model should emphasize the cooperation process that links designers, information resources, tools, methods and methodologies (i.e. engineering resources) in a feasible organization that can carry out the engineering design process.

The distributed design model used in this thesis intends to represent the inherently distributed and heterogeneous nature of the design activity and reflects the critical importance of the design information resources, as they are the main material for any cooperation.

Figure 2.2 presents distributed engineering design as an information transformation process effected by a human component and enabled by an infrastructure component. The *human component* subsumes the distributed *multidisciplinary* design teams involved in the design process. The *infrastructure component* (e.g. computer networks) enables the design

process by creating the medium or the designing space and also by providing advanced designing software tools (e.g. CAD tools). The *engineering design model* defines and determines the specifications of the methods and the methodologies used to carry out the design process. Finally, everything is functionally integrated in a common functional organization by means of the cooperation process that envelops the internal information flows used to harmonize, regulate and control the engineering design process. A key characteristic of these information flows is that they depend on and are influenced by the structures that produce, communicate, transform or consume them, e.g. human, infrastructure, design specific structures, as well as any combination of them. A result of the diversity of structures involved in the distributed engineering design is the highly diverse and often irreducible representations used to embody and codify the information structures.

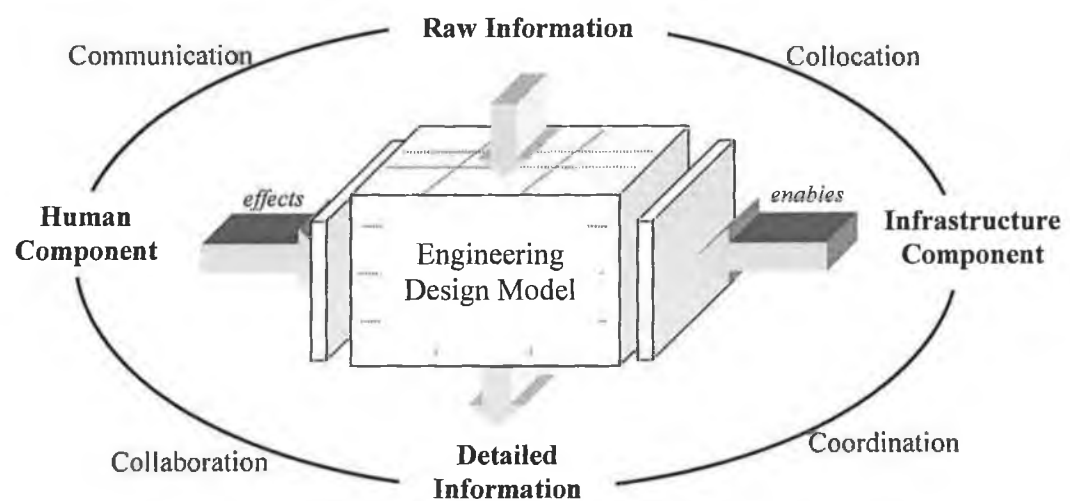


Figure 2.2 A Distributed Engineering Design model

In summary, the distributed design organization is the result of complex *cooperation enabled inter/intra-actions* among engineering designers, design methods, methodologies and tools, and information-communication technologies (ICT). Moreover, engineering design is considered an increasingly complex process applied on a pool of *desirably* shareable information by a team of designers *desirably* diverse (because the author thinks that diversity is advantageous in developing new insights and design ideas) in their disciplines.

The author believes that the design information structures represent a critical element of the engineering design, and moreover of distributed engineering design, since the process of designing can be viewed as an information transformation process (as shown in section 2.2.1). For this reason, this thesis focuses on the management of the above-mentioned information structures. More explicitly, the research concentrates on how to manage the information that needs to be exchanged or shared among different distributed engineering design components, in order to support a sound and feasible cooperation process.

2.2.4 Information Setbacks in Distributed Engineering Design

Because of the complex nature of distributed design, the associated problems are generally multifaceted. While some of these problems can be localized (e.g. infrastructure specific problems), the majority of the difficulties cover more than one of the distributed design components (e.g. a malfunction may be caused by poor infrastructure quality *and* communication difficulties *and* inadequate information). These circumstances only serve to complicate the study of the distributed design organization.

Regardless of the variety of distributed design problems that exists, this thesis focuses specifically on information related problems. Moreover, the author argues that the malfunctioning of the cooperation process causes most of the problems associated with distributed design. Furthermore, the main cause of cooperation faults can be ascribed to the lack of an efficient and effective management of design related data, information and knowledge (Thoben, Weber et al. 2002).

The author recognized that, with respect to the design data, information and knowledge, two aspects need to be considered, i.e. quantitative and qualitative aspects. The *quantitative* aspect is related to the unprecedented growth of the quantity of information (VanCuilenburg, Scholten et al. 1991). Ho claims that currently the overall amount of information that the world produces is in the range of one to two exabytes (a billion gigabytes) per year (Ho and Tang 2001). As this volume of knowledge and information increases it is naive and dangerous to assume that any one designer would be capable of grasping all aspects and nuances of a problem. As a result of these information and knowledge overloads, designers can often become perplexed, because they do not know how to handle such vast quantities of information and knowledge effectively for their design work (Chira, Chira et al. 2003). In modern design environments products are so complex that externalised information and knowledge must be readily accessible by the designer. It is widely believed that design engineers spend up to 47% of their time seeking

design information in the design process (Hales 1987), with no more than 10% being machine-readable (McGee and Prusak 1993). Given that design can be described as a problem solving process and considering that engineers tend to solve problems based on *available knowledge*, it is important to ensure that data, information and knowledge are available in enough quantities and at the correct time in the process (Lawson 1990; Cross 1994; Hubka and Eder 1996; Pahl and Beitz 1996; Roche 1999).

The *qualitative* aspect of design data, information and knowledge relates to the increasingly critical problem that arises for the designers to find information that is *relevant* or appropriate to the task at hand (Viano 2000). Research has shown that design engineers have strong problem solving skills but they are generally poor at creating imaginative ideas and conceptual models, and hence depend largely on the *quality* of the information which they have *ready* access to (Kolb 1984; Finger and Dixon 1989; Coyne, Rosenman et al. 1990; Brennan 1996; Hubka and Eder 1996; Roche 1999). If proper information is not easily accessible design engineers are unlikely to seek or share knowledge and expertise and as a result, at best are likely to sacrifice quality and to generate local rather than global 'optima', and thus sub-optimal design solutions (Coyne, Rosenman et al. 1990; Lawson 1990; Roche 1999).

Both the quantitative and the qualitative problematic aspects have the potential to be particularly augmented in the distributed environment because of the distribution of data, information and knowledge, the inherent dynamic nature of design information, the virtual communication processes and the increased complexity of products (Jagdev and Browne 1998; Roche 1999; Pena-Mora, Hussein et al. 2000).

The shortfall of data, information and knowledge management impinges on distributed engineering design and especially on the cooperation aspect of it. The literature acknowledges the low level of awareness and understanding of other designers and their work (Nakakoji, Yamamoto et al. 1998; Selater, Grierson et al. 2001; MacGregor 2002; Thoben, Weber et al. 2002). Moreover, because of the different languages, backgrounds, experience and expertise of the design stakeholders meaning is particularly difficult to transfer and communicate (Snow 1993; Harvey and Koubek 1998; Brazier, Moshkina et al. 2001; MacGregor 2002; Thoben, Weber et al. 2002). The sharing of knowledge and information becomes even more difficult in an environment where the tools are developed by and for experts (Cutkosky, Englemore et al. 1997) and their syntactic integration into

the distributed design environment is reduced (Crabtree, Fox et al. 1997; Siemieniuch and Sinclair 1999; Pena-Mora, Hussein et al. 2000).

To summarize, the problems associated with a distributed engineering design environment can be categorized as follows:

- The quantity of information structures (especially information and knowledge) which the engineering designers have and need to handle/manage (e.g. search, identify, retrieve, use/process, store) is already burdensome and is increasing at a rapid pace.
- Designers' knowledge and, therefore, performance depend on the readily available information, which is not always of the appropriate quality.
- The above problems result in an unsatisfactory cooperation within the organization and, consequently the intensification and recurrence of the same problems.

2.2.5 Changing the perspective

The existing approaches to distributed design generally apply a kind of “divide et impera” (i.e. divide and conquer) methodology. This consists on identifying a certain aspect of a distributed engineering design process that is simple enough to be studied. The result of the study is usually materialized in a software tool that will improve a specific functionality or will resolve local problematic issues. For example, the CAD tools such as ProEngineer and SolidWorks improve the drawing ability of designers by providing a visual 3D environment. In the author's view, this approach led to a number of different strategies or logics of organizing the design data and information, specific to the application in hand. Some applications store the data in specific files while others use database systems (such as Oracle or MySql). Moreover, specific to distributed engineering design, the human actors generally come from different backgrounds and specialties, and hence have their own professional and conceptual languages. The result is a semantical, functional and organizational ‘breakdown’ of the design information resources. Therefore design information sharing, reuse and integration are more a desirable situation than a reality. This can lead to poor cooperation among the distributed design structures, which in turn can result in difficult and costly design process management.

Given this situation, the author considers that, for achieving his research aim, the thesis needs a change of perspective from a **local** reductionist approach towards a more **holistic** approach to studying and implementing the distributed engineering design organization. This is because it is not intended to provide a software-based solution to a particular aspect

of the distributed engineering design process, but to enable a global software-based solution to the information setbacks of the distributed engineering design organization. However, this change of research perception needs to be informed, sustained and validated by established scientific theory(ies). Moreover, given that the research is in an early stage, the decisions taken at this point are having a milestone importance for the future work. With all of these in mind, the author contends that the process of selecting the needed scientific support is 'apriori' directed by the followings:

- The author's mental models (which are the result of his personal history e.g. experience, education and formation) and expertise
- The 'immediate' feedback from the 'close-by' researchers (e.g. colleagues, corresponding fellow researchers)

'A posteriori', this selection process is supported (in the sense that is positively or negatively confirmed) by the research results obtained at the different stages of the research work.

As has been said above, the author proposes a change of perspective from local views to a global view. Within the research community it is quite generally agreed that a global approach actually necessitate a **systemic approach**. Therefore, the first criterion of the selection process was to study and identify appropriate **system's** theories in order to find the needed support. The theories studied included General Systems Theory, Cybernetics, Complexity Theory, Chaos Theory, Systems Dynamics, Adaptive Systems, Autonomic Systems and Holonic Systems. A first conclusion of the investigations has been that all these theories have (almost) the same understanding of the concept of **system**, the differences generally consisting in the tools they are using for exploring this concept and the fields of science that immediately benefit of the findings (e.g. probably the main beneficiary of the Chaos Theory findings is meteorology). Therefore, given the research's time and other resources constraints, the other criteria of the selection process were the author's proficiency with the tools used by the different theories and the grade of immediate benefit for the distributed engineering design domain. For example, the Complexity Theory and Systems Dynamics seem to have instant results if used in the research of distributed engineering design domain. However, they both require advanced mathematics which, from the author's expertise point of view would have been necessitate some time to acquire. On the other hand, the author had the expertise for using Adaptive Systems and Holonic Systems. However, it was not obvious for the author that these two theories would have a straight impact on the research since they are mainly used in sociology and biology. Therefore, based on his expertise and informed by discussion with

specialists in engineering design the author has selected the General Systems Theory (sometimes called only Systems Theory) and Cybernetics as the supporting scientific theories. Another reason for this selection is that these two theories are somehow more general than the other theories (and all the other theories can be seen as specializations of one or the other of these two) and can be seen as the first steps towards a consistent and systematic employment of system's theories in studying the engineering design domain.

2.3 A Systemic Approach to Distributed Engineering Design

The author believes that the achievement of the (distributed) design goal, i.e. to reach a good solution for a quality product with the least commitment of time and resources, necessitates a seamless administration of the organization as a whole. However, such an administration requires a throughout understanding of the distributed engineering design organization and its problematic aspects. Moreover, the understanding, or the knowledge that can be acquired is constrained by *Weltanschauung* or the *worldview* one has about the object in question (in this case distributed engineering design). In the following, based on the literature, the author presents the overall perspective from which he sees and interprets distributed engineering design.

If in the past, a designer or a group of co-located designers used to be sufficient to perform an engineering design project, currently, the large-scale engineering design tasks require the involvement of globally distributed “multi-disciplinary teams of individuals” (Olsen, Cutkosky et al. 1994). From the literature, the author synthesized the following main characteristics of such a design team (Harvey and Koubek 1998), (Siemieniuch and Sinclair 1999), (Ahn, Roundy et al. 1999), (Lang, Dickinson et al. 2002), (Olsen, Cutkosky et al. 1994):

- Distribution: - team members are geographically, temporally, functionally and semantically dispersed.
- Multi-disciplinarity: - the team consists of members originating from different engineering disciplines and/or scientific backgrounds.
- Concurrency: - the functioning of the entire process is conditioned by different stages of the workflow that depend on each other.
- Parallelism: - team members can be required to work in parallel when it is possible for a faster completion of the design process.

The above acknowledged performer of the distributed engineering design process (i.e. the distributed design team) together with the supporting methodologies and technologies (e.g. computer networks, design tools, supporting software applications) form the distributed engineering design organization. Hence, distributed design emerges as a kind of complex organization headed towards an explicitly specified goal. It depends on the information exchanges with its environment and has control mechanisms that monitor and manage its internal functionality.

To summarize, the author contends that distributed engineering design is not some kind of machine or organization built or manufactured in a laboratory, but a consistent set of self-organizing structures interacting with certain conformity to some internal laws towards a pragmatic goal. It is an evolutionary response to the forces that act in its environment (e.g. market and legislative forces). In this light, distributed design can be viewed as a *system* or a distributed engineering design system (DEDS). Additionally, not only is DEDS a system but it will be also argued that it is an *open cybernetic system*.

This thesis offers insights of applying a different approach to the research in distributed design, i.e. the systemic approach, as opposed to a reductionism approach. The systemic approach “unifies and concentrates on the interaction between elements, studies the effects of interactions and emphasizes a global perception” (Rosnay 1979; Rosnay 1997). However, an introduction to systems theory and cybernetics is firstly required to formally define the concept of systems.

2.3.1 Systems Theory and Cybernetics

Systems Theory has its roots in the belief that there is an implicit order in the Universe (however complex and diverse the Universe is) (Rosnay 1979; Heylighen, Joslyn et al. 1993). It evolved as a field of science (also under the name of General System Theory or GST) in parallel with a number of philosophical views, i.e. constructivist, functionalist and holistic, as a balance to the reductionist approach (Heylighen, Joslyn et al. 1993; Skyttner 1996).

The theory is built on the concept of a *system*, which has accompanied human thinking since its early history (Skyttner 1996). There are many definitions (both strong and weak) of a system (see (Skyttner 1996; Backlund 2000) for a review), from which two are more appropriate to this thesis. The first one is a pragmatic definition circulated in the realm of management, and considers that “a system is the organized collection of men, machines

and material required to accomplish a specific purpose and tied together by communication links” (Skyttner 1996). The second definition has a more formal articulation and states that a system is made by a set with a cardinality¹ at least two and satisfies the following three conditions (Ackoff 1981):

1. The behaviour of each element has an effect on the behaviour of the whole.
2. The behaviour of the elements and their effects on the whole are interdependent.
3. When subgroups of the elements are formed, all have an effect on the behaviour of the whole but none has an independent effect on it.

Hence, a system is an heterogeneous collection of interdependent elements and group of elements kept together by an overall purpose. It is not necessary for every element or group of elements from the system to have specific local goals, but generally, the local goals work together to achieve a global goal. In addition to organization and goal directness, there is one more necessary condition required for something to qualify as a system, i.e. “continuity of identity” (Skyttner 1996) or the capacity to preserve structure within a changing environment.

Systems Theory is defined as the “trans-disciplinary study of the abstract organization of phenomena, independent of their substance, type, or spatial or temporal scale of existence. It investigates both the principles common to all complex entities, and the (usually mathematical) models which can be used to describe them” (Heylighen and Joslyn 1992). Moreover these *organizations* can be described and modelled by concepts and principles that are independent of a specific domain (Heylighen, Joslyn et al. Oct 1, 1993 (created)). Evidently, this model will not describe the *real* piece of the world exhaustively (Geyer 1994), but it will provide the researcher with (the necessary and) sufficient approximation for studying it (Skyttner 1996; Backlund 2000; Heylighen, Joslyn et al. Oct 1, 1993 (created)).

The roots of *Cybernetics*² come from a variety of disciplines including mathematics, technology, biology, information theory, system dynamics, chaos theory and Artificial Intelligence (Geyer 1994; Beer 2002; Heylighen, Joslyn et al. Oct 1, 1993 (created)), and originated from a series of problems and projects concerned with feedback loops and control systems for constructing intelligent machines. One of the earliest examples is

¹ Cardinality measures the number of elements of a given (mathematical) set.

² Cybernetics derives from the Greek word for steersman (*kybernetes*) Heylighen, F., C. Joslyn, et al. (Oct 1, 1993 (created)). What are Cybernetics and Systems Science? [Principia Cybernetica Web](#). F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels.

Shannon and Weaver's problem of reducing noise in telephone lines (Shannon and Weaver 1963), which led to the development of information theory (Corning 2001).

The mathematician Norbert Wiener (whom is considered to be the father of cybernetics) proposed one of the first well-received definitions. He defines cybernetics as "the science of control and communication in the animal and the machine" (as cited by (ASC; Heylighen; Beer 2002)). The definition stresses the close interdependence between the concepts of *communication* and *control*. In order to control, communication is necessary. Moreover, the concept of *control* does not mean "pulling levers to produce intended and inexorable results" (Beer 2002), but refers to the observation that no matter how complicated and unpredictable a non-trivial system is, "something can be done to generate a predictable goal" (Beer 2002). Furthermore, the control is applied upon animal *and* machine. In this way a unification of animal and human made artifacts is envisioned and intended. Naturally, this definition has evolved towards an "interdisciplinary approach to organization, irrespective of a system's material realization" (Heylighen). The main focus of cybernetics is not the *thing* itself but the design and discovery of principles of regulation and control (ASC).

At their core, Cybernetics and Systems Theory focus, in fact, on the same object, i.e. *organization* independent of substrate (Joslyn 1992; Geyer 1994; Heylighen, Joslyn et al. Oct 1, 1993 (created)). They differentiate in the approach they take to investigate their object. While systems theory focuses on "the *structure* of systems and their models" (Heylighen, Joslyn et al. Oct 1, 1993 (created)), cybernetics focuses on "how systems *function*, that is to say how they control their actions, how they communicate with other systems or with their own components" (Heylighen, Joslyn et al. Oct 1, 1993 (created)). Because the similarities are considered to be greater than the differences, the two domains (i.e. cybernetics and system theory) have practically merged. (Heylighen, Joslyn et al. 1993; Geyer 1994).

2.3.2 The Distributed Engineering Design System (DEDS)

The research carried out at the author's institution has investigated the structure, the characteristics, the properties and the functionality of the distributed engineering design organization. Accordingly with observations made,

Based on characterization of the distributed engineering design organization (from section 2.2) the author considers that a different approach to studying the distributed design is required, an approach that perceives the distributed design as a whole entity, i.e.

Distributed Engineering Design System or DEDS. The initial phase of this approach consists of three steps, as follows:

- (1) It is argued that a generic DEDS complies with the characteristics of a generic system, so therefore a DEDS is a system.
- (2) More specifically, the structure of a generic DEDS is described in terms of an open system.
- (3) The behaviour of a generic DEDS is expressed from a cybernetic perspective.

(1) Distributed Engineering Design as a System

Based on Skyttner's review of the literature, the hallmarks of system theory are as follows (Skyttner 1996):

- *Interrelationship and interdependence of objects and their attributes* – the elements that constitute a system are related and or dependent on each other. Unrelated and independent elements do not constitute a system.
- *Holism* – the overall system is more than the sum of its parts and this should be possible to be defined in the system.
- *Goal seeking* – a system functions towards a goal or an equilibrium point.
- *Transformation process* – to achieve its goal, a system must transform inputs into outputs. This, in fact, defines the overall functionality of the system.
- *Inputs and outputs* – The system takes some raw resources (e.g. information, matter, and energy) as inputs and transforms them into some final forms, according to its goal(s).
- *Entropy* – the amount of disorder and randomness present in any system (entropy) tends to increase in time. When maximum entropy is reached the system cease to function. However, it is possible to delay the increase of entropy by importing additional resources (usually energy).
- *Regulation* – the components of a system have to be controlled and regulated, so that the global goal(s) are achieved. This regulation includes the corrections of deviations, so a feedback mechanism is required.
- *Hierarchy* – a system generally consists of a hierarchy of several sub-systems.
- *Differentiation* – or division of labor, implies that specialized units within the system perform specialized functions.

- *Equality and multifinality* - there is more than one way to reach an objective that complies with the overall goal(s) and, in the same way there is more than one objective (mutually exclusive) that complies with the overall goal(s).

Within a DEDES, human designers geographically distributed collaborate using communication technologies (e.g. web cams, microphones, collaboration applications) and negotiations strategies for the sharing of data, information and knowledge. Locally, the designers use past cases and CAD tools for drawings, and stored (in books, reports, hard drives, own memory and so on) information and knowledge for ideas or insights. Hence, a DEDES consists of *interrelated* and *interdependent* elements.

Moreover, the DEDES itself is more than the sum of the individual designers, computer networks and data, information and knowledge stored in the system. The complex interactions among the distributed design components add value to the whole, value that is not achievable by simply summing the individual values. It matters how and what information is stored, the time to access it and the imaginative response of the designer to the information in hand. Therefore, the DEDES has the property of *holism*.

As mentioned earlier, the necessity or need for a DEDES is triggered by specific legislative, market and business needs and the primary *goal* is to generate better product specifications in decreasing time and cost. The achievement of the goal necessitates employing an information *transformation process* from raw information (*input*) towards a detailed design (*output*).

However, the author contends that, without market feedback, or in the absence of evolutionary tools and methodologies the DEDES will fail to provide proper solutions and will finally disintegrate, i.e. it will reach its maximum *entropy*.

For the organization to function, management mechanisms are necessary to *regulate* and control the function of the design components together with their intra/interactions. These administrative components, usually consisting of humans (e.g. designers, team leaders, project managers, etc) helped by specific tools (e.g. PDM), based on the information at hand to take decisions during the 'unfolding' of the engineering design process.

Depending on the state of the engineering design process or on the specific type of intra/interactions required, specific tools are employed to handle it. For example, a collaboration tool (such as IBM Lotus Sametime (IBM 2003)) together with specific devices (e.g. web cams, headphone sets) can support a virtual brainstorming session among a team of designers at the early stages of designing, while CAD tools (such as ProEngineer or SolidWorks) are successfully used at later stages for visually representing the product

structures. Such *differentiation* of labor can also be observed within the humans involved in distributed engineering design organization (e.g. vertical hierarchies, horizontal division of tasks, etc).

Finally, it is possible and probable to exist/find more than one final product specification (there are a number of global 'optima') that is able to fulfill the initial requirements and constraints, and there is more than one path that will lead from the initial conditions to a global 'optimum'. For example, even if the requirements for wireless telephony are almost all the same, then the final product (i.e. mobile phone) differs from producer to producer even if, after all they are approximately equal in what they are doing.

To conclude, a generic DEDS complies with the generic system characteristics, and therefore acts as a system. The question is, does it behave as an open or closed system?

(2) DEDS as an Open System

The Newtonian model of the world, assumes that the studied system is closed (Hawking 2001). This means that the system does not interact with the outside environment, so no exchange of matter and energy with the exterior occurs. Under this assumption, any system contains inside its boundaries all the information needed to analyze and predict its behavior. Hence, the study of the system is reduced to observations of the phenomena that take place within the system. This reductionism simplifies the study of the considered system, and in some cases the predictions correspond to what is observed (Barrow 1992; Hawking 2001).

Nevertheless, this approach is "too often narrow and inclined towards a restricted area" (Skyttner 1996). In his General Systems Theory, Bertalanffy argued that, there are systems in the world that depend on matter and energy exchange with the environment (which in fact are the most practical phenomena of the world) (Bertalanffy 1976). For this class of systems, called open systems, the reductionism assumptions are simply impossible. This is because a very important characteristic of such systems is exactly the *interaction* with (other systems from) their environment. For example, a biological system having its connections with its environment closed will most probably die of 'starvation' (Bertalanffy 1976).

The descriptions above, and the common-sense presupposition that a team of designers operating in isolation will soon deplete/exhaust the available information and will fail in finding an optimum solution, a priori suggest that DEDS could be an open system. For this

reason, the DEDS will be judged in relation to the open system concepts in order to verify the above supposition.

The main sets of concepts that characterize an open system (see figure 2.3) are as follows (Bertalanffy 1976; Heylighen, Joslyn et al. 1993; Geyer 1994; Bennett, McRobb et al. 1999):

1. Boundary, input, output, throughput
2. Control, feedback, feed-forward, state of a system,
3. Hierarchies of systems.

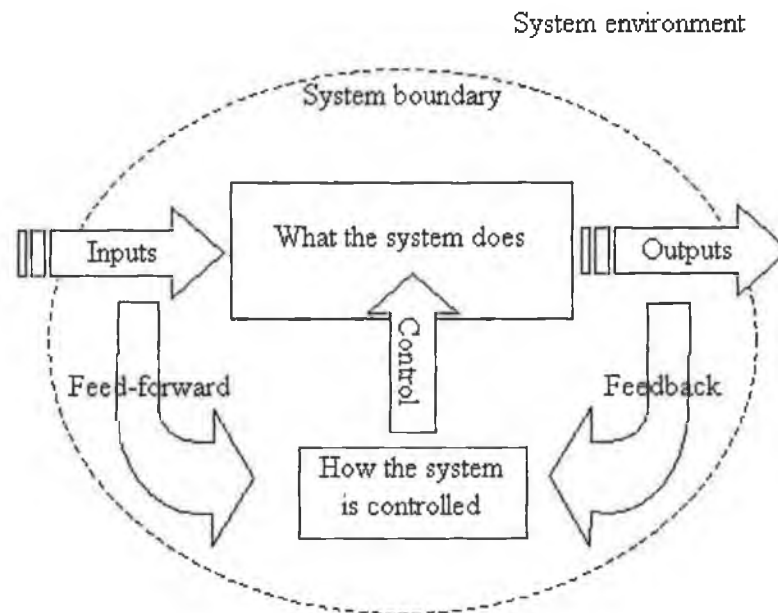


Figure 2.3 The main concepts of an open system (Bennett, McRobb et al. 1999)

1. *Boundary, input, output, throughput*

In order to discuss a system (e.g. structure, properties) it is necessary to distinguish the interior of the system from the exterior of the system. Generally, a *boundary* separates the system from its environment. Open systems interact with their environment. It is safe to say that, in fact, they interact with other open systems from the environment. This interaction has two components, i.e. the *input* and the *output*. The input defines what enters the system from outside (e.g. client requirements, business, market constraints), while the output defines what exits the system to outside (e.g. detailed information about a specific artefact to be manufactured). The output of a system generally is different from and

depends directly or indirectly on the system's input (e.g. the inputted raw information determines how the outputted information is structured and detailed). Usually, it is not possible to have an output without an input. The transformations that take place inside the system, from input to output are called *throughput*. Within the DEEDS, the throughput subsumes all the internal processes and activities required by the concurrent engineering design process (e.g. the engineering design itself, negotiations among engineers, CAD assisted drawings, etc).

2. *Control, feedback, feed-forward, state of a system*

An open system is also characterized by the set of advanced concepts, i.e. *control, feedback* and *feed-forward*. The *control* is the component in charge of the functionality of the system. It can take decisions depending on the current state of the system and is in command of the operation of the system as a whole (Heylighen, Joslyn et al. 1993) (Bennett, McRobb et al. 1999). The DEEDS needs a design management component to control the information transformation process and to implement decision-making and problem solving mechanisms. To improve the quality of the outputs, this design management component needs *feedback* not only from the market, but also from manufacturing, end of life department and users. *Feed-forward* information is also necessary in order to cope with changing market, supplying and manufacturing conditions.

3. *Hierarchies of systems*

As the open systems are usually complex systems, they can be split in a set of subsystems (each subsystem acting as a system). Depending on the purpose of the researcher, an open system generally contains a *hierarchy* of subsystems (Geyer 1994; Bennett, McRobb et al. 1999). Based on the findings concerning the distributed engineering design organization, the DEEDS can also be viewed as a hierarchy of systems, as follows:

- *Infrastructure System* - consists of the hardware and the software components and tools located within DEEDS;
- *Human System* - consists of all the designers and all other human actors involved in the distributed design process;
- *Engineering Design Model System* - defines, characterizes and informs the unfolding of the engineering design process itself, the methodologies, and methods used to support the transformation of information.

Each of the above mentioned subsystems could be further decomposed in *simpler* subsystems. For example, a particular team of designers or a set of designers with the same specialty is subsystem of the human subsystem.

In summary, in order to achieve its goals, a DEDS system must and needs to act as an open system.

(3) DEDS as a Cybernetic System

Cybernetic systems are generally defined by a set of specific characteristics that include the following (Joslyn 1992):

- *Complexity* refers to the fact that cybernetic systems consist of complex hierarchies of heterogeneous interacting components;
- *Mutuality* describes the nature of interactions among system components. Usually, the interactions occur in both real time and parallel and are mediated by cooperation processes.
- *Complementarity* arises from the complex relations among the heterogeneous system components that underlay information flows dependant on multiple structures and on which multiple structures depend. In such cases, any reductionism or any single dimension description is a priori incomplete. Any system description requires “multiple complementary, irreducible levels of analysis” (Joslyn 1992).
- *Evolvability* portrays the characteristic of cybernetic systems to grow and evolve based on the environment fluctuations and internal phenomena, rather than in a pre-programmed manner.
- *Constructivity* refers to the fact that cybernetic systems are increasing in size and complexity. This increase is based and depends on the previous system states and can occur in an indeterministic manner.
- *Reflexivity* is a consequence of the complex positive and negative feedback processes that take place within the system. The ultimate phases of these processes consist in “reflexive self-application” (Joslyn 1992) that generate phenomena such as self-reference, self-modelling, self-production, and self-reproduction.

The structural and functional analysis of a generic DEDS reveals that the DEDS system complies with the above characteristics, for a number of reasons as follows.

As already identified, structurally, a DEDES consists of human teams collaborating over computer networks with the goal of generating formal structured information that describes complex structures and behaviors (e.g. product specification). This activity is based on raw information given in the form of requirements and constraints. This raw information is particularly difficult to be synthesized as it is hidden in the users' or markets' informal language or tacit knowledge. Manual or automated (usually in the form of software applications) design models, methodologies and tools help in creating and structuring the information. Therefore, a DEDES can be perceived as a *complex* organization made of varied interacting components (e.g. functionally distributed humans, heterogeneous information sources and resources, diverse design methods, methodologies and tools).

The distributed design model (see section 2.2.3) reveals some interactions that take place during the cooperation process, as follows:

- Human-Human – necessary for the good functioning of a team of designers as well as critical for the coordination of different teams of designers;
- Human-Computer - subsumes what is known in literature as human computer interaction (HCI). It is also a key interaction in the process of the cooperation of distributed humans (i.e. human to computer to human interaction);
- Human-Information – deals with the perception of information by humans, especially the effect of information and the human response to the quantity and quality of the information;
- Computer-Information – mediates between the hardware and software architectures and the philosophy and logic for storing, retrieving and maintaining design data, information and knowledge.

These interactions are critical for reaching the design goal(s) and neither/no main component of the DEDES (e.g. infrastructure, human) can be analyzed without taking into consideration the effects that the other components have upon it. Therefore, a DEDES can be characterized by *mutuality* and *complementarity*.

Depending on the market needs and on past experience, a DEDES can be reorganized accordingly, so that the mistakes will not be committed again and the good things will be repeated. Advances in the ICT industry could also lead to the modernization of the system infrastructure with improved computers (e.g. faster processors, bigger memories). Moreover, the development of better software tools for design, collaboration and resources

management will more than surely have a strong impact on DEDS structure and functionality. Therefore, in order to attain a long-term viability, the author contends that a generic DEDS needs to *evolve*.

The present structure and functionality of a DEDS is the result of the evolution of the design process. In the past, a single designer managed the design process. In time, because of specific needs, more and more human participants became involved in the process. Moreover, in recent times, the design process gained important improvements in terms of complexity management, speed and quality from the ICT revolution (both hardware and software). Within a DEDS, the quantity and quality of human designers, infrastructure and especially information resources tends to increase in time with every application of the design process to existing data, information and knowledge. Therefore, *constructivity* is an important characteristic of a generic DEDS.

The *reflexivity* characteristic, while not yet formally acknowledged within the DEDS, once implemented could bring significant improvements in terms of data, information and knowledge such as self-generated and self-organized information resources, self-reproduction of successful work flows and self-simulation functionality for predictions. Hence, DEDS functionality can only be improved once the *reflexivity* characteristic is implemented (of course, the cost of implementing it should be considered). Anyway, pre-reflexive characteristics are identifiable in the form of so-called circular processes: self-organization, self-references and feedback cycles (Geyer 1994; Bennett, McRobb et al. 1999; Heylighen and Joslyn 2001).

2.3.3 A Distributed Engineering Design System Model

The DEDS model summarises the findings of the systemic approach. The intension is to adapt and restructure the distributed engineering design model (presented in section 2.2.3) to the proposed view, incorporating the found perspective of systems theory and cybernetics. Its purpose is not to mirror in detail the entire system, but to simplify it, concentrating on a specific key facet, i.e. information position within DEDS. The result is pictured in figure 2.4.

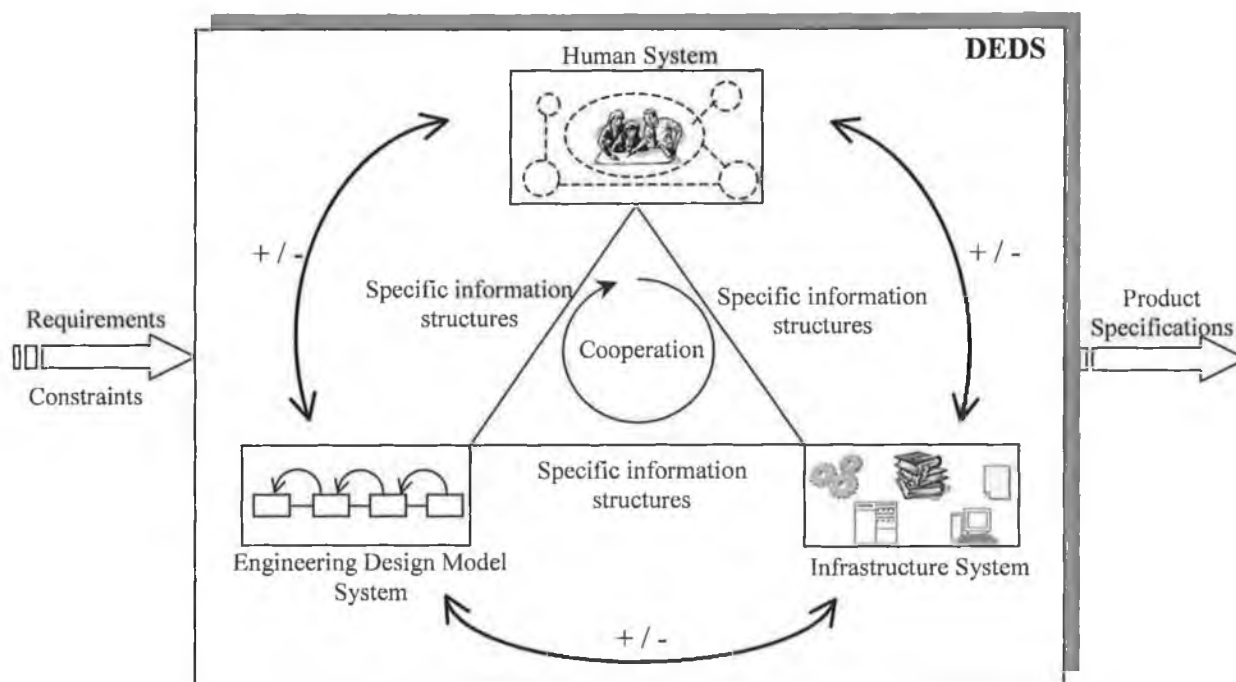


Figure 2.4 A high-level model of the DEDES

The DEDES is modelled as an organized collection of humans, machines and methodologies working together to transform information-based inputted requirements and constraints into appropriate product specifications information.

The adaptation of the previous distributed engineering design model includes the conversion of the *Human Component*, *Infrastructure Component* and *Engineering Design Model* into respectively the following DEDES subsystems (or systems):

- *Human System* – the collection of organizationally and hierarchically distributed multi-disciplinary engineering design teams working sequentially, concurrently or in parallel to design products. Its main characteristics include distribution (geographical, temporal, functional and semantical), concurrency and parallelism, which emphasize the need for a welding cooperation process to enable and support teamwork.
- *Infrastructure System* – the collection of manual and automated tools (e.g. drawing boards, computers with their applications, books, reports) acting as the medium or the workplace where the engineering design process takes place. The existence of this subsystem is due especially to the importance that the computers

and their applications have gained in today's design environments (see section 2.2.2).

- *Engineering Design Model System* – integrated life cycle design methods and methodologies for the development of products that guide and inform the process of design.

The author contends that the behaviours and structures of these three subsystems influence and condition each other during the overall progression of the engineering design activity, by means of feedback (i.e. the “-“ sign) and feed-forward (i.e. the “+” sign) processes (represented in the figure by the thick double arrowed lines). For example, the allocation and configuration of human resources and activities together with the type of tools to be used, depend on the phase of the engineering design process as expressed by the specific design model employed. In this example the *Engineering Design Model* feeds forward the Human and Infrastructure subsystems.

The performance of the engineering design process (i.e. the global behaviour) is conditioned by the interoperations among the Human, Infrastructure and Engineering Design Model subsystems. However, specific to the DEDES, these subsystems are heterogeneous in their substance: the Human System consists of humans, the Engineering Design Model System consists of cognitive and visual concepts which express methodologies and methods, and the Infrastructure System consists of manual tools, software applications, computers and other media formats. Nevertheless, the interrelationships and interdependencies among subsystems are possible and are expressed by means of exchanges of specific information structures using specific mediums. For example, an information exchange between Human and Infrastructure system may use a visual medium and may consist on mouse movements on the Human side and bits of data and graphical translations of geometrical formulas on the Infrastructure side. Thus, while the ‘information triangle’ may implicitly translate that the same kinds of information structures are exchanged, it actually only says that *information structures are exchanged*. Of course those information structures are specific, in terms of complexity, representation and broadcasting medium to each dialog that takes place. Besides its beneficial aspect (e.g. dialog is made possible), this situation results in a proliferation of irreducible representations of data, information and knowledge within the system. Furthermore, the relationships are enabled and supported by the cooperation processes, by means of components co-location, communication, coordination and collaboration. Therefore, the

cooperation process is viewed to hold a critical importance in the regulation of information flows and, consequently, the regulation of the whole DEDES.

In summary, the author contends that, from a high level point of view, the distributed engineering design activity is depicted as a complex of holistic inter and intra cooperation processes among mutually dependent structures (i.e. Human, Infrastructure and Engineering Design Model subsystems). The main material of the process is the information (as a common denominator of the different system components), which is distributed, assembled in vary degrees of complexity (i.e. data, information and knowledge) and heterogeneously represented within the system.

2.3.4 Information Setbacks in Distributed Engineering Design System

As shown in section 2.2.3, a series of both qualitative and quantitative information-related problems impinge on the effective operation of the cooperation process within a distributed design environment. This, in turn, may lead to misunderstandings, errors and poor product information structures. Therefore, local rather than 'global' design optima are more probable. What is even more 'unpleasing' is that the inferior qualitative information structures assembled in a mediocre cooperative environment will be used as informative means or even reused as building blocks in later design processes (after all, the DEDES does not disappear once a design process is finished, i.e. it has continuity in time). This historical propagation of local 'optima' will further obscure the desirable global 'optima' (of course, a good manufacturing, usage and market feedbacks can hinder the above course of events). Nevertheless, the reevaluation of the required quality and quantity of information may prove to be too costly in terms of time and human resources allocated. No wonder that, as the majority of human knowledge gains, the distributed engineering design may be viewed as a trying and error process.

By applying the systemic reassessment of information related problems, the author identifies that, as expressed in section 2.2.3, problems are human-centered. Designers need to browse and search through the diverse, dispersed and huge amount of information resources. Before being able to use information structures, they also have to interpret and understand their meaning. Furthermore, designers have to share among each other not only syntactics, but also semantics. Therefore, human processing or team processing performance is the measure of the information performance of the entire DEDES.

From the system perspective, this local state of affairs already creates difficulties for the whole system. If the human information processing capacity critically influences the quantity and quality of information that flows within the system, it means that no matter how advanced the design tools, methods and methodologies are, once this (i.e. human) processing limit is reached all the other system components can only do is to maximize their quantitative features (which is an unacceptable under-use of their potential capabilities).

Therefore, a designer's limit to access, process and distribute information structures propagates throughout the system and becomes a system's (i.e. DEDS's) limit to access, process and distribute information. This state of affairs subjectively confines the other system components' development.

Hence, where possible, a redistribution/decentralization of information processing from the human subsystem components to different DEDS components is critical. In this case, the human component should have more of a feedback and feed-forward role, than a central role in managing information structures. By doing this, the human designers can focus on what they are much more superior than any other component, i.e. imaginative thinking.

In summary, an efficient functioning of the system can be reached by maximizing the quality and quantity of information, which in turn can be achieved by distributing the control of different information aspects (e.g. processing, storing, retrieving, searching, validating, producing, consuming) among system components that are best prepared for the specific role. In order to achieve this desiderate, the first step is to enable the access of the different system components, especially machines, to information. In other words, it is essential for the information to be not only human readable as it usually is (90% of information is strictly human enabled) but also machine-readable.

2.4 Distributed Engineering Design System Requirements

A series of high-level preliminary requirements for a better functionality of the DEDS concludes the systemic approach carried out. Their purpose is to deduce a minimal set of necessary conditions under which the behaviour of the system can be controlled and can be kept between certain feasibility parameters. Moreover, the requirements should identify the features that enhance DEDS functionality and reduce its negative aspects.

Accordingly with the discussed issues, this thesis proposes the following requirements for a solution system:

- The design information structures should be organized so as to allow sharing, reuse and maintenance of meaning along the various human and non-human DEDES actors. This requirement necessitates that the information resources be map-able (for translations both at the system and subsystem level), domain independent (in order to support distribution and growth) and machine-readable (for inter-subsystem interfaces).
- The DEDES cooperation process should consist of a hierarchy of cooperation processes as follows: a cooperation process for each subsystem and a cooperation process for each inter-system interface. In this way, the implementation of cooperation at the subsystem level can take advantages of the positive particularities of the concerned subsystem (for example communication between humans should not be restricted by communication constraints specific to software applications).
- The translations of information resources between DEDES and its environment and inside DEDES should be as hidden as possible from the human user. This requirement deals with non-technological issues, such as user acceptance of new technologies and working environments.
- Subsystems interfaces should require minimal human intervention and should have an advisor and informative types of roles. In this way, the designers can spend more of their time designing and not undertaking secondary activities.
- The interfaces between the human subsystem and the other subsystems should be autonomous (background work is performed without the awareness of the designer), proactive (when possible information is brought to the designer when the need arises and before he/she requested it) and should encourage learning and support creativity.

In summary, the identified requirements convey the need for an *efficient* and *useful cognitive mean to model* the distributed engineering design domain. The *efficient* and *useful* terms translate to a minimal resource investment and minimal impact on the organization structure (i.e. the implementation does not require massive business reorganizations), while the quality of the added value is significant. *Cognitive* refers to the necessity for the system to be semantically enabled and therefore to be able to deal with whatever complex information structures (i.e. data, information and knowledge). The solution system should *model* or represent the engineering design domain, which means a

deliberate usage of structures of signs able to facilitate research, classifications, and consequently knowing/knowledge.

The efficient, useful and semantically enabled *mean*³ for achieving the requisite solution system wraps up the core of the identified requirements and consists of the following:

- Reusable, shared and formal structures of information kept together by the intrinsic logic of the engineering design domain.
- Integrative mechanisms capable of translations and mappings between different contexts (e.g. Human, Infrastructure, Engineering Design Model, other systems from the outside environment).
- Control mechanisms able to regulate the functionality of the system.

The advances in the Artificial Intelligence field, particularly in Distributed Artificial Intelligence provide the technologies for developing this necessary *mean*.

The research associated with this thesis has identified *ontologies* as the solution for organizing the system's information resources. The study of ontologies has developed gradually from specific needs associated with the problem of knowledge management within a computational environment and particularly from the problem of knowledge sharing and reuse. Ontologies specify content specific agreements to facilitate knowledge sharing and reuse among systems that submit to the same ontology/ontologies by the means of ontological commitments (Spyns, Meersman et al. 2002). They describe concepts and relations assumed to be always true independent from a particular domain by a community of humans and/or machines that commit to that view of the world (Neches, Fikes et al. 1991; Gruber 1993; Guarino 1997).

Coupled with ontologies, *agent-based systems* are able to provide that autonomous, proactive and cooperative hard working helper that can both integrate disparate components and regulate the behaviour of DEDS. Considered an important new direction in software engineering (Jennings 2000; Wooldridge and Ciancarini 2001), agents and multi-agent systems (MAS) represent techniques to manage the complexity inherent in software systems and are appropriate for domains in which data, control, expertise and/or resources are inherently distributed (Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999).

³ Mean has the sense from the "workers are the *means* of production".

2.5 Conclusions

This chapter presented a literature-based characterization of the engineering design process and of the distributed engineering design organization. Information related deficiencies of current state of affairs have been identified and acknowledged.

The systemic approach to the distributed design organization (see figure 2.5) has been identified as a viable research alternative.

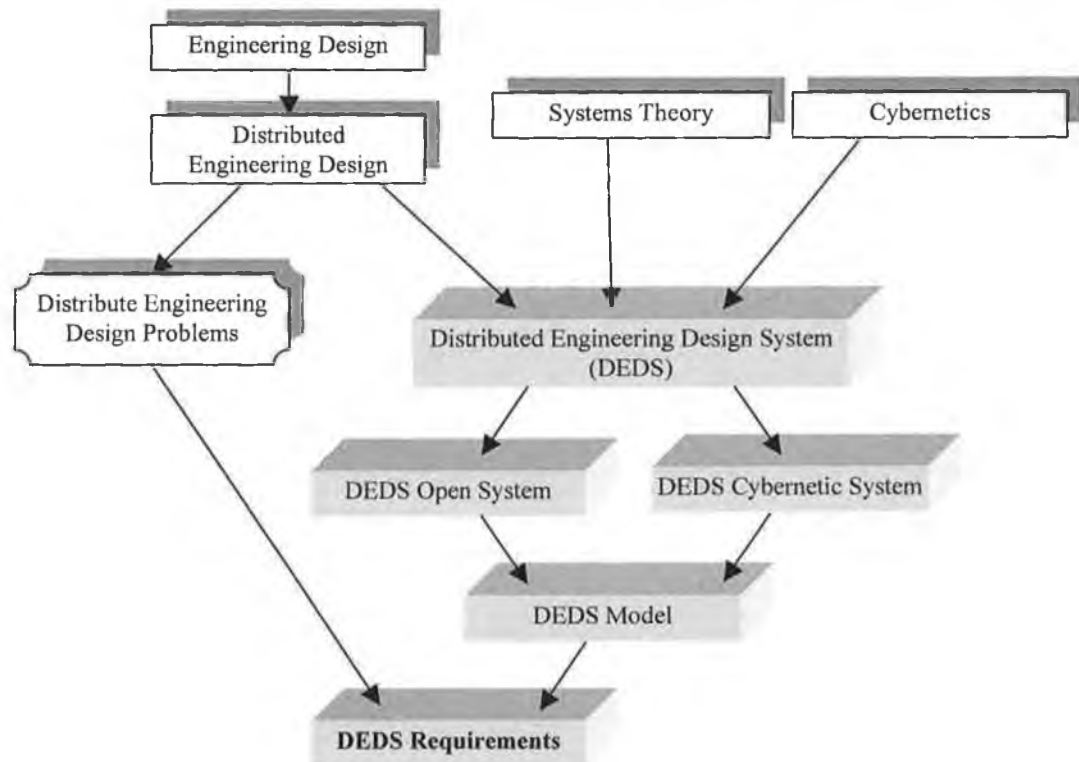


Figure 2.5 A systemic approach to distributed engineering design.

Once the key concepts of the proposed approach were defined, a characterization of the DEDS has been depicted. Furthermore, a systemic (structural and functional) reassessment of the DEDS has been performed. The cooperation process has been seen as being a critical component for the DEDS functionality. The analysis also stressed cooperation's reliance on data, information and knowledge (as they form its working material).

Based on the existing problems of distributed engineering design interpreted from a systemic perspective, a set of DEDS requirements was identified. Next the systemic analysis of the DEDS concluded with what the author considers to be the problem statement of this thesis, as follows:

- Within the distributed engineering design organization, negative issues exist which cannot be disregarded. Moreover, such issues will become more acute in time.
- The proposed solution is to identify an efficient and useful cognitive mean to model the distributed engineering design domain.

Further investigations identified two suitable technologies for supporting that mean, i.e. ontologies and agent-based systems. Chapter Three will explore in depth the key issues regarding these two technologies.

Chapter 3

Distributed Technologies

3.1 Introduction

3.2 Ontologies

3.3 Software Agents

3.4 Conclusions

3.1 Introduction

This chapter introduces two of the most promising technologies for implementing semantic enabled environments, i.e. ontologies and software agents (Gruber 1992; Genesereth and Ketchpel 1994; Nwana 1996; Gomez-Perez 1998; Guarino 1998; Jennings, Sycara et al. 1998; Wooldridge 1999; Hendler, Berners-Lee et al. 2002). These two technologies are envisioned to form the next distributed computational environment, capable of managing inherent complex and inherent distributed systems. Various kinds of software agents will *act* in a *semantically* enabled (by the means of ontologies) environment. The relationship between ontologies and software agents is mutual benefic. While both of them can function independently their true performance is achieved by the means of each other. The software agents will benefit of the shareable and machine enabled pool of knowledge, and the ontologies will reach their full potential when exploited by software agents.

The first part of the chapter portrays what is advocated to be the enabler of human-machine integration and knowledge sharing and reuse at a worldwide scale, i.e. ontologies. The various definitions proposed in the literature are introduced and the most important ones are also analyzed. The research in ontologies typologies and methodologies for engineering ontologies are presented and discussed to further deepen the understanding.

The second part of the chapter presents the state of the art literature review of software agents. The main characteristics of a software agent are discussed and analyzed. Following the presentation of some of the most cited agent typologies and agent architectures, the multi-agent system theory is introduced.

The chapter ends with a final set of conclusions regarding these two distributed technologies.

3.2 Ontologies

The term *ontology* emerged outside philosophy as a fancy denotation of some results of conceptual analysis and domain modeling (Guarino 1998) with an *implicit* understanding of its meaning among researchers. This means that when a researcher or a group of researchers would employ ontology-based *meanings* they would depict their own understanding of the concept in their particular field for their particular needs. This is because while the term “ontology” has an unquestionable defined meaning in the science of philosophy, when *imported* in other domains it loses some of its characteristics and gains others (specific to the borrowing domain) without any of these phenomena being

explicitly explained and defined by the *borrowers*. These circumstances led to an inflation of interpretations of the concept of ontology when used outside philosophy. However, as the research in knowledge-based systems progressed, a shift of focus brought the term to the attention of the researchers. An explicit understanding, as well as the formal study on the methodological side has become necessary. Up until now, the results in formal ontology research appear to be encouraging and give optimism not only within the Artificial Intelligence (AI) field, but also to all researchers dealing with computational environments in which explicitly represented knowledge serves as a communication medium among people and machines.

3.2.1 Background

Ontologies have appeared from a need emerged within AI: sharing and reuse of knowledge. Knowledge bases form the foundation of AI. They are bodies of information used to formalize a universe of discourse by describing facts and assertions assumed to be always true within a particular domain (Guarino 1997). Any knowledge base contains *background information* about the specific domain to which it is applied (Gruber 1991; Neches, Fikes et al. 1991; Gruber 1995; Guarino 1998). In other words they (i.e. knowledge bases) are capturing the knowledge within the domain of interest so that specific software applications can be developed (using neural networks, genetic algorithms and other AI specific techniques) to take that *background knowledge* as an input (Gruber 1995) and process it. Therefore, the software programs are able to process not only data but also knowledge. This has opened new perspectives concerning the scope, the role and the power of the computational machines. In this way it is possible to build large and powerful AI systems.

However, one of the main limitations of the knowledge bases is exactly their submissiveness to the domain they are representing and formalizing. This is because representing new knowledge (from adjacent or separate domains) and integrating it to the already built system generally requires building the system from scratch (Neches, Fikes et al. 1991). In this way, any knowledge engineering process becomes time consuming and very expensive (Neches, Fikes et al. 1991; Gruber 1993). For the same reasons, once a knowledge base system has been built, its maintenance and testing is a cumbersome task (Gruber 1993). Therefore, finding ways of preserving, sharing and reusing the existing knowledge bases across the different domains of the human endeavor has concentrated a good deal of effort from Artificial Intelligence researchers (Gruber 1991; Neches, Fikes et al. 1991; Gruber 1992; Guarino 1995; Fensel 2000). Neches et al have identified four

critical impediments to knowledge base share and reuse, which are (Neches, Fikes et al. 1991):

1. *Heterogeneous Representations*. Across the knowledge representation field different communities use different formalisms to represent knowledge. Hence, a direct exchange and reuse of knowledge among different formalized knowledge-based systems is syntactically and semantically impossible.
2. *Dialects within Language Families*. A formal knowledge representation language is required to represent knowledge. Even if the languages within a language family (that are submitting to the same formalism) are sharing a *core* philosophy and terminology, there are still too many “arbitrary and inconsequential differences in syntax and semantic” (Neches, Fikes et al. 1991) between different dialects for a *natural* translation to be realistic.
3. *Lack of Communication Conventions*. Knowledge-based systems lack on agreed-upon standard protocols to facilitate knowledge exchange among them, or between them and other software systems.
4. *Model Mismatches at the Knowledge Level*. Even if the above impediments (1, 2, and 3) are to be resolved, an effective communication or knowledge exchange would be quite difficult in the absence of a “shared vocabulary and domain terminology” (Neches, Fikes et al. 1991).

Ontologies have been proposed to overcome the difficulties raised by “*monolithic, isolated knowledge systems*” (Gruber 1991), by specifying content specific agreements to facilitate knowledge sharing and reuse among systems that submit to the same ontology/ontologies by the means of ontological commitments (Gruber 1995; Spyns, Meersman et al. 2002). Hence, while the knowledge bases are characterized by “*high internal coupling*” (Gruber 1991) – that is the implicit assumptions made regarding facts, procedures, terminology, and axioms of the specific domain – ontologies provide a way of building “external coupling interfaces that would enable the developer to reuse software tools and knowledge bases as modular components“(Gruber 1991).

From a functional point of view an ontology is seen as an equivalent of a database schema. In general, a data model (e.g. database schema) “represents the structure and the integrity of the data elements of the, in principle ‘single’, specific enterprise application(s) by which it will be used” (Spyns, Meersman et al. 2002). Hence, a data model usually implements some kind of informal agreement between the developers and the users of that specific data model regarding the semantics of the data (i.e. the specific needs the application has to

fulfill). But this agreement starts and end with the above mentioned developers and users and it is not intended for sharing with other communities. An ontology, such as any database schema is a partial account of a conceptualization (Guarino and Giaretta 1995; Spyns, Meersman et al. 2002), so they both have the same functions (e.g. establishing agreements, albeit in varying degrees). The difference is the domain they cover: database schemas are task-specific and implementation oriented (Spyns, Meersman et al. 2002), while ontologies are as generic and as task-independent as possible. These result in some key differences between ontologies and databases, as follows (Fensel 2000):

- “A language for defining ontologies is syntactically and semantically richer than common approaches for databases.”
- “The information that is described by an ontology consists of semi-structured natural language text and not tabular information.”
- “An ontology must be shared and consensual terminology because it is used for information sharing and exchange.”
- “An ontology provides domain theory and not the structure of a data container.”

Therefore, an ontology lies somewhere between a knowledge base and a database schema. Because of the encouraging results as well as the potential positive outcomes, ontologies have widen beyond the boundaries of AI, in domains such as Database Theory and Computational Linguistics (Guarino 1998). They (i.e. ontologies) are currently very popular mainly within fields that require a knowledge-intensive approach to their methodologies and system development, such as knowledge engineering (Gruber 1993; Uschold and Gruninger 1996; Gaines 1997; Gomez-Perez 1998), knowledge representation (Artala, Franconi et al. 1996; Guarino 1998), qualitative modeling, language engineering, database design (Van de Riet 1998), information modeling (Weber 1997), information integration (Bergamaschi, Castano et al. 1998; Guarino 1998; Mena 1998), knowledge management and organization and agent-based design (Nwana 1996; Odell 2000; Chaib-draa and Dignum 2002).

3.2.2 Definition

Because of some particular terminological and semantical issues raised by the concept of ontology, a special attention has to be given to its interpretation in differentiae to concepts and terms with which it came in contact. This means that the concept of ontology has to be clearly delimited from the concepts and situations for which it was used as a synonym, i.e. knowledge base, ontology as a philosophical term, knowledge sharing and reuse.

The term of *ontology* has been borrowed from Philosophy where it is defined as a “branch of metaphysics concerned with identifying, in the most general terms, the kinds of things that actually exist. Thus, the “*ontological commitments*” of a philosophical position include both its explicit assertions and its implicit presuppositions about the existence of entities, substances, or beings of particular kinds” (Kemerling 2002).

This meaning was particularly useful in the first stages of its usage when the term was pointed to some agreed-on formalism and conventions at a general level that should “provide software interfaces to knowledge representation systems” (Neches, Fikes et al. 1991). As this line of research deepened it became clear that the AI interpretations of an ontology differs from the philosophical understanding. While for a philosopher the ontology is a “particular system of categories accounting for a certain vision of the world” (Guarino 1998), independent on a particular language, for the AI researcher an ontology refers to a “particular *artifact* constituted by a specific *vocabulary*” (Guarino 1998) that describes a certain domain by explicitly constraining the intended meaning of the vocabulary words. Usually the constraints are implemented in respect to the First Order Logic form and the vocabulary words are unary (concepts) or binary (relations) predicate names (Gruber 1995). Therefore, a commitment to a certain language should be assumed in order to develop an ontology. From this point forward, in order to distinguish between the philosophical sense and the AI sense, Guarino’s terminological distinction (Guarino 1998) will be adopted:

- Philosophy: as a language independent system of categories **the Ontology** is a **conceptualization**;
- AI: **an ontology** as a language dependent formal artifact;

The role of ontologies is to represent knowledge in such a way so as to make possible the communication between different machines and between machines and humans to a knowledge level contrasting to communication at data level. The research in knowledge representation and knowledge engineering, although successful, failed to provide by themselves cost-effective and time-effective shareable knowledge, so that different knowledge-based systems to be able to communicate between each other (Neches, Fikes et al. 1991; Gruber 1993; Gruber 1995; Guarino, Borgo et al. 1997). A solution to this deadlock is to use ontologies for describing concepts and relations assumed to be always true independent from a particular domain by a community of humans and/or agents that commit to that view of the world (Guarino 1997). In this way ontologies may be viewed as knowledge bases integration mechanisms with the condition of an agreed-upon vocabulary used. Therefore, while an ontology is a kind of a *shallow* knowledge base from the specific

domain integration point of view, a generic knowledge base may also contain background information explicitly and implicitly within its structure, describing a particular instantiation or state of affairs (Guarino 1997).

Neches et al have proposed one of the early definitions of an ontology. It states that “an ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary” (Neches, Fikes et al. 1991). This definition of ontologies has been proposed within an effort to envision a computational environment where knowledge-based systems, AI tools and conventional software will interact at a knowledge level between each other by the means of shared ontologies. As any knowledge base incorporates explicitly or implicitly an ontology, a shared ontology will *declaratively* specify the “*ground rules*” (Neches, Fikes et al. 1991) for modeling a domain in the form of top-level interconnected abstraction. In this way, an ontological commitment is viewed as a semantical assurance for providing specific services to systems (e.g. humans, agents, conventional software) that adopt a specific ontology or a specific library of ontologies.

Gruber (Gruber 1993) proposed another definition of ontology, by establishing its relationship with the concept of formal knowledge. A body of formally represented domain knowledge generally consists of objects and relationships between objects (i.e. universe of discourse) based on the *conceptualization* of that domain (Genesereth and Nilsson 1987; Gruber 1993). A conceptualization is viewed as an “abstract, simplified view of the world” (Gruber 1993) to be formally represented. Following those clarifications, Gruber states that “an ontology is an explicit specification of a conceptualization” (Gruber 1993).

A *vocabulary* is also needed to explicitly represent the universe of discourse. The main advantage of Gruber’s definition is that it requires the ontology to be *explicit* i.e. to be publicly available, not implicitly incorporated in some knowledge base. In this way an application no longer needs to have background knowledge about the specific domain in order to have access to its inputs. It will be enough to commit to the ontology that describes that domain. Therefore it is possible to separate the symbol level of an application (where the internal algorithms are represented) from its knowledge level (where the communication protocols are defined).

Gruber’s definition is based on the assumption that every system that incorporates formally represented knowledge is explicitly or implicitly committed to a conceptualization. For this reason, while Gruber’s definition is one of the most widely used definitions of ontology, it

still needs further clarifications of the terms it uses, especially the distinction between ontology and conceptualization.

Guarino and Giaretta distinguished between diverse interpretations of the term ontology have among different researchers. They discovered seven angles of understanding that can be classified in three classes (Guarino and Giaretta 1995) (see Figure 3.1).

In order to avoid possible confusions Guarino and Giaretta suggested a terminological clarification of the three possible classes they identified as follows (Guarino and Giaretta 1995):

1. To use “Ontology” with capital “o” as the term to identify the philosophical discipline.
2. To use the term “conceptualization” to identify a conceptual semantic entity.
3. To use the term “ontological theory” to identify a specific syntactic object intended to represent knowledge.

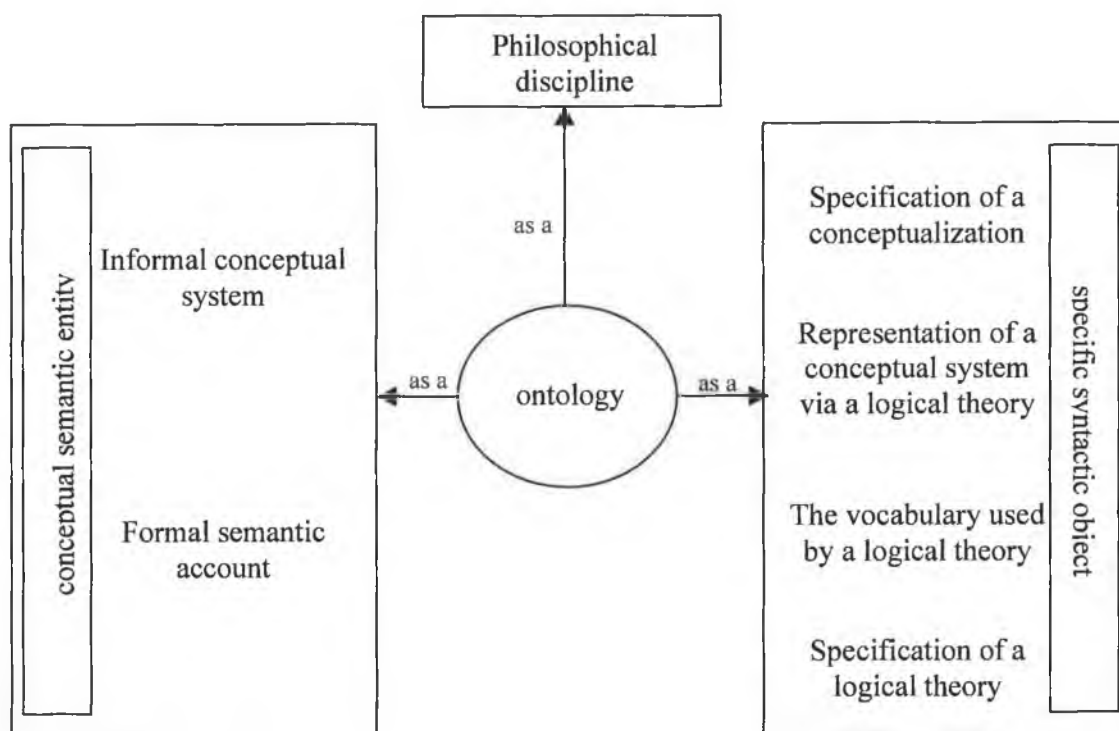


Figure 3.1 Possible interpretations of the term “ontology” after (Guarino and Giaretta 1995)

Therefore, while ontological theories are a special kind of artefact, conceptualisations are their semantical counterpart, with the specification that “the same ontological theory may

commit to different conceptualizations, as well as the same conceptualization may underline different ontological theories” (Guarino and Giaretta 1995).

Because of its wide use, special attention has been given to Gruber’s definition of ontology from all other interpretations. While the term “explicit” seems not to raise any doubts when explained as a “concrete, symbol level object” (Guarino, Carrara et al. 1994), not the same thing can be said about the term conceptualization. Guarino et al argue that while the meaning of the term conceptualization appears to be understood as “a set of extensional relations describing a particular state of affairs”, the actual meaning it has is “an intensional one [...] something like a conceptual grid which we superimpose to various possible states of affairs” (Guarino, Carrara et al. 1994; Guarino and Giaretta 1995; Guarino 1998). In other words, while an extensional interpretation states that different snapshots of a universe of discourse represent different conceptualizations, the intensional interpretation affirms that they are different states of affairs of the same conceptualization. With all the above denotations clarified, “given a language **L** with ontological commitment **K**, an ontology for **L** is a set of axioms designed in a way such that the set of its models approximates as best as possible the set of intended models of **L** according to **K**” (see Figure 3.2) (Guarino 1998).

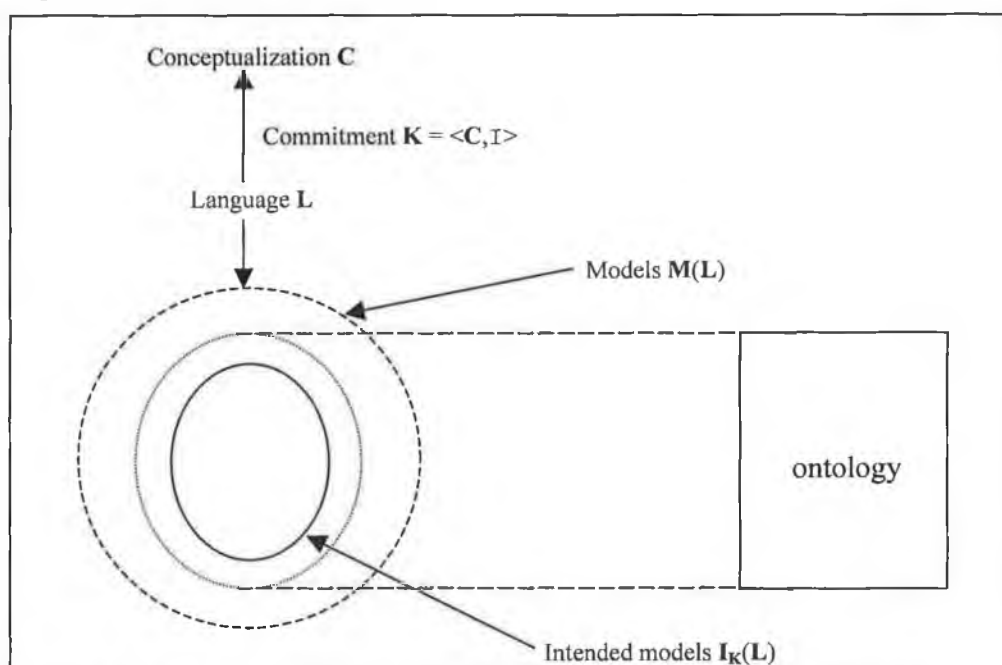


Figure 3.2 Conceptualization-language-ontology relation (Guarino 1998).

Following all these terminological clarifications Guarino proposes a refined definition of an ontology, by making clear the difference between an ontology and a conceptualization: “An ontology is a logical theory accounting for the intended meaning of a formal

vocabulary, i.e. its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. An ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models.” (Guarino 1998)

Borst et al have given an elaboration of Gruber’s definition, as follows: “Ontologies are defined as *formal* specification of a *shared conceptualization*.” (Borst, Akkermans et al. 1997)

Generally a merge of both, Gruber’s and Borst’s et al, definitions is used in literature: “Ontologies are *explicit formal* specification of a *shared conceptualization*” (Studer, Benjamins et al. 1998). Studer et al have explained the terms as follows (Studer, Benjamins et al. 1998):

- “explicit” – “the type of concepts used, and the constraints on their use are explicitly defined”
- “formal” – “the ontology should be machine readable, which excludes natural language”
- “shared” – “reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group”
- “conceptualization” - “abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon”

A pragmatic alternative, based on their experience in building ontologies, for defining an ontology, has been given by Noy and McGuinness as follows: “an ontology is a formal explicit description of concepts in a domain of discourse (*classes* (sometimes called *concepts*)), properties of each concept describing various features and attributes of the concept (*slots* (sometimes called *roles* or *properties*)), and restriction on slots (*facets* (sometimes called *role restrictions*))”. (Noy and McGuinness 2001)

Uschold adopts a broader and more informal point of view when he proposes a working definition of an ontology: “An ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretations of terms. An ontology is

virtually always the manifestation of a shared understanding of a domain that is agreed between a number of agents.” (Uschold 1998)

Recently, Fikes and Farquhar have been given the following definition: “We consider ontologies to be domain theories that specify a domain-specific vocabulary of entities, classes, properties, predicates, and functions, and a set of relationships that necessarily hold among those vocabulary items.” (Fikes 1999)

Sowa’s definition takes a philosophical perspective: “The subject of *ontology* is the study of the *categories* of things that exist or may exist in some domain. The product of such a study, called *an ontology*, is a catalogue of the types of things that are assumed to exist in a domain of interest D from the perspective of a person who uses a language L for the purpose of talking about D.” (Sowa 2000)

The common understanding of all the definitions and interpretations of an ontology orbit around two main characteristics: formality and consensus. All of the definitions stress the importance of representing the knowledge from an ontology in a consensual manner, at least among a specified group for knowledge sharing to be possible and implementable. Not the same thing can be said about *formality* requirement. Uschold allows ontologies to be expressed in a restricted and structured form of natural language, while Gruber’s line of definition enforces a well-defined logical model for ontologies. Nevertheless, the general vision is that ontologies should be machine-enabled and, if not directly human-readable, they should at least contain plain text notices or explanations of concepts and relations for the human user (Borst, Akkermans et al. 1997; Guarino 1998; Studer, Benjamins et al. 1998; Uschold 1998; Fikes 1999; Sowa 2000; Noy and McGuinness 2001).

A simple example of ontology is the *Publication-ontology* (found on Ontolingua Server) that defines 13 classes and 28 relations (see table 3.1) (Benjamins, Fensel et al. 1998):

Concepts – Class hierarchy	Relations
On-Line-Publication	Abstract, Book-Editor, Conference-Proceedings-Title, Contains-Article-In-Book, Contains-Article-In-Journal, Describes-Project, First-Page, Has-Author, Has-Publisher, In-Book, In-Conference, In-Journal, In-Organization, In-Workshop, Journal-Editor, Journal-Number, Journal-Publisher, Journal-Year, Last-Page, On-Line-Version, On-Line-Version-Of, Publication-Title, Publication-Year, Technical-Report-Number, Technical-Report-Series, Type, Volume, Workshop-Proceedings-Title
Publication	
Article	
Article-In-Book	
Conference-Paper	
Journal-Article	
Technical-Report	
Workshop-Paper	
Book	
Journal	
IEEE-Expert	
IJHCS	
Special-Issue	

Table 3.1 Example of an ontology from (Benjamins, Fensel et al. 1998)

3.2.3 Typologies

Guarino proposes a classification of ontologies under three headings, as follows (Guarino 1997):

1. By the level of detail
 - a. *reference* (off-line) ontologies
 - b. *shareable* (on-line) ontologies
2. By the level of dependence of a particular task or point of view
 - a. *top-level* ontologies
 - b. *domain* ontologies
 - c. *task* ontologies
 - d. *application* ontologies
3. Representation ontologies

There are two types of ontologies depending on the *level of detail* planned for building ontologies, i.e. *reference ontologies* and *shareable ontologies*. An ontology approximates the intended models of a logical language. (Guarino 1997; Guarino 1998) There is no formula to calculate an *optimal distance* between the intended models of a logical language (according to an ontological commitment) and the underlined ontology. This distance depends on the practical needs that ontology should fulfill. Nonetheless, there are tradeoffs between a detailed approach and a coarse approach to designing ontologies. While a fine-grained (reference) ontology will specify more precisely the intended meaning of a vocabulary (Guarino 1998) (and therefore can be used off-line for reference purposes), it would be difficult to be assembled and reasoned on it (Guarino 1998). On the other hand, a coarse (shareable) ontology would be much easier shared among its clients that “already agree on the underlying conceptualization” (Guarino 1998), and therefore it can be used on-line to support the system’s services (Guarino 1997; Guarino 1998).

The need for classifying ontologies by the *level of dependence* was raised by a major application area of ontologies, i.e. information integration. (Guarino 1997; Weber 1997) In other words, in what conditions can two systems communicate at a knowledge level, and how such a communication can be enabled.

To solve this kind of difficulties, Guarino proposes a bottom-up approach to developing “different kinds of ontology according to their level of generality” (Guarino 1998). Hence,

depending on their *level of dependence* on a particular task or point of view, there are four types of ontologies (see Figure 3.3) (Guarino 1997; Guarino 1998). The *top-level ontologies* specify very general concepts, “which are independent of a particular problem or domain” (Guarino 1997; Guarino 1998)(e.g. engineering, person, agent); *domain ontologies* and *task ontologies* specialise these concepts (of *top-level ontologies*), referring to, respectively, a generic domain (e.g. mechanical-engineering, software-engineering, car disposal) or to a generic task or activity (Guarino 1997; Guarino 1998) (e.g. requirements analysis, disassembly); at *application ontologies*’ level further specialization is involved by describing concepts “depending on a particular domain or task” and are often “*roles*” of domain or task entities performed during a certain activity (Guarino 1997; Guarino 1998) (e.g. disassembly time).

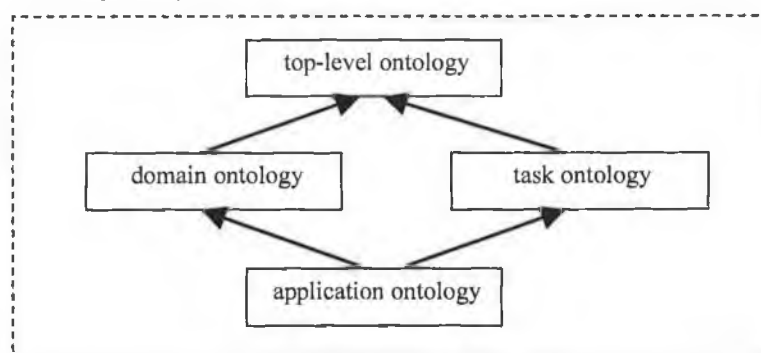


Figure 3.3 Kinds of ontologies, according to their level of dependence on a particular task or point of view. Thick arrows represent specialization relationships (Guarino 1997; Guarino 1998).

Representation ontologies stand for a special kind of meta-level ontologies “describing a classification of the primitives used by a knowledge representation language (like concepts, attributes, relations)” (Guarino 1997). An example of a representation ontology is the Frame Ontology (Gruber 1993) introduced within the Ontolingua system for capturing “common knowledge-organization convention” (Gruber 1993) with the purpose of enabling translations among different knowledge representation languages.

While agreeing that any ontology development, after all, depends on particular circumstances and needs (so any typology should depend on the practical use of ontologies), Uschold sets out three main dimensions by which ontologies may be classified, as follows (Uschold 1996):

1. Formality: “the degree of formality by which a vocabulary is created and meaning specified”

2. Purpose: “the intended use of the ontology”
3. Subject Matter: “the nature of the subject matter that the ontology is characterizing”

While some authors’ request an ontology language to be formal, Uschold adopts a weak position regarding the formality requirement. Hence, along the *formality dimension*, he identifies four kinds of ontologies stretching from ontologies with no formality requirement at all to ontologies articulated in a meticulous formal language, as follows:

- a. ontologies “expressed in loosely natural language” (Uschold 1996) in the case of *highly informal* ontologies;
- b. adding some degree of structure to the natural language results in *structure informal* ontologies that are “expressed in a restricted and structured form of natural language, greatly increasing clarity by reducing ambiguity” (Uschold 1996);
- c. an artificial language has to be developed for expressing *semi-formal* ontologies;
- d. *rigorously formal* ontologies are expressed by “meticulously defined terms with formal semantics, theorems and proofs of such properties as soundness and completeness” (Uschold 1996).

The *purpose dimension* deals with the intended use an ontology may have. Three main application areas have been identified, as shown in figure (see Figure 3.4).

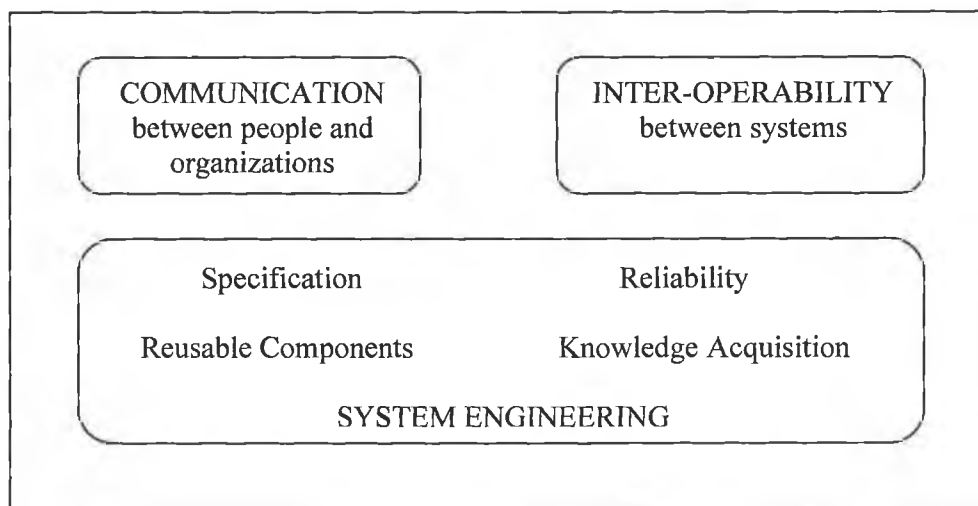


Figure 3.4 Uses for Ontologies (Uschold 1996)

The main categories of the *purpose dimension* (i.e. Communication, Inter-operability, System Engineering) can be further granulated into categories and subcategories (e.g. within communication between people one may want to specify who the intended users are). (Uschold 1996) A special kind of category related to *purpose dimension* is *genericity*,

which “is the extent to which an ontology can or is intended to be reused in a range of different situations” (Uschold 1996). These sorts of ontologies cover efforts that range from organizing human knowledge (in the case of upper-level ontologies) to particular knowledge systems for specific applications (in the case of application ontologies). Noy and Hafner have identified four classes of ontologies relative to their genericity dimension, as follows: natural language applications, theoretical investigations, knowledge sharing and reuse, simulation and modelling ontologies (Noy and Hafner 1997).

The *subject matter dimension* covers all the topics an ontology may represent, that is anything conceivable, under the following headings (Uschold 1996) :

- *the subjects area like medicine, engineering design, geography*
- *the subject matter of problem solving*
- *the subject matter of knowledge representation languages*

Ontologies concerned with different areas of science or so (e.g. medicine, engineering design, geography) are usually called *domain* ontologies; ontologies involved in problem solving are called *task, method* or *problem solving* ontologies; ontologies used for knowledge representation languages are denoted *representation ontologies* or *meta-ontologies* (Uschold 1996).

3.2.4 Methodologies for Building Ontologies

WordNet defines the term *methodology* as follows (Miller, Fellbaum et al.):

1. “the branch of philosophy that analyzes the principles and procedures of inquiry in a particular discipline“;
2. “the system of methods followed in a particular discipline”.

A *method* is defined as “a way of doing something”, especially a “systematic one; implies an orderly logical arrangement (usually in steps) “. Hence, from the point of view of this thesis, a methodology is a systematic approach to conducting an engineering project that suggests the activities to be performed at certain stages of the ontology development process.

Before illustrating the main attempts to formulate a methodology for building ontologies a set of terms used in ontology design are introduced, as follows:

- *Taxonomy* consists of a set of terms that alongside their definitions and relations among them form an ontology. Sometimes a taxonomy is viewed as a simple

ontology. (Gruninger and Fox 1995; Guarino 1998; Fernandez-Lopez, Gomez-Perez et al. 1999; Gomez-Perez 1999; Noy and McGuinness 2001)

- *Axioms* are formal sentences that are always true (Guarino 1998; Gomez-Perez 1999).
- *Concepts / Classes* are general, abstract or concrete notions within a domain of discourse. An ontology is formally describing a domain by describing its concepts. (Uschold and King 1995; Gomez-Perez 1999; Noy and McGuinness 2001)
- *Relations* represent “a type of interaction between concepts of the domain” (Gomez-Perez 1999) (e.g. subclass-of connected to).
- *Slots / Roles / Properties* represent the various features and attributes of a concept. (Noy and McGuinness 2001)
- *Facets* describe restrictions on slots. (Noy and McGuinness 2001)
- *Instances* represent elements (Gomez-Perez 1999).

In his attempt to enable knowledge sharing and reuse, Gruber suggests a set of overall guidelines for converting a possible “*monolithic system into reusable building blocks*” (Gruber 1991). He proposes the use of three important proven decomposition techniques from Artificial Intelligence (Gruber 1991):

1. “Separate knowledge from programs with a declarative knowledge representation language.”
2. “Identify general classes and relations underlying application specific facts, and organize knowledge to enable inheritance from these constructs.”
3. “Characterize general problem solving tasks (e.g. classification) and classes of inference (e.g. subsumption), and design corresponding methods and algorithms.”

Two more methods need to be employed to these three techniques (specific to software engineering), in order to obtain shareable and reusable knowledge bases (Gruber 1991):

4. “Specify a *canonical form* for declarative knowledge.”
5. “Define common ontologies [...] with agreed upon definitions in form of human readable text and machine-enforceable, declarative constraints on their well-formed use.”

While presented in this form, these five engineering *tools* do not satisfy the requirements for a methodology (as defined above), they are still useful for they pinpoint the main procedures an ontology developer is performing.

Generally, the practice of building ontologies is highly dependent on the context and goals of a specific project. For this reason, the first (historically speaking) methodologies proposed were based on the experience gained while developing a certain project. Two of the most well known methodologies are the ones created within the Enterprise Ontology project and respectively the TOVE project.

The methodology proposed by Uschold and King (Uschold and King 1995) is based on the experience gained within the Enterprise Ontology project. It provides guidelines for developing ontologies as follows (Uschold and King 1995):

1. *Identify purpose* - why the ontology is being built and what its intended users are
2. *Building the ontology*
 - a. *Ontology capture* – a middle-out approach for identifying the most important concepts rather than most general or most particular ones, followed by generalization and specialization process in order to obtain the remainder of the hierarchy.
 - i. Identification of the key concepts and relationships in the domain of interest (scoping).
 - ii. Production of precise unambiguous text definitions for such concepts and relationships.
 - iii. Identification of the terms to refer to such concepts and relationships.
 - iv. Agreeing on all of the above.
 - b. *Coding* – explicitly representing the knowledge/conceptualization captured at the sub-step above, in a formal language.
 - c. *Integrating existing ontologies* – during either or both of the capture and coding processes, there is the question of how and whether to use ontologies that already exist.
3. *Evaluation* – it is adopted the definition of (Gomez-Perez, Juristo et al. 1995): “to make a technical judgment of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... The frame of reference may be requirements specification, competency questions, and/or the real world.”
4. *Documentation* – guidelines to be established for documenting ontologies, possibly differing according to type and purpose of the ontology.

The methodology proposed by Gruninger and Fox (Gruninger and Fox 1995) based on the development of the TOVE project ontology consists of the following (Gruninger and Fox 1994; Gruninger and Fox 1995):

1. *Capture of motivating scenarios.* The motivating scenarios are story problems or examples that rise from a given situation. In the case of ontologies they (e.g. motivating scenarios) motivate their development and suggest possible solutions that further provide informal intended semantics for the objects and relations that will form the ontology. Any ontology development process should start by describing one or more motivating scenarios and the set of the intended solutions for the problems presented in the scenarios.
2. *Formulation of informal competency questions.* The competency questions are based on the motivating scenarios obtained in the preceding step and can be considered as expressiveness requirements that are in the form of questions. An ontology must be able to represent these questions using its terminology, and be able to characterize the answers to these questions using the axioms and definitions. There is no single ontology associated with a set of competency questions. Instead the competency questions are used to evaluate the ontological commitments that been made to see whether the ontology meets the requirements.
3. *Specification of the terminology of the ontology within a formal language.*
 - 3.1. *Getting informal ontology.* The set of the terms used by ontology can be extracted from the available competency questions. These terms will serve as a basis for specifying the terminology in a formal language.
 - 3.2. *Specification of formal terminology.* The terminology of the ontology is specified using a formalism such as Knowledge Interchange Format (KIF), and later (step 5) will allow to express the definitions and constraints.
4. *Formulation of formal competency questions using the terminology of the ontology.* Once the competency questions have been pose informally and the terminology of the ontology has been formally defined, the competency questions are defined formally.
5. *Specification of axiom and definition for the terms in the ontology within the formal language.* The definitions of terms in the ontology and the constraints on their interpretation are expressed as first-order sentences using axioms. Simply proposing a set of objects alone, or proposing a set of ground terms in first order logic does not constitute an ontology. Axioms must be provided to define the semantics of these terms.

6. *Establish conditions for characterizing the completeness of the ontology.* Once the competency questions have been formally stated, the conditions under which the solutions to the questions are complete must be defined.

When building ontologies, there is nothing to help the developer in selecting the proper technique(s) and tools for his/her needs. After analysing the developing process of a set of representative ontologies (e.g. TOVE and Enterprise Ontology), Uschold has created a general framework that helps the developers to “better understand how to chose the most appropriate techniques for their particular set of circumstances” (Uschold 1996). The framework should identify the common steps and techniques applicable in all the cases and the conditions that require specific steps and techniques (Uschold 1996). The framework consists of a set of five sequential steps every step containing sub steps and/or techniques to apply and/or guidelines to follow (Uschold 1996):

1. Identify the *purpose* of the ontology: detection of the target users, the specification of the purpose relative to the range of purposes already identified (see above Uschold typology for ontologies), the elaboration of motivation scenarios and competency questions for further clarifications of the purpose, and the writing of a user requirement document.
2. Decide the *level of formality*.
3. Identify the *scope* (what *is* and what *is not* in the ontology): motivating scenarios and informal competency questions, brainstorming and trimming.
4. *Build* the ontology.
 - a. In the case of simple and small ontologies, any or all of the steps presented above may be ignored and the building of ontology is just done without any prerequisites.
 - b. Also for small and simple ontologies, after the first steps are covered, the formal encoding may begin.
 - c. From the results of the previous steps a complete document, that is an informal ontology, has to be generated. In some cases this can also be the final result, but usually it represents the starting point for formal encoding and is also viewed as the documentation.
 - d. The construction of the informal ontology can be replaced with the process of identifying the formal terms from the set of informal terms generated at the third step. These (i.e. the formal terms) are used to translate the informal competency questions in formal competency questions that in turn, would enable the specification of axioms and definitions (i.e. the ontology).

5. Evaluate/Revise

The METHONTOLOGY approach (Fernandez, Gomez-Perez et al. 1997; Gomez-Perez 1998; Fernandez-Lopez, Gomez-Perez et al. 1999) has been developed within the Laboratory of Artificial Intelligence at the Polytechnic University of Madrid and is used for building ontologies either from scratch, reusing other ontologies as they are, or by a process of reengineering them. The skeleton of the Methontology framework is based on the IEEE 1074-1995 standard (Fernandez-Lopez 2001), for Developing Software Life Cycle Processes (IEEE96 1996). The framework is supported by the ODE (Ontology Development Environment) tool (Blazquez, Fernandez et al. 1998; Fernandez-Lopez, Gomez-Perez et al. 1999) and WEB-ODE, a scalable workbench for Ontological Engineering (Arpírez, Corcho et al. 2001). The proposed framework includes the following main processes (Blazquez, Fernandez et al. 1998):

1. *Identification of the Ontology Development Process*
 - 1.1. *Project Management Activities*: planning, control, quality assurance;
 - 1.2. *Development-Oriented Activities*: specification, conceptualization, formalization, implementation and maintenance;
 - 1.3. *Support Activities*: knowledge acquisition, evaluation, integration, documentation and configuration management;
2. A life cycle based on evolving prototypes.
3. *Particular techniques for carrying out each activity*

The Ontology Development Process identifies *what* activities need to be carried out when building an ontology and consists of three categories of activities as shown in table 3.2 (Fernandez-Lopez, Gomez-Perez et al. 1999; Fernandez-Lopez 2001) (Fernandez, Gomez-Perez et al. 1997).

Category of Activities	Activity	Description
Project Management	Planning (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Identify which tasks are to be performed and how they will be arranged; • Determine how much time and what resources are needed for the completion of the tasks identified;
	Control	<ul style="list-style-type: none"> • Guarantees that the planned tasks are completed in the manner they were intended to be performed;
	Quality Assurance	<ul style="list-style-type: none"> • Assures that the quality of each and every product outputted is satisfactory;

Development-Oriented	Specification (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Why the ontology is being built; • What are its intended uses; • Who are the intended users;
	Conceptualization (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Structures the domain knowledge as meaningful models at the knowledge level;
	Formalization (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Transforms the conceptual model into a formal or semi-computable model;
	Implementation (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Builds computable models into a computable language;
	Maintenance	<ul style="list-style-type: none"> • Updates and corrects the ontology;
Support	Knowledge acquisition (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Acquires knowledge of a given domain;
	Evaluation (Gomez-Perez, Juristo et al. 1995; Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Makes a technical judgment of the ontologies, their associated software environments and documentation with respect to a frame of reference during each phase and between phases of their life cycle (Gomez-Perez, Juristo et al. 1995);
	Integration (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Required when building a new ontology by reusing other ontologies that are already available;
	Documentation (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Details, clearly and exhaustively, each and every one of the phases completed and products generated;
	Configuration management (Fernandez, Gomez-Perez et al. 1997)	<ul style="list-style-type: none"> • Records all the versions of the documentation, software and ontology code to control the changes;

Table 3.2 The Ontology Development Process.

The ontology life cycle (see Figure 3.5), based on evolving prototypes, identifies the *set of stages* through which the ontology moves during its lifetime. It also describes what activities are performed during each stage and how the stages are related (Fernandez, Gomez-Perez et al. 1997; Fernandez-Lopez 2001).

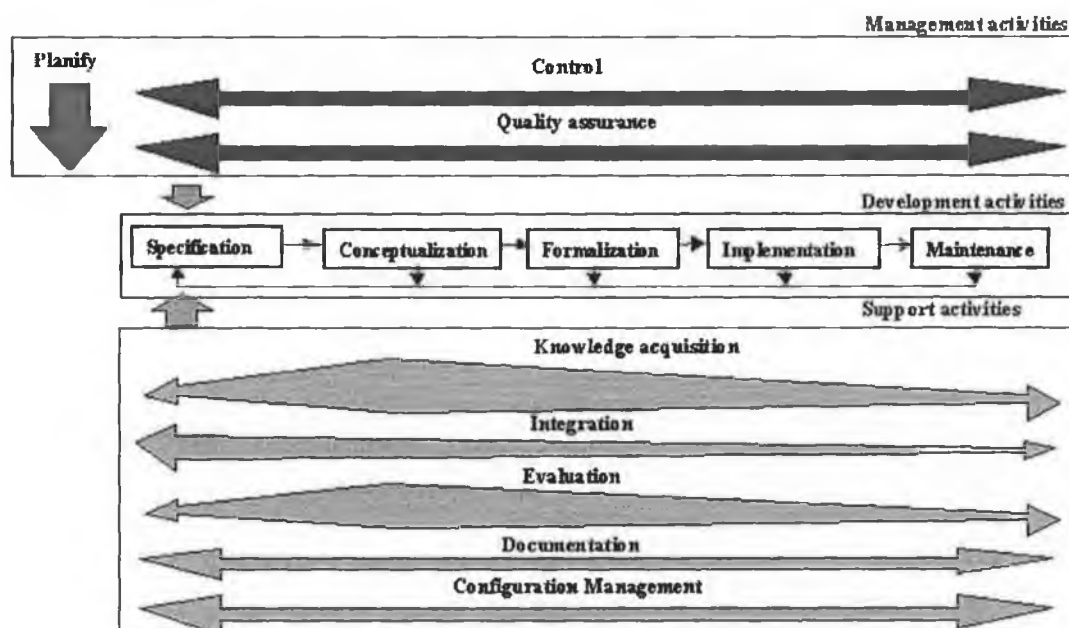


Figure 3.5 Ontology life cycle in the Methontology approach (Fernandez-Lopez 2001)

The construction of each prototype starts with the specification process, sustained by the knowledge acquisition activity. Once the first prototype has been specified, the ontology building continues with the development of the conceptual model, formalization and implementation. During all these phases, the knowledge acquisition activity supplies the ontology development activities with needed knowledge. Also, control, quality assurance, integration, evaluation documentation and configuration management activities are carried out simultaneous to development-oriented activities. Some of these activities have different degrees of intensity during specific life cycle stages (e.g. evaluation and knowledge acquisition are more intense during conceptualization, while the integration activity is more important at the specification time). The Methontology methodology enables a dynamic control of interconnected ontologies (e.g. different activities performed when building an ontology may require performing other activities on already build or under construction ontologies). It also supports the process of ontological reengineering, that is, the process of retrieving and mapping a conceptual model of an implemented ontology to another, more suitable conceptual model, which is re-implemented by the means of reverse engineering, restructuring and forward engineering activities (see Figure 3.6) (Chikofsky and II 1990).

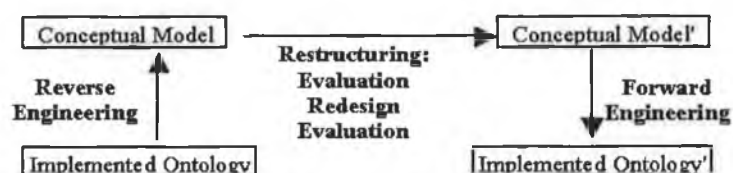


Figure 3.6 The Ontological Reengineering process after (Chikofsky and II 1990).

There are other approaches to methodologies for building ontologies such as Cyc methodology, KAKTUS methodology, SENSUS methodology and OTK (On-To-Knowledge) methodology, but they are usually project specific and they, actually, represent a technical document of how a specific ontology has been built. In fact, the lack of collaboration among different ontology research groups has resulted in a variety of proposals for ontology building methodologies (for each group applies its own methodology).

A unified methodology is needed for the ontology field to be removed from research laboratories into computational world. Moreover, none of the methodologies proposed are as *mature* as methodologies from the knowledge engineering and software engineering fields. This is because no one has undergone a complete test and validation process. On one hand, there are no tools available for the test and the validation of methodologies for building ontologies, and on the other hand, because of its relatively young age, there are not many/enough ontology developers to practically test the different methodologies. Given this current state, the Methontology methodology seems to be the most appreciated one since the Foundation for Intelligence Physical Agents (FIPA) has recommended it for ontology construction (Gomez-Perez 1999).

3.3 Software Agents

Software agents represent an important and fast growing area of AI and more generally of Computer Science (Bradshaw 1997; Green, Hurst et al. 1997; Jennings 2000). The starting point for software agents was formed by Multi-Agent Systems (MAS) which is one of the three research areas of a relatively youthful branch of AI – Distributed Artificial Intelligence (DAI). Therefore, agents inherit potential benefits from both DAI e.g. modularity, speed, reliability and AI e.g. operation at knowledge level, easier maintenance, reusability, platform independence (Nwana 1996).

3.3.1 Background

Started in the early eighties, the research in the area of software agents and MAS evolved into what is now “one of the most active areas of research and development activity in computing generally” (Wooldridge and Ciancarini 2001). Dealing with collections of interacting, coordinated knowledge-based processes (Gasser 1998), DAI demonstrates a distinct feature through the communication and coordination among intelligent and autonomous agents during a problem solving process. This approach decomposes the complexity of the domain problem (agents work together in a problem solving team as opposed to a single agent dealing with a problem) and enhances the system’s performance (Chu, Srihari et al. 1996).

Based on a vast experience in using agent-based techniques, Jennings suggests the necessity of using autonomous agents for developing robust and scalable software systems. He brings two arguments to this statement (Jennings 2000):

1. The Adequacy Hypothesis: “Agent-oriented approaches can significantly enhance our ability to model, design and build complex, distributed software systems”.
2. The Establishment Hypothesis: “As well as being suitable for designing and building complex systems, the agent-oriented approach will succeed as a mainstream software engineering paradigm”.

Jennings believes that agent-based techniques will become widely adopted since “the agent-based approach can be viewed as a natural next step in the evolution of a whole range of approaches to software engineering” and “agent-based techniques are the ideal computational model for developing software for open, networked systems” (Jennings 2000).

Also, Wooldridge and Ciancarini indicate that intelligent agents and MAS represent techniques to manage the complexity inherent in software systems. The reasons why agents are considered an important new direction in software engineering can be summarised as follows (Jennings 2000; Wooldridge and Ciancarini 2001):

- *Natural metaphor.* “Just as many domains can be conceived of consisting of a number of interacting but essentially passive objects, so many others can be conceived as interacting, active, purposeful agents”.
- *Distribution of data or control.* The overall control of many software systems is distributed across different computing nodes that can be geographically and temporally dispersed. “In order to make such systems work effectively, these nodes must be capable of autonomously interacting with each other – they must be agents”.

- *Legacy systems.* “A natural way of incorporating legacy systems into modern distributed information system is to agentify them”.
- *Open systems.* In order to make open systems work effectively, “the ability to engage in flexible autonomous decision-making is critical”.

3.3.2 Definition

Over the last years, many researchers in the area of agents and agent-based systems have offered a variety of definitions for the notion of agency. Nwana notes, “we have as much chance of agreeing on a consensus definition for the word agent as AI researchers have of arriving at one for artificial intelligence itself” (Nwana 1996). This general problem in AI of defining “intelligence” led to an extensive discussion about whether “some particular system is an agent, an intelligent agent or merely a program” which generated as many definitions as there are researchers (Anumba, Ugwu et al. 2002).

Table 3.3 summarizes the most important definitions proposed by researchers in the area of software agents.

Author(s)	Reference	Definition
S. Russell P. Norvig	(Russell and Norvig 2003)	<i>An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.</i>
P. Maes	(Maes 1995)	<i>Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks that they are designed for.</i>
H.S. Nwana	(Nwana 1996)	<i>When we really have to, we define an agent as referring to a component of software and/or hardware which is capable of acting exactly in order to accomplish tasks on behalf of its user.</i>
S. Franklin A. Graesser	(Franklin and Graesser 1996)	<i>An autonomous agent is a system situated within and part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.</i>
Y. Shoham	(Shoham 1998)	<i>An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments.</i>
N.R. Jennings M. Wooldridge	(Jennings and Wooldridge 1998) (Wooldridge 1999)	<i>An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.</i>

FIPA ¹ (standard)	(Poslad, Buckle et al. 2000)	<i>An agent is an encapsulated software entity with its own state, behavior, thread of control, and an ability to interact and communicate with other entities – including people, other agents, and legacy systems.</i>
---------------------------------	------------------------------	--

Table 3.3 Various definitions of an agent

Although many definitions have been given, most researchers agree that *autonomy* is a crucial property of an agent (Nwana 1996; Wooldridge 1999; Jennings 2000). Autonomous agents act on behalf of their users, so therefore they can take decisions without the intervention of humans or other systems. These decisions are based on the individual state and goals an agent has. An agent should act in such a manner as to pursue its internal goals. Autonomy implies that agents have control both over their internal state and over their behaviour (Wooldridge 1999; Jennings 2000).

Other than autonomy, many researchers consider that an agent *should* also be characterised by one or more of the following properties (Nwana 1996; Wooldridge 1999):

1. *Reactivity*: An agent is situated in an environment and is able to perceive this environment and to respond to changes that occur in it (reactive behavior).
2. *Pro-activeness*: An agent should have the ability to take the initiative in order to pursue its individual goals (goal-directed behavior).
3. *Cooperation*: An agent should have the capability of interacting with other agents and possibly humans via an agent-communication language.

Being embedded in a particular environment, agents receive inputs about the state of that environment through sensors and they can perform actions through effectors. Jennings et al refer to this concept using the term *situatedness* (Jennings 2000). The actions of an agent will potentially affect its environment. Wooldridge refers to this ability of an agent to modify its environment through the performance of an action as the agent's *effectoric capability* (Wooldridge 1999). An action has a set of associated pre-conditions that specify the possible situations when it can be performed.

Nwana considers *pro-activeness* a key element of an agent's autonomy (Nwana 1996) while Wooldridge and Ciancarini list autonomy and pro-activeness as two separate properties an agent should have (Wooldridge and Ciancarini 2001). Furthermore, these properties are more challenging than they seem. Agents should attempt to achieve their

¹ Foundation for Intelligent Physical Agents (<http://www.fipa.org/>) is a non-profit standard organization established in 1996, which promotes the creation of specifications of generic agent technologies.

goals but not continuously: that is, agents should cancel actions when it is clear that those actions will not work (because of some factors that modified the environment for example) or when the goal of the action is not longer valid. In such a situation, reactivity should be demonstrated: the agent should react to the events that occur in its dynamic environment. While pro-activeness (in a system that exhibits goal-directed behavior) and reactivity (in a purely reactive system) can be easily implemented independently, integrating goal-directed and reactive behaviour within a system turns out to be very difficult. This problem of achieving an effective balance between pro-activeness and reactivity represents one of the key problems of the agent designer and is basically still open to discussion (Wooldridge and Ciancarini 2001).

The third property of an agent i.e. *cooperation* involves the ability of an agent to dynamically negotiate and coordinate (Wooldridge 1999). Nwana considers cooperation to be the reason for having multiple agents situated in an environment instead of having just one agent in an environment. Because of their social ability, agents can cooperate with other agents and humans. However, Nwana notes that coordination among different agents is possible without cooperation (Nwana 1996).

Some researchers in the area of agents and MAS add a number of other properties to characterise agents as follows (Franklin and Graesser 1996; Nwana 1996; Bradshaw 1997):

- *Learning*: An agent should have the ability to learn while acting and reacting in its environment.
- *Mobility*: A mobile agent has the ability to move around a network (even from one platform to another) in a self-directed way.
- *Temporal continuity*: The actions of an agent are performed through a continuous running process (over long periods of time).
- *Personality*: An agent should manifest a believable character and emotional state.

To summarise, this thesis considers that a software agent is a computer system situated in an environment that acts on behalf of its user and is characterised by the properties such as autonomy, cooperation, reactivity, pro-activeness, temporal continuity and learning. Autonomy is definitely the most important property of an agent without which the notion of agency would not exist. Furthermore, cooperation among different software agents may be very useful in achieving the objectives an agent has. The ideal is an agent characterised

by all the above-mentioned properties but the design and implementation of such an agent is yet a very difficult and complex task.

3.3.3 Typologies

As indicated in the previous section, the term “agent” is an elusive one. Psychology, sociology, economics and AI (and Computer Science more generally) are using it with equivalent meanings but through different perspectives. There are several classification schemes or taxonomies proposed in the agent research community from which the following three are well acknowledged:

1. Gilbert’s scope of intelligent agents (Bradshaw 1997)
2. Nwana’s primary attribute dimension typology (Nwana 1996)
3. Franklin and Graesser’s agent taxonomy (Franklin and Graesser 1996)

1. Figure 3.7 presents Gilbert’s scope of intelligent agents (Bradshaw 1997).

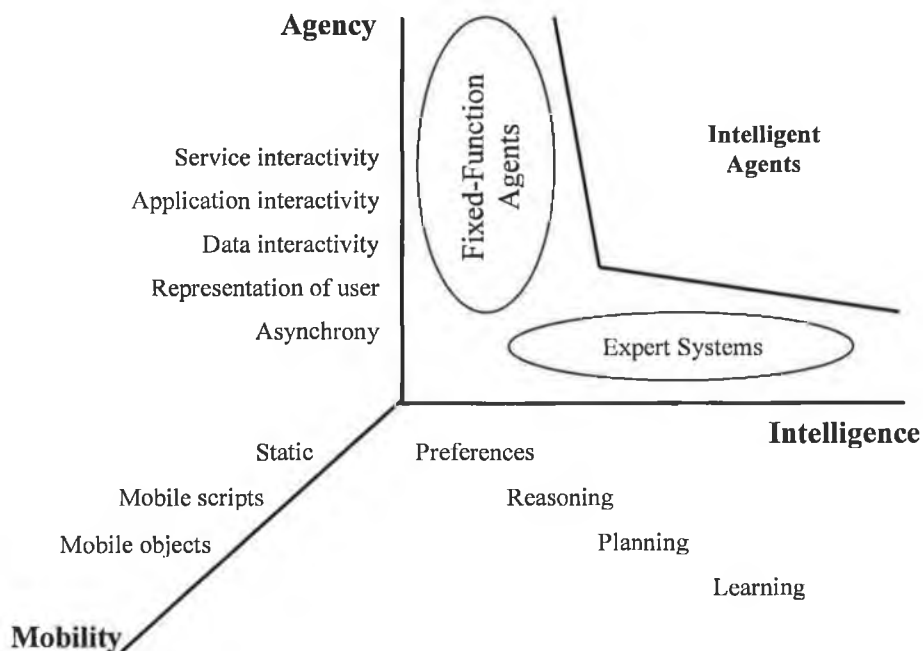


Figure 3.7 Scope of intelligent agents (adapted from Gilbert et al. by (Bradshaw 1997))

Gilbert et al described intelligent agents using the following three dimensions (Bradshaw 1997):

- *Agency* “is the degree of autonomy and authority vested in the agent, and can be measured at least qualitatively by the nature of the interaction between the agent

and other entities in the system. At minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some.” (Gilbert et al. 1995 as cited by (Bradshaw 1997)).

- *Intelligence* is the degree of reasoning and learned behaviour. Furthermore, intelligent agents should learn and adapt to their environment in terms of the user’s objectives and the resources available.
- *Mobility* is the degree to which the agents travel through the network.

2. Nwana uses the three minimal characteristics an agent should exhibit i.e. autonomy, cooperation and learning to classify agents in four categories as follows (see Figure 3.8) (Nwana 1996):

- *Collaborative agents*: There is more emphasis on cooperation and autonomy than on learning
- *Collaborative learning agents*: There is more emphasis on cooperation and learning than on autonomy.
- *Interface agents*: There is more emphasis on autonomy and learning than on cooperation.
- *Smart agents*: These agents implement all three properties equally.

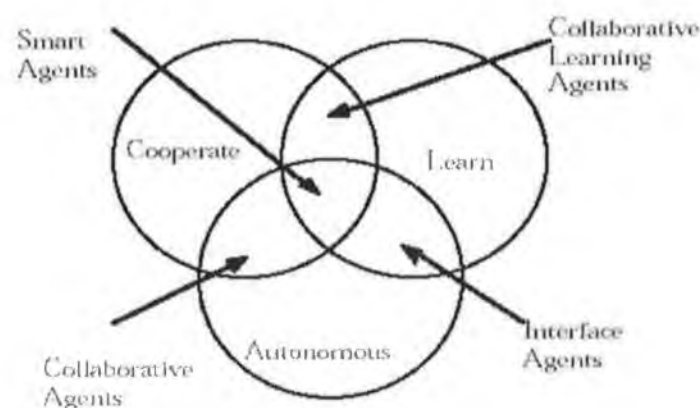


Figure 3.8 Nwana’s agent typology (Nwana 1996)

Mobility can also be used to classify agents in *static* or *mobile* while the presence of a symbolic reasoning model results in *deliberative* or *reactive* agents. Combining these types of agents with the ones already identified based on the ideal and primary attributes of an agent (as considered by Nwana) can produce other categories of agents such as static

deliberative collaborative agents, mobile reactive collaborative agents, static deliberative interface agents, mobile reactive interface agents, etc. Another classification proposed by Nwana uses the roles of agents and is exemplified with *information* or *internet* agents. This category of agents manages large databases in wide area networks like the internet. The last category of agents identified by Nwana is *hybrid* agents, which combine two or more agent philosophies. Furthermore, Nwana uses these agent typologies to identify only seven types of agents as follows (Nwana 1996):

1. *Collaborative agents* are “able to act rationally and autonomously in open and time-constrained multi-agent environments”.

Key characteristics: autonomy, social ability, responsiveness and pro-activeness.

2. *Interface agents* support and assist the user when interacting with one or more computer applications by learning during the collaboration process with the user and with other software agents.

Key characteristics: autonomy, learning (mainly from the user but also from other agents), and cooperation with the user and/or other agents.

3. *Mobile agents* are autonomous software programs capable of roaming wide area networks (such as WWW) and cooperation while performing duties (e.g. flight reservation, managing a telecommunications network) on behalf of its user.

Key characteristics: mobility, autonomy and cooperation (with other agents – for example, to exchange data or information).

4. *Information/Internet agents* are designed to manage, manipulate or collate the vast amount of information available from many distributed sources (information explosion). These agents “have varying characteristics: they may be static or mobile; they may be non-cooperative or social; and they may or may not learn”.

5. *Reactive agents* act/respond to the current state of their environment based on a stimulus-response scheme. These agents are relatively simple and interact with other agents in basic ways but they have the potential to form more robust and fault tolerant agent-based systems.

Key characteristics: autonomy and reactivity.

6. *Hybrid agents* combine two or more agent philosophies into a single agent in order to maximise the strengths and minimise the deficiencies of the most relevant techniques (for a particular purpose).

7. *Smart agents* are equally characterised by autonomy, cooperation and learning.

Furthermore, heterogeneous agent systems are obtained by combining agents from two or more of these categories. Unlike hybrid agent architectures, this agent category refers to an

integrated set-up of at least two or more types of agents (including hybrid agents). Agent-based software engineering facilitates the interoperation of miscellaneous software agents. An agent communication language is necessary for the communication process among different agents (Nwana 1996). This category of agent systems is generally referred to (by most researchers) as multi-agent systems and is discussed in more detail in the next section of this thesis.

3. Franklin and Graesser proposed the taxonomy of autonomous agents presented in Figure 3.9 (Franklin and Graesser 1996).

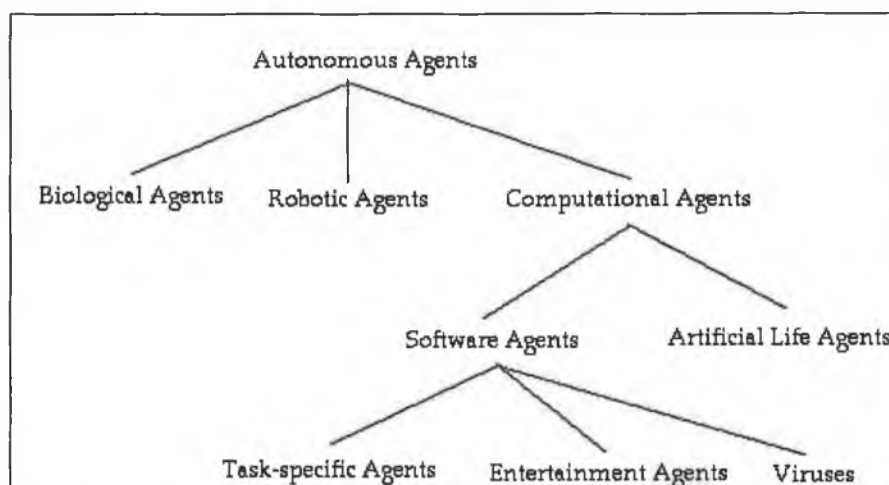


Figure 3.9 The taxonomy of agents proposed by Franklin and Graesser (Franklin and Graesser 1996)

This taxonomy includes biological, robotic and computational agents at the kingdom level, software agents and artificial life agents at the phylum level and task-specific agents, entertainment agents and computer viruses at the class level. A further taxonomy can be performed using schemes such as classification via the agent's control structures (e.g. regulation, planning and adaptive), via environments (e.g. database, file system, network, internet), via languages (in which the agent is written) and via applications. These sub-classification schemes provide a collection of features for an agent and therefore a possible category of classification (Franklin and Graesser 1996).

3.3.4 Multi-Agent Systems

MAS researchers study the behaviour of a group of autonomous agents (possibly pre-existing), which are working together towards a common goal. Including several interacting agents, MAS systems represent a great potential of agent-based systems. MAS

systems are ideal for solving complex problems for which some or all of the following apply (Jennings, Sycara et al. 1998):

- Multiple problem solving methods
- Multiple perspectives
- Multiple problem solving entities

The increasing interest in MAS research is motivated by many potential advantages including the following (Bradshaw 1997; Green, Hurst et al. 1997; Gasser 1998; Jennings, Sycara et al. 1998; Martin, Plaza et al. 1998):

- MAS systems provide robustness, efficiency, flexibility, adaptivity and scalability.
- MAS systems allow inter-operation of multiple existing legacy systems (e.g. expert systems, decision support systems).
- MAS systems have the ability to solve problems that are too large or complex for a single centralised agent
- MAS systems can cope with domains in which data, expertise, or control is distributed (in domains such as distributed sensing, medical diagnosis or air-traffic control, knowledge or activity is inherently distributed).
- MAS systems can enhance speed, reliability and extensibility.
- MAS systems offer conceptual clarity and simplicity of design.

Jennings et al define the term MAS as a “loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver” (Jennings, Sycara et al. 1998). The problem solvers from this definition are autonomous and possibly heterogeneous agents. A MAS system is characterised by the following (Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999; Lazansky, Stepankova et al. 2001):

- A MAS system consists of a collection of agents.
- Each agent acts autonomously.
- The agents in a MAS system are able to interact in order to reach an overall goal.
- Each agent has a limited set of problem solving capabilities.
- There is no global system control.
- Data is decentralized.
- Computation is asynchronous.

It is clear from the definition and from the main characteristics of a MAS system, that inter-operation among autonomous agents is essential to successfully find a solution to a given problem.

Agent-oriented interactions include simple information interchanges as well as planning of interdependent activities for which cooperation, coordination and negotiation are fundamental. Jennings notes that these agent interactions differ from those that occur in other computational models from two perspectives (Jennings 2000). Firstly, an agent knows which goals should be followed and, therefore, agent-oriented interactions are conceptualised as taking place at the knowledge level. Secondly, agents are flexible entities in an environment over which they have partial control and, therefore, they have to make run-time decisions about their interactions that were not foreseen at design time (Bradshaw 1997; Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999).

Since agents in a MAS system have to exchange information and knowledge in order to solve a problem coherently, the following areas have become of crucial importance in MAS research (Green, Hurst et al. 1997):

1. *Coordination*: The agents in a MAS system must coordinate their activities (to determine the organisational structure in a group of agents and to allocate tasks and resources).
2. *Negotiation*: Agents must negotiate if a conflict occurs.
3. *Communication*: Any agent in a MAS system must be able to communicate with other agents. An agent communication language (ACL) enables agents to collaborate with each other providing them with the means of exchanging information and knowledge (Labrou, Finin et al. 1999).

Coordination is considered a central issue to MAS research. Agent-oriented interaction would be ineffective without a valuable coordination among cooperative agents working together towards a common goal. Nwana et al define coordination as “a process in which agents engage in order to ensure a community of individual agents acts in a coherent manner” (Nwana, Lee et al. 1996). Coordination is considered an essential aspect of MAS systems for several reasons as follows (Nwana, Lee et al. 1996; Green, Hurst et al. 1997):

- Coordination prevents anarchy or chaos during conflicts. Such a situation is possible because each agent has a partial view over its environment and therefore, its actions might interfere with rather than support other agents’ actions.

- Agents' behaviours have to be coordinated to meet global constraints e.g. a MAS system constructing a design has to work within the constraints of a pre-specified budget.
- Coordination is necessary because agents in a MAS system have different and limited capabilities and expertise (distributed expertise, resources or information).
- Interdependent activities require coordination (an agent's action might depend on the completion of another agent's task).
- Coordination enables efficiency. One agent can discover information that is of sufficient use to another agent even if their activities are independent.

Nwana et al indicate that coordination may require *cooperation* (although coordination can also occur without cooperation) but cooperation among agents does not necessarily result in coordination. Also, *communication* among agents may be required for coordination but agents can also be coordinated without communication via organisation provided they possess models of each other's behaviours (Nwana, Lee et al. 1996). Furthermore, coordination does not imply reciprocation since an agent can coordinate its activities with those of another agent unaware of its presence (Durfee 2001).

Researchers have proposed various coordination techniques including the following (Nwana, Lee et al. 1996; Green, Hurst et al. 1997; Oliveira, Fischer et al. 1999):

- Organisational structuring (Werkman 1990; Carver, Lesser et al. 1993; Tsvetovatyy, Gini et al. 1997)
- Contract Net Protocol (CNP) (Nwana, Lee et al. 1996; Green, Hurst et al. 1997)
- Multi-agent planning (Lesser and Corkill 1981; Durfee and Lesser 1991; Nwana, Lee et al. 1996; Green, Hurst et al. 1997)
- Social laws (Chaib-draa 1996; Green, Hurst et al. 1997)
- Computational market-based mechanisms (Oliveira, Fischer et al. 1999)

Many researchers have studied the subject of *negotiation* providing many and diverse techniques and definitions for this term through a vast literature (Nwana, Lee et al. 1996; Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999). Used for conflict resolution, negotiation is a significant aspect of the coordination process among autonomous agents in a system. Furthermore, negotiation is seen by many agent researchers as a key coordination technique also used to address several DAI issues (Nwana, Lee et al. 1996; Jennings, Sycara et al. 1998). Bussman and Muller define negotiation as "the communication process of a group of agents in order to reach a

mutually accepted agreement on some matter” (as cited in (Green, Hurst et al. 1997)). Jennings et al consider the following to be the main characteristics of negotiation (Jennings, Sycara et al. 1998):

- The existence of a conflict.
- Self-interested agents have to resolve the conflict in a decentralised manner.
- Bounded rationality.
- Incomplete information.

Many researchers argue that agents must reason about beliefs, desires and intentions of other agents for an effective negotiation process (Rao and Georgeff 1995; Nwana, Lee et al. 1996). The available negotiation techniques involve the use of human negotiation strategies, logic, case-based reasoning, multi-attribute utility theory, belief revisions, distributed truth maintenance, model-based reasoning, optimisation and game theory (Zlotkin and Rosenschein 1989; Nwana, Lee et al. 1996; Zlotkin and Rosenschein 1996; Green, Hurst et al. 1997; Jennings, Sycara et al. 1998; Oliveira, Fischer et al. 1999; Shintani, Ito et al. 2000).

To any MAS system, *communication* is essential in order to benefit from the added value provided by a collection of agents. Because agents generally have only a partial view over their environment, they will probably be required to *communicate* (to exchange information and knowledge in a distributed environment or to request the performance of a task) with each other in order to effectively cooperate. Nwana and Ndumu place communication “at the heart of cooperation and competition germane to multi-agent systems” (Nwana and Ndumu 1999). An *agent communication language* (ACL) should facilitate the interactions between two or more agents (through communication by exchanging messages) (Genesereth and Ketchpel 1994; Nwana and Wooldridge 1996; Green, Hurst et al. 1997; Labrou, Finin et al. 1999; Chaib-draa and Dignum 2002). Chaib-draa notes, “the main objective of an ACL is to model a suitable framework that allows heterogeneous agents to interact, to communicate with meaningful statements that convey information about their environment or knowledge” (Chaib-draa and Dignum 2002). The most known ACLs are the Knowledge Query and Manipulation Language (KQML) proposed by the Knowledge Sharing Effort (KSE) consortium (Finin, Fritzson et al. 1994) and the FIPA ACL proposed by the Foundation for Intelligent Physical Agents (FIPA) (<http://www.fipa.org>; Labrou, Finin et al. 1999; Poslad, Buckle et al. 2000).

Besides an ACL, a common understanding of the concepts used among agents is necessary for a meaningful agent communication. This is because agents may have different terms for the same concept or identical terms for different concepts (Odell 2000). Therefore, ontologies are used for representing the knowledge from various application domains. The ACL remains just syntax without a shared common ontology containing the terms used in agent communication and the knowledge (e.g. definitions, attributes, relationships between terms and constraints) associated with them (Nwana and Wooldridge 1996).

3.3.5 Final Remarks

Generally, the researchers in the area of agents and MAS systems believe that agent-based computing has the potential to improve the conceptualisation, design and implementation of complex distributed software systems. Even if the software agents are still bounded to the research laboratories, significant advances have been made towards defining an agent-oriented methodology and implementing agent languages and environments.

The available *methodologies* for the analysis and design of agent-based systems can be classified in two groups as follows (Iglesias, Garijo et al. 1999; Wooldridge and Ciancarini 2001):

- Methodologies that extend or adapt object-oriented methodologies e.g. AAIL (Kinny, Georgeff et al. 1996), Gaia (Wooldridge, Jennings et al. 2000), MaSE (DeLoach 1999), AUML (Odell 2000).
- Methodologies that adapt knowledge engineering models or other techniques e.g. CoMoMAS and MAS-CommonKADS (Iglesias, Garijo et al. 1999), DESIRE (Brazier, Dunin-Keplicz et al. 1997).

Although numerous *languages* and platforms have been created by different research groups and companies to support the development of agent-based applications, traditional languages are still used to construct agent applications. Nwana and Wooldridge indicate, “typically, object-oriented languages such as Smalltalk, Java or C++ lend themselves more easily for the construction of agent systems” (Nwana and Wooldridge 1996). The reason for this is that Agent Oriented Programming (AOP) and Object Oriented Programming (OOP) techniques share some properties such as encapsulation, inheritance and message passing.

While there are still many problems associated with the design and implementation of agent-based applications, MAS systems “*provide a powerful model for computing in the*

21st century, in which networks of interacting, real-time, intelligent agents seamlessly integrate the work of people and machines, and dynamically adapt their problem solving to effectively deal with changing usage patterns, resource configurations and available sources of expertise and information” (Lesser 1999).

3.4 Conclusions

The present chapter has depicted the state of the art in the research concerning ontologies and software agents, the two technologies that are envisioned to enable semantics among human designers and their machines within the distributed engineering design system.

Specifying content specific agreements in a consensual manner, the ontologies facilitate knowledge sharing and reuse among systems by the means of ontological commitments (Gruber 1995; Spyns, Meersman et al. 2002). The general vision is that ontologies should be machine-enabled and, if not directly human-readable, they should at least contain plain text notices or explanations of concepts and relations for the human user (Borst, Akkermans et al. 1997; Guarino 1998; Studer, Benjamins et al. 1998; Uschold 1998; Fikes 1999; Sowa 2000; Noy and McGuinness 2001).

The agent-based approach is suitable and beneficial for environments that are naturally modeled as societies of autonomous cooperating components (the agent is a natural metaphor) as well as for systems that contain legacy components (Jennings and Wooldridge 1998). These environments are generally open, distributed, complex, highly dynamic and require flexible interaction and openness (Oliveira, Fischer et al. 1999) (Wooldridge 1998).

The combined capabilities of ontologies and software agents make them the appropriate candidates to semantically enable and manage DEDES, a system in which data, control, expertise and resources are inherently distributed.

The next chapter will recommend a supporting ontology-based and agent-based architecture to enable cooperation management within the DEDES.

Chapter 4

Semantic Framework for the Distributed Engineering Design System

4.1 Introduction

4.2 Thinking the Distributed Engineering Design System

4.3 The Need for Semantic Support

4.4 Proposed Framework for Enabling Semantics within the DEDS

4.5 Conclusions

4.1 Introduction

The systems approach to the distributed engineering design organization has defined cooperation as a critical component for the system's information management (see chapter two, sections 2.2.3 and 2.3.3). The approach has also stressed the cooperation process's reliance on data, information and knowledge (as they form its working material). Consequently, a set of requirements for improving DEDES functionality, in terms of information management, has been proposed (in chapter 2). These requirements together with the underlining approach will further translate into an architectural framework that is aimed to be proposed in this chapter.

Systems Thinking provides an holistic and practical mode of thinking about complex systems (such as DEDES) as wholes (Richmond 1994; Mulej, Vezjak et al. 1999; Bartlett 2001). The author will show that Systems Thinking facilitates a deeper and more intimate understanding of the DEDES as a whole, as distinct from the case of "when the only tool you have is a hammer, every problem begins to look like a nail". Given the critical importance attributed to the cooperation process, it will be argued that, the main pattern of the DEDES, concerning its information structures dynamics, consists of a hierarchy of cooperation processes whose ramifications are stretching to all system components.

After acknowledging the pivotal role of the cooperation processes in the DEDES information flows dynamics, the author will be argued, based on the requirements that concluded the second chapter of this thesis, that probably one of the most critical deficiencies for the functioning of the distributed engineering design organization is the poor semantic integration of the information structures into the whole (i.e. the DEDES system). Therefore, the need to provide semantic support is seen as significant for improving the way engineering design is performed in distributed environments. Moreover, given the importance of using computers for designing not only as advanced tools, but also as an engineering design medium or workplace, the semantic support required to be implemented in the form of software tools.

As the result of the conclusions and findings of the research to date, an architectural framework model is proposed to characterize a DEDES context in which ontologies and software agents will be used to implement a computational support for cooperation processes. A structural and functional description of this architecture defines the computational space and behavior of the supporting system.

4.2 Thinking the Distributed Engineering Design System

Systems Thinking is a cognitive and applied mode of thinking aimed at understanding the system as a whole, its behavior and its emerging properties in terms of its structure, its interdependencies and the patterns that describe it (Richmond 1994; Mulej, Vezjak et al. 1999; Bartlett 2001).

In practical terms, thinking a system means to emphasize the whole over the individual parts. While the literature describes various frameworks, skills and methods, a consensus to apply this mode of thinking requires the followings (Draper and Swanson 1990; Sterman 1991; Richmond 1994; Ossimitz 1997):

- To think in models and to distinguish between them and reality. This implies that models are to be used to focus the awareness on the components or behaviors of main importance for the specific research.
- To identify the feedback and the feed-forward loops that interrelate structures and processes and not just one-way cause-effect relations. This approach enables the understanding of how local and global behaviors interact with each other and therefore how the parts work together.
- To recognize patterns and not just events.
- To steer the system by applying the right action at the right time in the right place.

In the context of this thesis, the main purpose of systems thinking is to facilitate an intimate understanding of the DEDES from which an implementational evaluation of the requirements for a supporting framework, can be carried out.

4.2.1 Prerequisites

As identified in chapter two, the distributed engineering design can be seen as a system, i.e. DEDES (see figure 2.3.3), consisting of a hierarchy of three subsystems, as follows:

- *Human System* – the collection of organizationally and hierarchically distributed multi-disciplinary engineering design teams working sequentially, concurrently or in parallel to design.
- *Infrastructure System* – the collection of manual and automated tools (e.g. drawing boards, computers with their applications, books, reports) acting as

the medium or the workplace where the engineering design process takes place.

- *Engineering Design Model System* – integrated life cycle design methods and methodologies for the development of products that guide and inform the process of design.

The performance of the engineering design process is conditioned by the interoperations among the *Human*, *Infrastructure* and *Engineering Design Model* subsystems. Furthermore, these interoperations are mediated by means of information structure exchanges which, in turn, are supported by *cooperation* processes, which control and regulate these exchanges by facilitating inter and intra subsystem communication, co-location, collaboration and coordination.

Due to the fact that the performance of a complex system, such as a DEDS, depends on how all the parts work together, not on how each part performs when taken separately, the author will argue that, in fact, the key information-focused patterns of the DEDS are cooperation processes organized in closed-loop (feed-forwards and/or feedbacks) mechanisms that enable the transformation of engineering design information structures from initial requirements and constraints towards final product specification.

4.2.2 The Cooperation Process

Cooperation is not considered a subsystem, but an emergent property of the DEDS, as the cooperation process causes and is caused by a meaningful operation and co-working of the DEDS components acting as a whole. The author contends that the cooperation process has the following dual nature:

- It is a human-made artifact consisting of specific hardware (e.g. audio and video mechanisms, boardrooms, blackboards, and so on), dedicated software (e.g. messenger type of applications), and methods and methodologies (e.g. meetings, brainstorming, broadcasting, conferences, negotiation strategies).
- It is a control mechanism that regulates the functionality of all the parties (i.e. Human, Infrastructure and Engineering Design Model subsystems) involved in the system.

Therefore, the cooperation process implements inter and intra-subsystem interfaces. That means that the cooperation process influences and determines the exchanges (notably of information structures) between the subsystems and within the subsystems. These behaviors are supported by means of *communication*, *co-location*, *coordination* and *collaboration* processes.

Communication

Based on the model originally developed by Shannon and Weaver (Shannon and Weaver 1963), in their endeavor to lay out the basis for a communication science, Van Cuilenburg et al (VanCuilenburg, Scholten et al. 1991) recommend a fundamental model of communication (see figure 4.1) for representing the *communication* process.

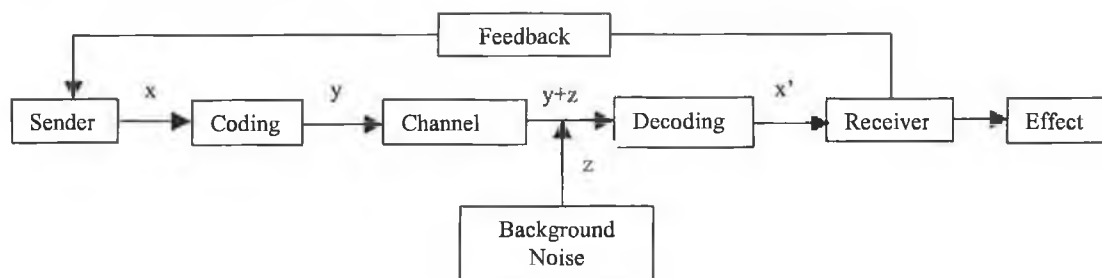


Figure 4.1 The fundamental model of communication (VanCuilenburg, Scholten et al. 1991)

The essential concept of the model is the concept of *message* (i.e. typographically codified as x , y , z , $y+z$, x' in figure 4.1), which represents the actual information or signal sent from a sender to a receiver. For the message to be circulated, the following components and procedures are needed (Shannon and Weaver 1963; VanCuilenburg, Scholten et al. 1991):

- *Sender* (or Encoder): An information source; a person or device that originates a message.
- *Coding*: The procedures and operations required to code a message from receiver's representation (i.e. x) into a representation suitable to be transmitted through the channel (i.e. y).
- *Channel* (or Medium): The method used to transmit a message (e.g., print, speech, telephone).

- *Background Noise*: Technical or semantic obstacles; i.e. anything that interferes with the clear transmission of a message (e.g., audio disturbances, reduced refreshing rate for video frames, poor ink quality, unknown language, different interpretations of the same concept).
- *Decoding*: The procedures and operations required to decode a message to a representation suitable for the receiver (i.e. x').
- *Receiver* (or Decoder): The audience for a message; also known as the addressee.
- *Feedback*: Information about a message that a receiver sends back to the sender; the receiver's reaction or response to a communication.
- *Effect*: The consequence a message has upon the receiver.

Another important concept, not pictured in figure 4.1, is the concept of *interpretation*, which subsumes all the operations performed by a receiver in order to decode and understand a message. A communication process can be considered successful when the receiver's interpretation of the message x' corresponds to the sender's meaning of message x . Therefore, communication is achieved when the representation x of the message is semantically equal or equivalent with the representation x' of the message. Within a DEDS, generally human designers assume the position of the receiver or the sender of a message. Nevertheless, a knowledge base, a computer or a software application can also act as senders and receivers. Therefore, the author contends that the communication process can be initiated or can be received by any component of the DEDS.

Generally, the *material* of the message consists of data, information or/and knowledge. Therefore, communication and hence, cooperation heavily rely on how the system information structures are managed. A poor administration of information and knowledge will amplify the semantic or syntactic noise (or will increase entropy) as well as can *weaken* the message itself.

Co-location

When considered within a human society context, the co-location process refers to the bringing together of participants (i.e. human beings) geographically and temporally, to enable them to use a wide variety of communication channels, such as language, gestures, mimics, and other non-verbal aspects of speech (Chira 2002) with which to

communicate. Within a distributed environment, where the resources (especially human) are geographically dispersed, the role of co-location is greatly reduced, being more idealistic than practical. Nevertheless, a good interface design and the development of applications that implement virtual environments could enhance its positive effects.

When considered within the context of a distributed engineering design environment, the co-location process subsumes the infrastructure to provide effective communication among the distributed participants (i.e. human beings, software tools, information sources, etc). For example, in the scenario that a design engineer needs to know what available materials have a better eco-indicator while with comparable fastening characteristics with a material called `mat_x`, the co-location process may consist of the followings:

1. the designer knows where the appropriate information concerning available materials is, or knows how to find out (e.g. the material department, a folder on his/hers personal computer);
2. the designers knows how to establish the connection with the source of information (e.g. a telephone number and a telephone line, a physical location within the company and the pathway to that location, an OS path to the folder);
3. the designer establishes the connection to the source of information (e.g. making a call, paying a visit, opening a file).

Once these steps are successfully executed, a communication process can then be initiated with the information source (e.g. an employee of the material department, a paper or a computer file, a bill of materials, etc) for fulfilling the initial information needs (i.e. material names).

Coordination

Based on the characterizations presented in chapter two (see section 2.2.2), the author pictures the coordination process as the management of resources, information structures, and communication processes. Its role is to establish a harmonious combination of co-location and communication relations or actions. Therefore, the coordination process describes strategies to synchronise design resources (e.g. human, information, etc) with the purpose of increasing the efficiency of the co-location and the communication of information structures.

In the above scenario, a good coordination process can assure that somebody from the material department will answer the phone and will have access to the needed information. In the case that the collocation process may be achieved by simply opening a computer file, the coordination process supports a strategy to keep that specific file updated at all times.

Collaboration

The collaboration process consists of creating a shared understanding in a distributed environment, so that the DEDES actors can work with one another. This means that the interacting actors from the DEDES share common or equivalent languages at both syntactic and semantic levels.

For example, in the case of the designer requesting material information and the professional from the Material Department, the collaboration process responding to that request means that:

1. they both speak English and,
2. the technical description or query provided by the designer is understood by the Material Department specialist
3. the technical answer of the Material Department specialist is understood by the designer

Equivalently, in the case that the needed information is on a computer file, collaboration means that the designer knows to read the file and then understands the terms used therein.

From a structural and functional perspective the cooperation process is the result of the following four interlaced processes:

1. the *communication* process which describes a mechanism for physically exchanging messages (i.e. information structure),
2. the *co-location* process which provides the physical and informative infrastructure for implementing communications,
3. the *coordination* process which synchronizes design resources, co-locations and communications, and
4. the *collaboration* process which is responsible for semantically integrating collaborations, co-locations and communications in a comprehensible and intelligible whole.

From this perspective, the main role of cooperation is to enable and support the exchanges of data, information and knowledge, by integrating the DEDS actors in a common syntactical and semantical enabled pool of information structures. This, in turn, provides the cooperation process with the means to supervene and influence the control and regulation of the DEDS, as will be shown in the next section.

4.2.3 Cooperation Processes as the Main Information Flow Patterns

The systems approach facilitates the understanding of how local and global behaviors interact with each other. In other words, the systems approach reveals how the parts work together. Knowing the repeating patterns from which the specific system is constituted, will inform this understanding because, in expressing key behaviors, the patterns are more easily comprehended and managed than collections of various structures connected in a web of interdependencies.

After acknowledging (chapter two and above) its critical importance, the author argues that the main pattern within a DEDS consists of a fractal cooperation process (i.e. a pattern that repeats itself on an increasingly smaller scale or a set of self-similar patterns).

As already mentioned, the engineering design process is an information transformation process. Equivalently, at the highest level, the distributed design system is an information transformation system from raw, unstructured information representing the initial requirements and constraints (that may come from different sources) to structured, detailed information representing an artifact or a product (as depicted in figure 4.2).

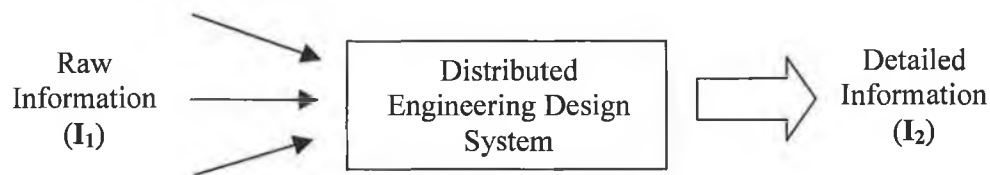


Figure 4.2 A black-box view on DEDS as an information transformation system.

At a closer examination, this process can be represented in a closed-loop model as a cooperation process (see figure 4.3).

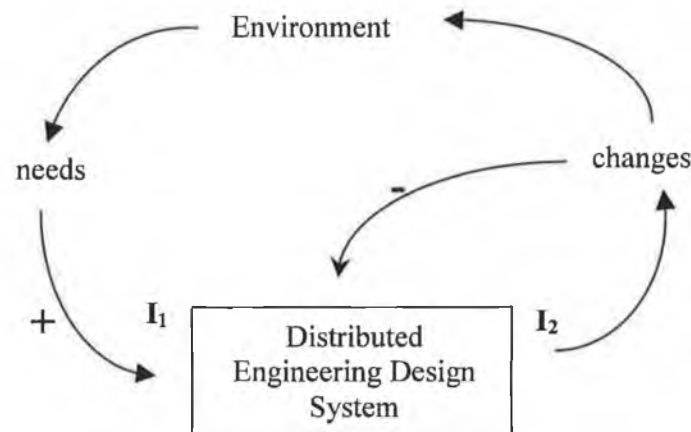


Figure 4.3 High-level DEDS – Environment interaction (“+” sign stays for positive enforcing or feed-forward, “-” for negative enforcing feedback).

As pictured in figure 4.3, specific external needs (e.g. markets in general, but not exclusively) trigger a series of events (e.g. informative meetings and negotiations between the parts, contracts, etc) with the purpose of feeding the DEDS with information that informally describes the desired situation. The co-location and collaboration of the involved parts is required in order to enable the identification of information to be fed forward (therefore to start the design process) into the system. Once these prerequisites have been met, the information that describes the needs and the eventual constraints needs to be represented in an agreed form or message (i.e. I_1 in figure 4.3). In other words, the information is encoded in a message understood by all parties involved. Afterwards, the initial message undergoes a series of specific transformations inside the DEDS, after which the result (i.e. I_2 in figure 4.3) is outputted to the external environment. Its decoding fulfils the triggering needs, therefore changing the state of the environment, which in turn provides the DEDS with feedback, which can further generate other needs. As a fractal, this “high-level” cooperation process repeats its structure within the DEDS.

At the system level (i.e. DEDS level), the Human System, based on and following the methodological specifications defined by the Engineering Design Model System, transforms information structures in a working space determined by the Infrastructure System. In other words, according to the specific phase of the design or depending on specific needs, the cooperation process is dynamically initiated among the DEDS

subsystems. Therefore, it is inaccurate to consider that the solely the engineering designers (i.e. Human System) perform the transformation of information (i.e. the engineering design activity). The author argues that it is the inter-operation or the cooperation among the three subsystems that execute or carry out the engineering design activity (see figure 4.4).

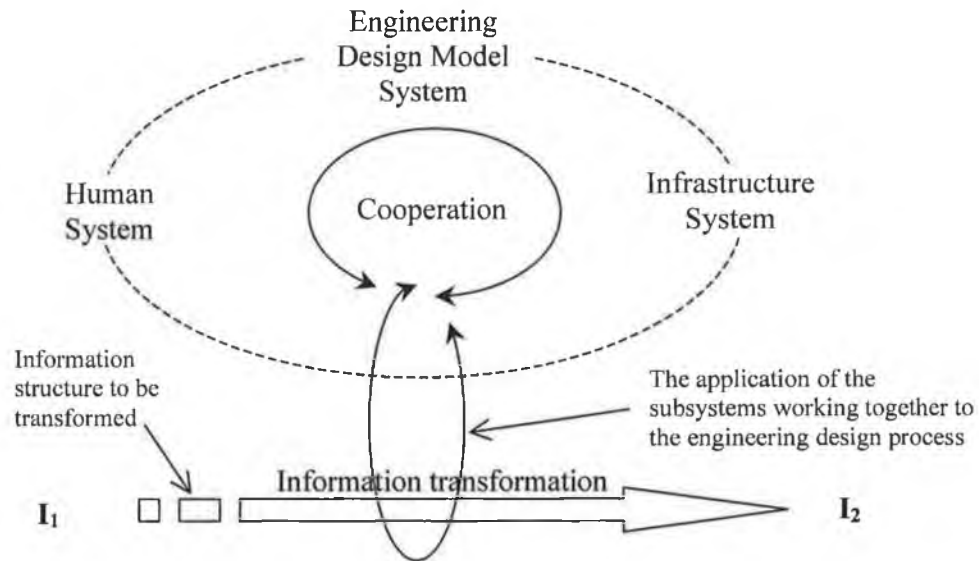


Figure 4.4 The transformation of information from I_1 to I_2 is performed through the cooperation of Human System, Engineering Design Model System, and Infrastructure System.

As the engineering design activity progresses, depending on the partial results, feedback mechanisms may trigger changes, reorganizations or reiterations of activities within the three subsystems (e.g. a delay in the performance of a design team may be dealt with by assigning to it designers proficient in working with CAD tools).

At the next level of detail, the overall cooperation process consists of simpler cooperation processes: cooperation among the three subsystems is made up of the interactions of 'simpler' one-to-one co-operations. The access of Human System to the necessary methodologies from the Engineering Design Model System requires cooperation between these two subsystems. Similarly, the ability of the Human System to perform in the distributed engineering design environment is conditioned by its cooperation with the Infrastructure System. Moreover, a superior performance of the Infrastructure System (e.g. better CAD tools, enhanced process management

software, etc) depends on the inter-exchanges with the Engineering Design Model System (e.g. detailed and formalized methodological steps can trigger software specifications).

Furthermore, the level of detail can be deepened to the subsystem plane and, even further, to the subsystem component level. Whatever the depth (even in the mind of a single designer), the pattern remains the same. A transformation and/or exchange of information is involved. Generically, a cooperation process will, therefore, look like in the following:

- The need for cooperation arises;
- Co-location(s) process ‘puts together’ the participants (components involved in the cooperation);
- Collaboration(s) process generates a shared understanding among the participants;
- Coordination(s) process manages the resources needed for cooperation and the flow of information structures;
- Communication(s) process implements the effective exchange of information structures among participants.

In conclusion, the DEDES, viewed as an information transformation system, consists of an hierarchical series of closed-loops cooperation processes (forming a fractal structure). The material of the structure is made of information configurations, i.e. data, information and knowledge. The participating actors can be any component (however simple or complex) of the DEDES.

4.3 The Need for Semantic Support

Using Systems Thinking, as an holistic technique for gaining knowledge and understanding of the distributed engineering design organization (or DEDES), it has been argued (in section 4.2.) that the DEDES is (informationally) ‘kept together’ by information structures and cooperation processes dynamics. The information structures of the system (i.e. data, information and knowledge) are the means by which the *interactions* among heterogeneous structures (of Human, Infrastructure and Engineering Design Model Systems) are possible. The cooperation processes, in turn, form (by means of communication processes), enable (by means of co-location

processes), regulate (by means of coordination processes) and support (by means of collaboration processes) these information-mediated *interactions*.

In the followings, based on the analysis of the requirements that concluded the second chapter of this thesis, it will be argued that probably one of the most critical deficiencies for the functioning of the distributed engineering design organization is the poor semantic integration of the information structures into the whole. Therefore, given the pivotal role that the information structures play within the DEDES, the need to provide semantic support is seen (by the author) as significant for improving the way engineering design is performed in distributed environments. Moreover, given the importance of using computers for design not only as advanced tools, but also as an engineering design medium or workplace (as shown section 2.2.2), semantic support is further required in the form of software tools. As will be shown, the DEDES cooperation processes qualify for the role of semantic integration, whereby ontologies and multi-agent systems provide the necessary technological infrastructure for implementing the cooperation processes.

4.3.1 A DEDES Requirements Analysis

The requirements needs for a better functionality of the DEDES, as identified in chapter two, can be grouped along the following three dimensions:

1. The need for *reusable, shared and formal structures of information* maintained by the intrinsic logic of the engineering design domain.
2. The need for *integrative mechanisms* capable of translation and mapping between different contexts, i.e. Human, Infrastructure, Engineering Design Model, and other systems from the outside environment with which the DEDES may interact (e.g. Manufacturing, Suppliers, EndOfLife, Market, etc.).
3. The need for non-human/software-based *control mechanisms* able to regulate the functionality of the system (i.e. the human intervention in administrating and controlling system functionality should be minimal, as this would have a beneficial effect on designers' concentration and time spent on effective design).

The first dimension of requirements, i.e. of *reusable, shared and formal information structures*, means that the design data, information and knowledge employed, created,

modified, and used should be easily accessed by any DEDES component (especially the components of Human System, e.g. engineering designers) that may need them. Therefore, given the identified role of the overall DEDES cooperation process, the design information structures should be enabled for any cooperation process within the DEDES.

Furthermore, this requires that the design information structures be represented in such a manner so that their codifying and de-codifying at the sender's respectively receiver's points are achievable, no matter what DEDES components are involved (or, in other words, the information structures circulated within the DEDES are independent on their representation at consumer and producer level). Therefore, the basic strategy adopted requires the representation of system information in a common-agreed form (or pool of knowledge) that transcends the diverse irreducible and irreconcilable local representations. Moreover, this system-level representation should allow the re-representation of information structures into the specific local representations.

In order to achieve such a reversible representation (or a common denominator representation) of system information structures, it is imperative to preserve their meaning and, therefore the relationships among the design information structure. The reason for this is that, once the basic meaning is agreed upon and codified in a formal representation, the re-structuring of information structures for local needs (i.e. its translation towards data or information understood by specific DEDES components) requires only simplifications and reductions, which are easily achievable.

For example, presume that a formal representation of a generic product structure is agreed upon, and the respective information structures are stored accordingly. In order to provide information to a local CAD tool (e.g. ProEngineer), so that specific structural product information can be presented to a designer in a visual form, all that is required is to identify the proper subset of concepts (e.g. product, assembly, part, feature, parameter) and relations (e.g. is_part_of, has_feature, has_parameter) and the necessary instances (e.g. smoke_alarm, cover, button, circular_shape, 25 radius) that are transmitted to the Application Program Interface (API) of the tool. The specific data or information fed to the CAD tool generally does not have any sense or meaning outside the tool or, in other words it does not have a meaning for the system as a whole. This situation greatly reduces the possibility of sharing information and therefore this information may be lost for other tools (e.g. SolidWorks) or other system components that may require product structure related information (e.g. bill of

materials). On the other hand, when the meaning is preserved at the system level it is possible to obtain a local representation (usually meaningless at the system level but meaningful at the local level) for a specific component (provided of course that the respective component has an interface to the system). In summary, metaphorically, the author considers that, while it is improbable to obtain meaning from meaningless, it is possible to obtain meaningless from meaning.

The second dimension of requirements, i.e. the *integrative mechanisms*, refers to need for specific means of 'plugging-in' the various DEEDS components into a functional unit, thus enabling inter-component dialog (or relations). The author identifies two kinds of integration, as follows:

1. *Static integration*
2. *Dynamic integration*

A component that is *statically integrated* can access the most current state of the system. In other words, since for the present research the system is thought and described in terms of its information structures, the statically integrated component is enabled to *read* the appropriate up-to-date system information regardless of its distribution (geographical, semantical or functional). For example, suppose that a project manager wants to check the overall progress of the current project that involves several teams geographically distributed. If he or she is statically integrated into the system, then, when asking its integration mechanism (e.g. a software application) to open the current project, the appropriate and necessary up-to-date information (which is not necessarily the last approved information) is *automatically* collected from throughout the organization, brought locally and presented in a suitable predefined form (e.g. the product structure in a CAD or other visualization application, the temporal progression in a project planning application and so on). The advantages of this scenario are obvious, since all the project manager has to do for collecting the proper information is to use a single application.

A component is *dynamically integrated* when it is able to enter into dialog or establish relationships with any other DEEDS component that, of course, is also dynamically integrated. For example, in the above scenario, if the project manager is dynamically integrated, then his or her integration mechanism would be able to *talk* to other components. Thus, when a change in the state of the system occurs and that change is

interesting for the manager, the integration mechanism would be able to *know* about the change and announce such a change to its dynamically integrated user.

As a final point, since the inter-component relationships are mediated by means of information structures through cooperation processes, the integrative mechanisms need to be able to perform translations of data and information (or even knowledge when and where possible) from specific local representations, as produced by the various system components, in the system form emphasized within the first dimension and vice-versa.

The third dimension of requirements needs, i.e. the *control mechanisms*, concerns the human role in controlling the DEDS functionality. Traditional approaches generally necessitate explicit user actions (e.g. mouse clicks, keyboard inputs) to control the logic flow. This makes them (i.e. the traditional approaches) significantly dependant on user reaction times and expertise. Moreover, in the case of a complex application, the user is required to spend time learning how to operate the application and to spend time administering it. This situation can have a detrimental effect on the user's real productivity, as she or he is required to continuously shift focus between using the tool and working with the tool. Therefore, the proposed solution system should not be human-centered, i.e. its core functionality should not need human mediation. Moreover, from the system's control and regulation point of view, humans need to be seen as any other component of the system (in fact, structurally and functionally humans are part of the Human System). Of course, the human is a very important component for the reason that it is the main producer and consumer of information and, as the same, it is the cause and the purpose of the system. For all these reasons, the human should *use* the system at maximum effectiveness and delegate something else to *work* with the system on her or his behalf. Hence, as any other system component, the human component is integrated or plugged-in by means of the integrative mechanisms. However, since the human inputs and human requirements from the system are much more diverse and complex than more of the other components, the author contends that the human needs special integrative mechanisms that are adaptive, capable of learning and especially tailored for each individual.

Concluding the requirements analysis, the system information structures need to be represented in such a way that their semantics are preserved independent of the local

informational representation needs. Moreover, it is necessary to allow the translations among system level and local levels representations (so the local information structures can be shared and reused). These information-related characteristics will enable the cooperation processes to reach any DEDES components for which integrative mechanisms are implemented, and therefore to facilitate the inter-component dialog. Furthermore, by enabling this intra-system dialog (so therefore the DEDES components are able to interact biasing the human mediation) when possible, the control and regulation of some of the time consuming system behaviors can be shifted from the human components towards automated and autonomous software tools.

4.3.2 Cooperation Processes – the Semantic Enablers

In the preceding section it has been shown that *semantics* can facilitate a superior functionality (or behaviour) of the cooperation processes and, therefore, given that the main patterns within the DEDES consists of cooperation hierarchies, *semantics* can play a major role in the improvement of the distributed engineering design activity. In the subsequent section, the author intends to show that, while semantics can augment the cooperation processes, the cooperation processes themselves have the potential and capability to enable semantics along the DEDES components.

It has already been shown that the main functions of the cooperation processes are to facilitate, support and perform the exchanges of information structures within and among the DEDES subsystems. However, the semantics are qualitative characteristics of usually complex information structures (as expressed in chapter two), extensionally articulated in terms of relationships (as shown in chapter three). Therefore, given their significant role concerning the information structures, it is natural to assume that the cooperation processes are also the ones that can enable semantics within and among the DEDES subsystems.

It has also been shown that structurally, the DEDES consists of three subsystems with each subsystem consisting of hierarchies of subsystems. Taking into consideration that the DEDES behavior (or functioning) is facilitated by cooperation processes and that the behavior is the result of the system structures functioning as a whole, then, functionally, the DEDES is made of an arbitrary number of ad-hoc or predetermined

functional entities with a life span conditioned by the specific function(s) that has(ve) to be performed. Basically, any function handled by the proposed architecture is a cooperation process, a part of it, or a combination of them. Hence, a functional entity is a temporal functional cell consisting of system's structural components engaged in a dialog (i.e. a cooperation process).

A generic scenario involving a functional entity can be characterized as follows. Suppose that two structural components (called c1 and c2) need to exchange some information structures. This desired situation requires setting up a dialog between the two components, which further translates to the need for a cooperation process. The underlining functional entity will therefore, consist of the structural components c1 and c2, plus the cooperation relationships (called 1coop2) established between c1 and c2.

Specific to the engineering design process, the DEDS structures are highly heterogeneous and, therefore, it is fairly probable that the components c1 and c2 will need different local representations for the information they require for a proper functioning. However, for a dialog to be possible some information exchanges are required. Moreover, the information that is exchanged has to have sense, i.e. meaning, at the both c1 and c2 component levels. A solution would be that each component would 'understand' or at least 'know how to codify' the information required by the other component and then a cooperation process would be employed to effectively execute the dialog. Alas, this simplistic view of the cooperation process results either in an implementation of monolithic systems or in a support for quite simple and too plain information structures that can be exchanged.

Nevertheless, a proper use of the cooperation process (i.e. 1coop2) should take advantage of its possibility to (at list conceptually) reach almost anything that qualifies as an information structure. Suppose that, a commonly agreed representation of information is reached at the system level. Then, the collaboration constituent of the 1coop2 will find the most appropriate representation that can be shared by both component c1 and component c1. Further, the coordination constituent of 1coop2 will identify the activities for transformation or mapping the information from the representation local to c1, to the DEDS shared representation and further, to the representation local to c2 and vice versa if necessary. Next, the co-location constituent of 1coop2 will discover the channel between c1 and c2 and, finally, the

communication constituent will effectively perform the exchange of information structures between c1 and c2 components. Therefore, the 1coop2 process will be capable of enabling the dialog, i.e. to preserve the semantics of local information structures, between the c1 and c2 components without the need for c1 and c2 to be aware of each other's local information logic.

In summary, in case that some prerequisites are met (e.g. system level information structures can and are represented in a common agreed form by the DEEDS components), the cooperation process can converse semantics among the DEEDS components. Moreover the components are not required to sacrifice their otherwise beneficial autonomy for the sake of enabling necessary dialogs.

4.3.3 Ontologies and Software Agents – the Technological Enablers

It has been demonstrated that the cooperation processes are capable of preserving and moving semantics between the DEEDS components. Therefore, it is possible for DEEDS components to exchange meaningful information structures, thus making possible inter-component complex conversations. This situation can only enrich the conversation capabilities since the expressiveness of the dialog is not limited to simple strings of data.

However, with few exceptions, the DEEDS structural components will generally not be aware of each other, and will not even be aware of the existence of cooperation processes. For example, the components from the Human System can establish conversations with each other or with other components (with the condition that they know how to operate them), unlike the case with the non-human components, which, if not explicitly (or by nature) enabled for conversations, they will not be able to actively pursue a dialog. Thus, the need for interfaces that are able to enable the various DEEDS components for dialog. For the present thesis, the author confines himself to characterizing the computer-mediated interfaces (i.e. software based interfaces).

The author has identified two technologies, i.e. ontologies and software agents (see chapter three), which, working together, can support the implementation of computer-mediated interfaces for enabling the inter-component dialog in a distributed environment (such as the environment underlined by the DEEDS).

Resulting from the findings to date, the author considers that there are two critical issues concerning the use of computer-based technologies for supporting a software implementation of the cooperation process. They are as follows:

1. How can the system level inherently distributed information structures (i.e. engineering design information) be represented in a common agreed form by the DEDS components?
2. How exactly are the DEDS structural components (for example c1 and c2 from the scenario above) capable of (i) recognizing that a dialog is necessary, (ii) initiating a dialog, and (iii) carrying out the dialog?

While both the first and the second issues are dealt with by the unitary employment of ontologies and software agents technologies by implementing a semantic enabled environment (Gruber 1992; Genesereth and Ketchpel 1994; Nwana 1996; Gomez-Perez 1998; Guarino 1998; Jennings, Sycara et al. 1998; Wooldridge 1999; Hendler, Berners-Lee et al. 2002), the ontologies will especially be used for the former and the software agents for the later (this state of affairs it is also used for clarity reasons).

Question 1: How can the system level inherently distributed information structures (i.e. engineering design information) be represented in a common/shared agreed form by the DEDS components?

Answer 1: By developing various ontologies for various behaviours and different ontologies for different levels of details, a semantically enabled computational environment is shaped. Moreover, this environment follows the different levels of granularity required by the DEDS components and indeed, by the different ramifications of the cooperation process. However, as mentioned above, the ontologies enable the sharing of the DEDS information structures for *applications* (especially software applications) and not specifically for the DEDS components.

This state of affairs introduces the second issue, for the reason that, actually, for *implementing* computer-based cooperation support, it is not necessary for the DEDS information structures to be shared directly among the DEDS components. It is enough to obtain agreed representations among applications, i.e. software agents, which will act as an interfaces of DEDS components to the cooperation processes.

Question 2: How exactly are the DEDS structural components (for example c1 and c2 from the scenario above) capable of (i) recognizing that a dialog is necessary, (ii) initiating a dialog, and (iii) carrying out the dialog?

Answer 2: The DEDS structural components can be interfaced or computationally modeled by software agents, which are capable (i) of recognizing a dialog triggering change in environment, (ii) based on which to request a conversation (by means of an ACL and common agreed ontology) and (iii) can start cooperating with other agents (based on agreed upon ontologies).

In conclusion, using collections of ontologies as manifestations of a shared understanding of the DEDS domain that is agreed between a number of software agents (based on (Uschold 1998)), a cooperation enabled semantic environment can be obtained. Moreover, the environment has its information structures represented in a *reusable, shared and formal* way (by means of ontologies), it consists of the *integrative mechanisms* (software agents) that interface DEDS components, and it is *controlled* by non-human *mechanisms* (cooperating agents enabled by ontologies).

4.4 Proposed Framework for Enabling Semantics within the DEDS

In the subsequent sections, the author will propose an architecture that will describe the framework for implementing a solution agent-based software system to the identified information related problems within the DEDS. The anticipated architecture intends to portray a *suitable* and *viable* framework for enabling semantics for the cooperation processes within a distributed design environment. *Suitable* refers to the fact that, with minimal changes within the DEDS, the underlining software system should overcome the critical information concerned impediments that impair the functionality of the engineering design process *in a distributed environment*. *Viable* means that the system will be able to bear evolution and change without functionality losses. An important point is that this thesis, and therefore the proposed architecture, does not intend to change distributed design in what it is or in the way is carried out. It is planned to provide a information-focused support mechanism that improves the positive outcomes of the distributed design and that eliminates or minimizes its negative aspects, as identified in chapter two. Therefore, the components of a *real* distributed design environment are viewed as *given* components of the DEDS. The human participants (in the design process), the design methodologies and tools

(including CAD tools) and the design information structures are not intended to be replaced. However the intention is to better integrate them into the distributed engineering design environment by means of autonomous (i.e. not mediated or performed by humans) cooperation processes.

4.4.1. The Architectural Framework

Given that the architectural specifications are the direct result of the findings from the previous sections and of the author's view of the DEDES, and based on the available semantic enabler technologies, the author proposes the architectural framework model depicted in figure 4.5. The proposed framework describes the author's description or characterization of the space in which an eventual software implementation (for supporting semantics within the DEDES) will act.

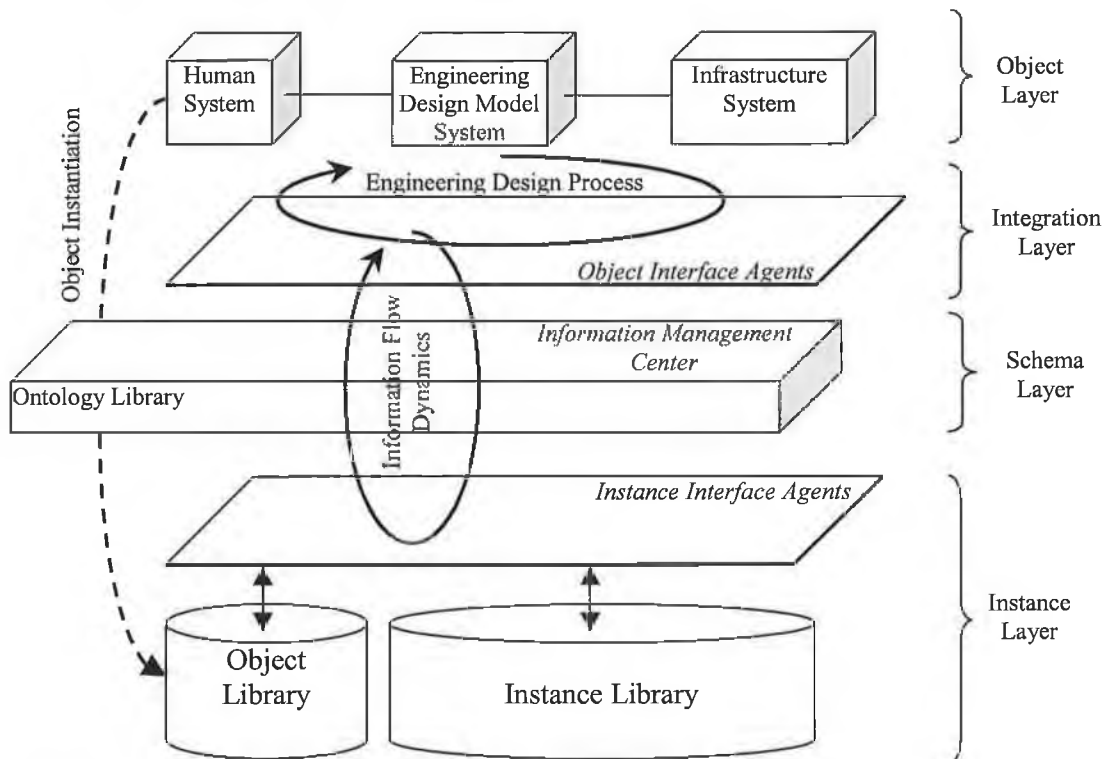


Figure 4.5 The proposed DEDES Architectural Framework

The proposed architectural framework is structurally and functionally built around two main information flows:

1. The *Engineering Design Process* viewed as an information transformation activity (as characterized in chapter two, especially in section 2.2.1)

2. The *Information Flow Dynamics* viewed as an inter and intra DEDS components relationships enabler (as characterized in chapter two, especially in section 2.3.3)

Within the *Engineering Design Process*, the unstructured, informal and generally highly abstract information structures (i.e. requirements and constraints) are continuously organized, formalized and detailed with respect to each phase of the engineering design process, until a suitable structure (or global optimum) is achieved. Of course, the progress is not linear but is characterized by a circular ‘tree-like’ configuration, consisting of a series of divergent dead-ends and returns, until a convergence towards the global optimum is reached. Accordingly, the different stages of the design information are captured in the form of working, released (including version control) or obsolete structures.

In order to support this transformation of information process, external information is required and produced for later stages (e.g. document management activities, i.e. check-in, check-out, obsolete, release and version control). This kind of support is provided by the *Information Flow Dynamics*, which ‘backs-up’ the Engineering Design Process by providing its building blocks and the informative ‘know-how’. Moreover, it mediates the dialog among the generally heterogeneous performers (i.e. the various DEDS components) of the engineering design activity. Therefore, the Information Flow Dynamics manages and deals with the information needs of the Engineering Design Process, thus vertically supporting the horizontal unfolding of the engineering design process.

For intelligibility reasons, the author will further describe and characterize the architectural framework from two perspectives, identified as the structural perspective and the functional perspective.

From a structural perspective (guided by the results of the systems thinking introduced in this chapter), the proposed architecture consists of four layers as follows:

1. The *Object Layer* consists of the distributed engineering design performers or actors (e.g. designers, CAD tools) together with their local typically implicit information structures (e.g. data, information and knowledge a designer has, data files used by a CAD tool);

2. The *Integration Layer* consists of specific mechanisms that create and support information mediated connections between components from Object Layer and the rest of the architecture.
3. The *Schema Layer* consists of patterns and logics that define the forms in which the information structures are represented within the architecture. It also consists of the cooperation processes regulation and control mechanisms.
4. The *Instance Layer* consists of the architecture information structures instances together with the mechanisms that serve them.

From a functional perspective (guided by the appropriate technologies as identified and characterized in chapter three), the proposed architecture can be described through two planes as follows:

1. The *Ontological Plane* specifies the hierarchy of ontologies that defines the concepts, the relations and the inference rules that compose the machine-enabled framework in which DEDS information resources are circulated and stored.
2. The *Agent System Plane* specifies the types and behaviors of the software agents required for the DEDS components integration, user interfaces, reasoning and system control.

4.4.2. The Architectural Layers

The architectural layers describe the structural context that characterizes the computational environment required for implementing cooperation processes among interfaced DEDS components.

Object Layer

The *Object Layer* consists of the three categories (as identified in chapter two) of ‘physical’ or ‘real’ objects that are engaged in the distributed engineering design process (i.e. the components of the Human, Infrastructure, and Engineering Design Model subsystems). Actually, this layer subsumes the distributed design organization itself made of human designers, integrated circuits and coaxial cable computer networks, various software designing tools, paper or electronic based design methods and methodologies, specific design information, and so on. The engineering design process is effectively performed at this ‘physical/reality’ layer level. In order to

facilitate the architecture's performance, the material, structural and functional Babel of physical objects need to be syntactically homogenized.

An Object Instantiation process represents the first step in accomplishing this necessity. As a prerequisite, the formal class hierarchies need to be identified and explicitly described in the Ontology Library. Then, for every object, the following needs to be accomplished:

- Identify the most specific class to which a particular object belongs
- Instantiate (i.e. represent) the object filling the information required by the slots that characterize the respective class
- Store the object's representation in the Object Library

Finally, the Object Layer is conceptually modeled in the Schema Layer and extensionally and formally represented in terms of information structures in the Object Library, since this is the place where engineering design activity is actually performed. However, it cannot be disregarded (in the sense that its ontological schemas and the object instances can replace it) when considering implementing semantic support for DEDES. This is mostly for the reason that the Object Layer still remains the driving layer of the architecture since it is here where the distributed engineering design actually occurs.

Integration Layer

The *Integration Layer* defines and employs the interfaces between the Object Layer and the rest of the architecture. Therefore this is the place where the integrative mechanisms (as identified in the requirements analysis, section 4.3.1) carry out the 'plugging-in' of the DEDES components (i.e. the objects from the Object Layer) in a syntactically and semantically information enabled workspace. Whilst this function (i.e. integration) is supported by both ontologies and agent-based software technologies, the main performers are a set of Object Interface Agents, which will be described later.

The Integration and Object layers define the space where the information flows of the Engineering Design Process progress.

Schema Layer

The *Schema Layer* is responsible for formally defining the patterns of representation of all the information structures that are circulated within the Information Flow Dynamics stream, by the means of the Ontology Library. It is also the place of the main structure that controls and regulates the architecture's behavior, i.e. a Multi-Agent System called Information Management Center (IMC). While the inter-layer cooperation processes may not need its assistance, the inter-layer cooperation processes are mediated by the IMC. Further clarifications are detailed later.

Instance Layer

The *Instance Layer* is the place where the system information structures that obey the representation rules as defined by the Ontology Library are stored. For their local needs (e.g. security, soundness, read, write, query) a set of software agents, called the Instance Interface Agents is responsible. The Object Library contains the formal representations of the objects from the Object Layer as resulted after the application of the Object Instantiation process. The Instance Library, for a change, stores their information needs.

4.4.3. The Architectural Planes

The Architectural Planes, consisting of the Ontological Plane and the Agent System Plane (see figure 4.6 for a graphical representation), determine the characteristics and the scope of the technological enablers, i.e. ontologies embodied by an Ontology Library and agent-based systems embodied by a group of software agents and multi-agent systems.

The *Ontological Plane* specifies the hierarchy of ontologies (i.e. the Ontology Library that reflects the structural and functional granularity of the DEDS) defining the concepts, relations and inference rules that compose the machine-enabled framework in which the system's information resources are circulated and stored. It also includes engineering knowledge instantiated (i.e. in Instance Bases) according to the rules specified by the Ontology Library. The Ontological Plane intends to provide a homogeneous schema for representing the distributed engineering design domain. The *Agent System Plane* specifies the types and behaviours of the software agents required for the system's components integration, user interfaces, reasoning and system control. It intends to facilitate the access, retrieval, exchange and presentation of data,

information and knowledge to distributed design teams through agent systems such as the Object Interface Agents, the Instance Interface Agents and the Information Management Centre.

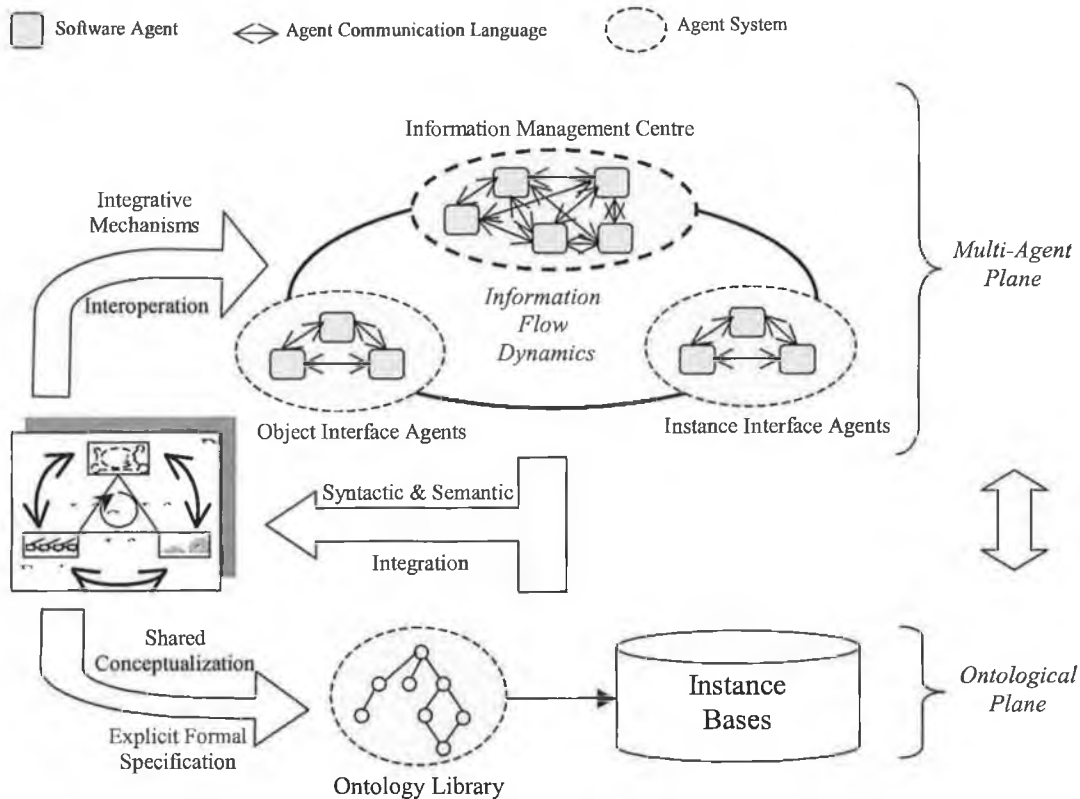


Figure 4.6 The bi-plane model view of the DEDES architectural framework

The Ontological Plane

If one can imagine an analogy, where the DEDES is the world, then the three subsystems (i.e. Human, Infrastructure and Engineering Design Model) represent the continents. Each continent is structured into countries, with each country consisting of regions, counties, cities, towns and villages. The village-town-city- ... - country-continent relations are the subsystem's structural relations. It is also possible to have countries that span two or more continents. In this case inter-subsystems structural relationships are involved. The roads that link different settlements are the functional relationships. These may be regional, national, continental, or even intercontinental. Furthermore, as the geographical maps represent the world with all its necessary detail in a bid to visually manage the large-scale complexity of the Earth itself, the Ontology Library provides a homogeneous schema for representing the distributed engineering design domain (its concepts and relationships) in a bid to manage its cognitive complexity and diversity.

Concluding the analogy, if the DEDES is the globe, then the Architecture is the globe's map and the Ontology Library is the agreed map key.

As identified in section 4.3.3, the hierarchy of ontologies (i.e. the Ontology Library) needs to reflect the structural and functional granularity of the DEDES. Its main purpose is to represent (i.e. reorganize), in a commonly agreed syntactic and semantic information makeup, the structural and functional components of the DEDES.

Therefore, where an information exchange process is concerned, the hierarchy of ontologies should reflect as close as possible the degree of granularity (neither too coarse, nor too fine-grained) required by the components that form the cooperation cell. Thus, there are two kinds of ontologies, i.e.

1. *structural ontologies* and
2. *functional ontologies*,

both of which specialize a high-level ontology that describes the most general and fundamental concepts and relations of the DEDES and notwithstanding its architectural role, represents the system introductory card for the outside world.

The *structural ontologies* conceptualize and formally describe the structures that form the DEDES (i.e. all the three subsystems and their components, together with their characteristics). They form the schemas used by the Object Instantiation process for translating the 'real' structures from the Object Layer into the information structures from the Instance Layer.

A top-level structural ontology with an informative kind of role reflects the structure and the relationships of the structural ontologies hierarchy. At the next level, another three hierarchies of ontologies reflect each of the three DEDES subsystems, as follows:

1. The Infrastructure hierarchy conceptualizes and formally describes the hardware and the software components located within the DEDES. It provides a formal way of describing computers in terms of their memory and storage capacity, operating system, network identifier, etcetera. It also formally represents the applications used by the designers. This later point is of main importance for the Information Translation Agents (as it defines how to mine for applications information and vice versa, how to feed applications with information).

2. The Human hierarchy describes the designers and all other human actors involved in the distributed design process based on their agreed profile. This hierarchy enables and supports the performance of the User Interface System Agent.
3. The Engineering Design Model hierarchy models the engineering design process itself. It contains the general description of any engineering design process, its necessary stages and the appropriate methods and methodologies. Furthermore, each engineering design phase (for example Requirement Definition, Functional Requirements, General Design and Detailed Design) is encoded in a dedicated ontology that will also include life-cycle information, (e.g. Raw Material, Manufacture, Use and End of Life) with the purpose of integrating the engineering design activity in the larger product realization process.

The *functional ontologies* formally specify the concepts and relations needed by the cooperation processes that take place among the different DEDS components or software agents. They also include mapping ontologies that relate and equalize concepts from different structural and/or functional ontologies (e.g. ID from ontology X1 is the same with Identificaton_number from ontology X2).

Similar to the structural ontologies, at the top level of the functional ontologies resides an ontology that reflects the structure and the relationships of the hierarchy. The next levels are inhabited by ontologies with different degrees of specializations that describe the context of the collaboration process. This (i.e. the context) includes parameters such as:

- The types of interaction (e.g. dialog, monolog, informative);
- The kinds of messages (e.g. request information, save information, check consistency);
- The structures of information (e.g. product features, user profile, tool description);

Finally, the DEDS Ontology Library needs to be an open structure, i.e. it can be modified, extended, or even decreased if necessary. This feature of openness is necessary because the distributed engineering design domain is a dynamic,

changeable environment and, therefore the ontologies (as conceptual reflections of the domain) should be capable of reflecting these characteristics of the DEDES.

The Agent System Plane

The interoperation of multiple agents within the DEDES is achieved through an agent communication language (or ACL) and a common shared Ontology Library. The ontology library creates a shared (i.e. design semantics are the same for all agents that commit to the ontology library), formal (i.e. design semantics are agent enabled) understanding of the design domain. This machine-enabled pool of data, information and knowledge represents the environment in which multiple agents act within the DEDES.

The author considers that the agent-based system plane consists of software agents that are characterized by autonomy, reactivity, pro-activeness, cooperation and temporal continuity. The following two kinds (by the function or role performed within the DEDES) of software agents:

1. *Operational Agents* perform operational roles and are responsible for the functions that the architecture performs.
2. *Regulation and Control Agents* form a Multi-Agent System called Information Management Center that is in charge of the architecture's behavior.

The *Operational Agents* are information-based sensors and effectors. They are able to react to informative stimuli generated by the component they attend, which they can also activate (i.e. the specific component) when necessary. The environment in which these agents act has the following properties (based on (Russell and Norvig 2003) classification on environment properties):

- Accessibility – the agents can obtain complete and accurate information about their environment because they act locally (at Object Layer or Instance Layer level) and generally interface well with described components (by the means of the Ontology Library);
- Deterministic – any action performed by an agent has a single guaranteed effect (e.g. capture information, store, retrieve, browse, search and so on);

- Dynamic –the environment is changed not only through agent actions (this can raise some problems for designing agents);

Since the *Operational Agents* are essentially the integrative mechanisms, the following kinds have been identified:

- The *Instance Interface Agents* interface and manage the data, information and knowledge instances that are ontology-compliant, and are stored on the system's computers. They perform specific functions such as storing, retrieving, consistency checking, revision control and maintaining. All the other agents have to summon these agents for their information needs (e.g. retrieving or storing information).
- The *Object Interface Agents* interface and manage the connection of the physical objects to the system's information and consist of the following agents:
 - *Information Translation Agents* – translate or re-represent information from an internal format as specified by the ontologies into an external format as required by the specific component that is served and/or vice-versa. They implement the interface with the software tools used during the design process, being activated by application specific events (e.g. save, load). Each agent is tailored for a specific application (e.g. SolidWorks) so it 'knows' where and how the application keeps its information structures. They also know how to translate the application representation of information into internal representation, as defined by the ontology library. Once activated, they initiate a cooperation process with specific Instance Interface Agents for either storing or retrieving information
 - *User Interface Agents* – are actually a special type of information translation agents, forming an agent-based system that assists the humans in using and working with the system, being capable of memorizing user patterns, behavior and operational needs. They are autonomous agents that deal with any user specific aspect within the DEDES, e.g. collaboration with other DEDES participants by enabling a virtual collaborative environment (e.g. chat, audio and video conferences, file sharing, whiteboard-ing),

captures of and requests for information. They are tailored (and ideally able to model themselves through learning) according to specific user needs and preferences, and act autonomously in the distributed design environment.

For the coordination and for the good performance of the *Operational Agents*, the *Regulation and Control Agents*, or the Information Management Center plays a critical role.

The *Information Management Center* forms a Cooperative Multi-Agent System responsible for the management and administration of the entire architecture. It represents a distributed mechanism that supervises system's behavior and makes sure that the system performs within normal parameters. Feedback and feed-forward mechanisms continuously inform specific control centers about the current state of different parts of the system, so proper actions can be taken in case of functional disturbances. The feedback and feed-forward mechanisms are in fact sensors that capture the current states (e.g. sleep, activate, wait, action and so on) of all the agents from the system. They also inform about the states of the different active cooperation processes (e.g. running, successfully finished, unsuccessfully finished with the error code xxx, pending, bottleneck). Having direct access to the Ontology Library, the Information Management Center also intervenes/interferes in the inter-layer agent cooperation processes, by identifying the most specific ontology that can be used in the desired cooperation process. In addition, it identifies and names the proper Instance Interface Agent required by an Object Interface Agent. Moreover, the Information Management Center also implements the securities policies necessary in the organization, based on the digital signature that every agent in the system has to have. To summarize, the Information Management Center functionally connects the different layers of the Architecture based on access rights and actively supervises the performance of the system's agents.

4.5 Conclusions

Applying Systems Thinking, as an holistic technique for gaining knowledge and understanding of the distributed engineering design organization, it has been argued

that the DEDS centripetal¹ force consists of information structures and cooperation processes dynamics. The information structures of the system are the means by which the interactions among the heterogeneous structures (of Human, Infrastructure and Engineering Design Model Systems) are possible. The cooperation processes, in turn, form (by means of communication processes), enable (by means of co-location processes), regulate (by means of coordination processes) and support (by means of collaboration processes) these information-mediated interactions or dialogs.

Also, the need to provide computational semantic support in order to improve the performance of the engineering in distributed environments has driven the research to the conclusion that the DEDS cooperation processes qualify for the role of semantic integration. Moreover, from a technological point of view, using collection of ontologies as manifestations of a shared understanding of the DEDS domain that is agreed between a number of software agents, a cooperation enabled semantic environment can be obtained.

Such an environment has been architecturally modelled, dimensionally along two main information flows (i.e. Engineering Design Process and Information Flows Dynamics), structurally along four layers (i.e. Object, Integration, Schema and Instance Layers), and functionally along two planes (i.e. Ontological and Agent System Planes).

The next chapter will provide an implementation-al validation of the computational context or framework described by the proposed architecture.

¹ Centripetal force – an force acting on a body causing it to move towards a centre Pollard, E. and H. Liebeck (2000). *The Oxford Paperback Dictionary*. New York, Oxford University Press Inc.

Chapter 5

An Instantiation of the Proposed Architectural Framework

5.1 Introduction

5.2 An Ontology-based Software Agent (OSA) System

5.3 An OSA Prototype

5.4 Conclusions

5.1 Introduction

The investigations carried out in the preceding chapter concluded with the description of an architectural framework (see chapter four, section 4.4) that bounds the author's view on the representation of the computational space describing any software system that supports machine-enabled DEDS cooperation processes. This architectural framework, supported by semantic technological enablers, i.e. ontologies and software agents, provides the 'skeleton' for implementing software systems in order to facilitate the access of DEDS components to however complex information structures.

In this chapter, a conceptual instantiation of such a software system (called OSA – **O**ntology-based **S**oftware **A**gent system) will be introduced with the purpose of testing and validating the results of the current research, i.e. the proposed architectural framework. Certainly, the anticipated system is not the only one possible to be modelled according to the architectural model, since the proposed architecture acts as a frame and not as software specifications. However, given the articulated reason, the author contends that a single functional instantiation of the OSA system is sufficient in order to accomplish the needs of this stage of the research.

This proposed OSA model will also provide the basis for the implementation of an operational prototype of the OSA system. The author argues that, being possible to implement a software prototype following the specifications of OSA system, which, in turn, obeys the structural and functional specifications of the proposed architectural framework, results in the validation of the architecture.

5.2 An Ontology-based Software Agent (OSA) System

The overall goal of the OSA system is to fulfil the DEDS requirements as identified in the previous chapters:

1. Reusable, shared and formal structures of information
2. Integrative mechanisms
3. Non-human software-based control mechanisms.

This typology of requirements articulates the need for (i) the system information structures to be represented in such a way that their semantics are preserved independently of the local informational representation logics and, moreover, to allow the translations among system level and local levels representations; (ii) DEDS object interfacing or 'plugging-in' mechanisms for enabling inter-object conversations or dialogs; and (iii) regulation and control mechanisms capable of functioning without human mediation.

The OSA system intends to do the followings:

- To improve the designer's access (qualitative=semantics and quantitative=about the right amount of information to be brought forward) to the *distributed* design information
- To do this by instantiating the architectural framework and, thus
- To validate the architectural framework

In summary, the proposed architectural framework is the cognitive result of this research process which synthesizes the results of the investigations carried out in the preceding chapters and recommends a both *functional* and *structural* solution context to the above requirements, which the author contends will improve the manner in which the distributed engineering design process is carried out.

From the *functional* perspective, the architecture proposes the use of two kinds of ontologies (i.e. *structural* and *functional* ontologies) grouped in an Ontology Library and two kinds of software agents (*operational* and *regulation and control* agents). In combination, these two technologies can generate a computational system capable of operating at knowledge-level and therefore, suitable for supporting complex DEDS behaviors.

From the *structural* angle, the architecture is divided into four autonomous parts or layers, with each layer being responsible for specific functions and therefore, being served by specific functional architectural components, as follows:

1. The *Object Layer* is a special layer that subsumes the actual performers of the distributed engineering design process. However, even if it does not have dedicated functional components, it actually represents the rationale of all the other layers.
2. The *Integration Layer* defines the place where a specific type of operational agent acts, i.e. the *Object Interface Agent* that consists of *Information Translation Agent* and *User Interface Agent*.
3. The *Schema Layer* represents the brain of any architectural implementation since it identifies the place where the Ontology Library resides and the regulation and control agents (i.e. the Cooperation Management Center) act.
4. The *Instance Layer* is the place where any OSA system will store its main working material (i.e. information structures) in the form of distributed Instance Bases, which are interfaced by a second type of operational agent, called *Instance Interface Agent*.

The holistic summation of these layers obtains a cooperation enabled semantic environment, thus further resulting in computer assisted DEDS behavior (i.e. the distributed engineering design process) at a knowledge level.

5.2.1 The Proposed OSA System

The OSA system facilitates the access of interested DEDS parties (e.g. design engineers, Materials department employees) to up-to-date design information. The proposed system deals with the following two issues (that were identified in section 2.2.4 and which actually have driven this research work) concerning the access to information structures:

1. *The qualitative issue*: the OSA system intends to bring to its users the right information with minimum expertise required.
2. *The quantitative issue*: the OSA system intends to limit the amount of information brought forward so as to avoid possible information overloads.

At the core of the OSA system resides the concept of *service*. Based on dictionary definitions, the *notion* of “service” is understood to be as follows (Pollard and Liebeck 2000):

- “A system or arrangement that performs work for customers or supplies public needs”
- “Use, assistance; a helpful or beneficial act”
- “Provision of help for customers or clients”

Thus, the role of the OSA system is to assist by supplying services for its clients or users, i.e. DEDS components that are integrated into the architecture by means of Operational Agents. Actually, the OSA services are conceptualizations of DEDS behaviors, e.g. search a specific instance base, browse the structure of product, initiate a chat session.

Formally represented in a dedicated ontology, the *concept* of service can have the structure presented in figure 5.1.

Each OSA service is made up of the following constituents:

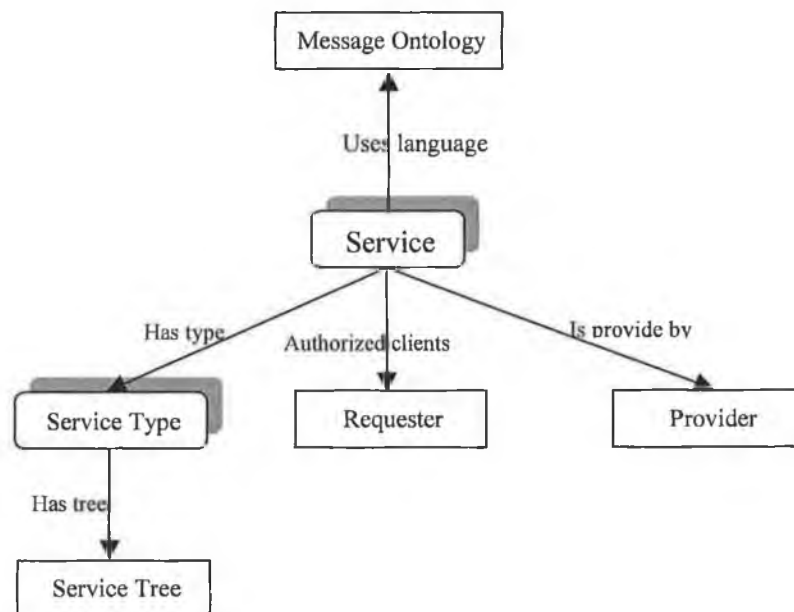


Figure 5.1 An ontological model of the *service* concept

1. *Type of service* is an ontological conceptualization of a DEDS behavior. In this particular case the author proposes the use of three structures for formalizing the DEDS services. For example, in the simple case of a service of type *browse service* (that is implemented in the proposed prototype) the tree structure consists of the root *browse* and the four leafs (material, product, fastener, resources). Thus four atomic services are made available, i.e.. *browse_material*, *browse_product*, *browse_fastener* and *browse_resource*. The main reason behind this logic is to be able to design agents capable of serving sub-trees of services (in opposition to designing agents for each atomic service).
2. The *Requester* (or the receiver of the service), an Object Agent, needs and requires specific services on behalf of its user. The services available to the Requester depend on the actual DEDS object the agent interfaces, i.e. the Object Profile that is predefined and stored in the Object Library.
3. The *Provider* (or the sender of the service), a CMC agent or an Instance Interface Agent, performs the specific operations defined with the particular type of service.
4. The *Message Ontology*, a functional ontology, names the ontology needed by the Requester and the Provider in order to communicate. If this ontology is not

a standard FIPA¹ ontology (which is known by any agent), then is the most detailed functional ontology to which both parties commit.

The proposed OSA system (see figure 5.2) fulfills the needs of the distributed engineering design process performers (e.g. human designers) for appropriate design information, by supporting an improved cooperation process between the interested parties (i.e. the DEDS object and the design information structures).

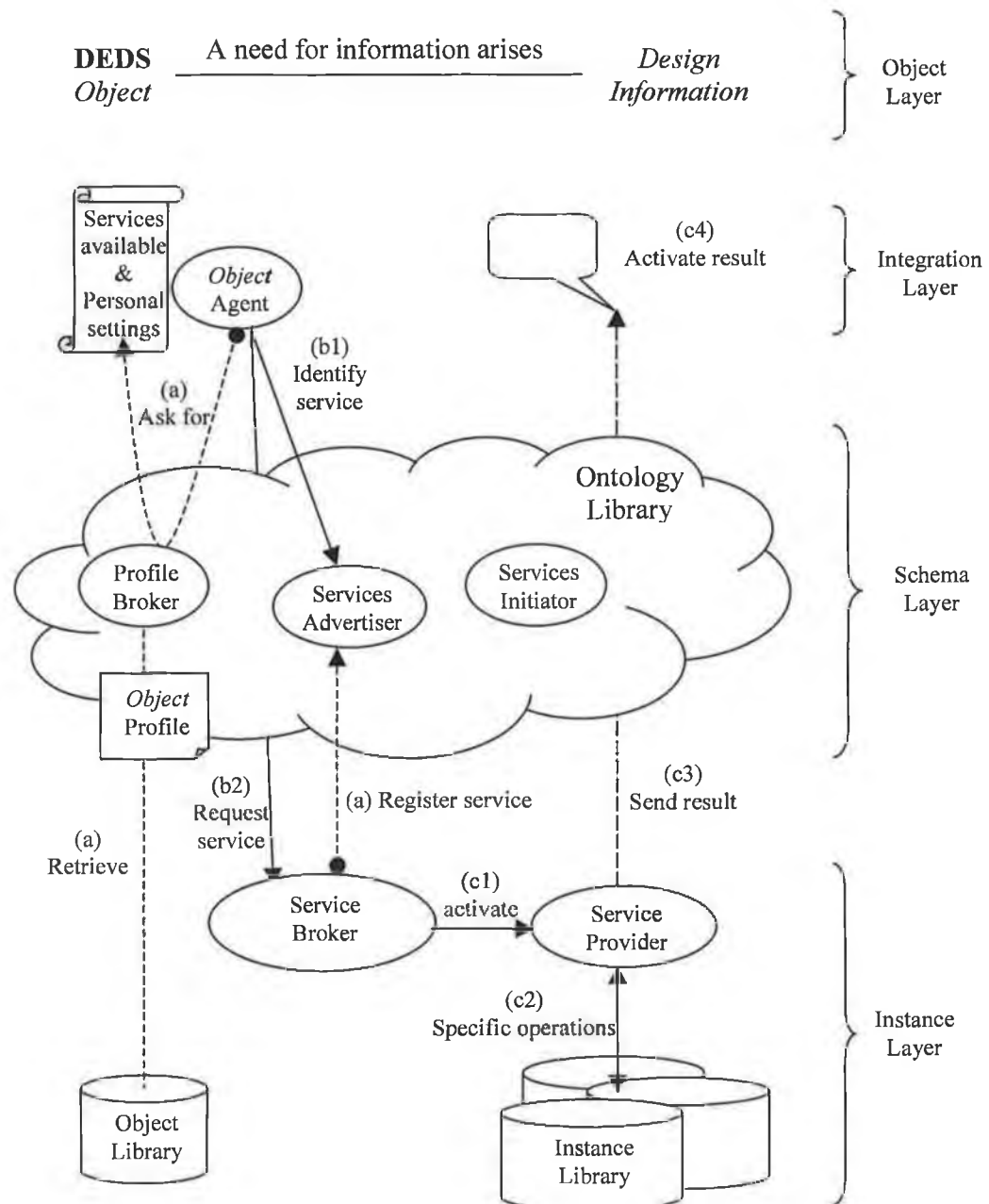


Figure 5.2 The proposed OSA system to support the access to design information.

¹ FIPA – Foundation for Intelligent Physical Agents www.fipa.org

In order to accomplish the need for *design information* of a *DEDS object*, the OSA system proposes the following phases:

- (a) This phase consists of operations performed at the activation of the OSA agents, i.e. retrieving of the object profile (in the case of an Object Agent) and service registration (in the case of a Service Broker).
- (b) At this phase the service provider is identified (sub-phase b1), and the service is requested (sub-phase b2).
- (c) This phase subsumes all the operations performed for accomplishing the required service, i.e. the activation of the specific service performer (sub-phase c1), which actually implements the service (sub-phase c2), sends the results to the requester sub-phase c3) and activates the results (sub-phase c4, for example the activation of a GUI for presenting the design information to the DEDS object).

The *Object Layer* of the OSA system is the place of the DEDS components (i.e. Human, Infrastructure and Engineering Design Model subsystems). At this layer the need for design information occurs. The purpose of all the other proposed layers is to fulfill the *DEDS Object* needs for *design information* structures, by means of performing services.

The *Schema Layer* of the OSA system contains three special kinds of services that do not implement DEDS behaviors. However, they are used by OSA system to regulate and control the behaviors of all the other agents. The agents that implement these behaviors are as follows:

1. The *Profile Broker* agent provides a special kind of service required by all the Object Agents at their activation, i.e. it retrieves the Object Profile based on the credentials of the particular agent that asked for it. The Object Profile describes the services available to the respective agent and may also contain (in the case of User Interface Agent) specific personal settings (e.g. service history, CV, display specific requirements).
2. The *Service Advertiser* agent plays the role of Yellow Pages for the OSA agents. The providers of services publicize themselves within it, while the requesters of services identify the required provider(s) from it.
3. The *Service Initiator* agent captures messages that are broadcasted within the system and, when a specific (predefined) change occurs (e.g. a write operation

in the material base or a change of a subassembly in the detailed design base), it announces the agents that may be interested in it (e.g. the User Interface Agent of a person from the Materials Department or of a design engineer)

The *Integration Layer* consists of Object Agents or Requesters of services. When activated, they request their up-to-date profile from the Profile Broker agent. In the case that their personal profile has been changed while active, the Profile Broker agent (informed about the update by the Services Initiator agent(s)) will proactively transmit the new profile to the concerned party. When (e.g. mouse click, some change in the state of the system) an Object Agent needs a service, it will consult the Services Advertiser agent(s) about the possible provider. If the proper provider(s) is(are) identified, that Object Agent will initiate a Service Message-based conversation with it.

The *Instance Layer* is the place where the Providers agents act. It also includes the Object Bases (that store the personal profiles of the Object Agents) and the Instance Bases (i.e. design information stored accordingly to the definitions from the structural ontologies from the Ontology Library). When becoming active, the Provider agents register themselves with the service(s) they supply. A Provider supplies a category of services. That means that each local Provider (called *Service Broker*) supplies the same kind of service (e.g. search, browse, query) for all local Instance Bases. Since the OSA Instance Bases (e.g. Materials Instance Base, Product Structure Instance Base) can be stored on more than one computer (after all, the DEDS resources are distributed), the same Service Broker agent can be found on different computers. When a certain service is required, the Service Broker will activate the proper Service Provider, which knows what to do in order to accomplish the tasks required on the particular Instance Base by the required service. Therefore, each set of local Instance Bases are interfaced by a Service Broker who controls a set of Service Provider agents, which, in turn, actually perform the services specific to each type of Instance Base.

5.3 An OSA Prototype

The main and sole purpose of the OSA prototype is to demonstrate that the OSA system is functional. Given the vastness of the (distributed) engineering design domain, only a small set of behaviors has been implemented. However, as shown in chapter four (see section 4.2), the main information flow patterns of the DEDS concerning this research work have a fractal nature, i.e. the overall cooperation process repeats itself at the different levels of

granularity within the system. Therefore, the author argues that, even the implementation of a small subset of cooperation-mediated behaviors can give a measure of the validity of the OSA system

5.3.1 Prototype Characterization

The proposed prototype consists of instantiations and implementations of the kinds of ontologies and software agents characterized in the architectural framework model (figure 4.5) and more specifically identified by the OSA model (figure 5.2). From the available tools and technologies, Protégée 2000 (<http://protege.stanford.edu/>) and JADE (<http://jade.tilab.com/>, (Bellifemine, Poggi et al. 1999; Caire 2002; Bellifemine, Caire et al. 2003; Bellifemine, Caire et al. 2003; Bellifemine, Caire et al. 2003)) have been used for the Prototype implementation.

Protégée 2000 “is an integrated software tool used by system developers and domain experts to develop *knowledge-based systems*. Applications developed with Protégé-2000 are used in problem-solving and decision-making in a particular *domain*” (http://protege.stanford.edu/doc/users_guide/index.html). Protégée 2000 has been used as an ontology editor and as a knowledge base editor.

Java Agent Development Framework (JADE) is an Open Source project defined as “an enabling technology, a *middle-ware* for the development and run-time execution of *peer-to-peer* applications which are based on the *agents* paradigm and which can seamless work and interoperate both in wired and wireless environment” (Bellifemine, Caire et al. 2003). JADE provides the necessary libraries for developing software agents and some basic services necessary to distributed peer-to-peer applications.

In order to keep the OSA system flexible and adaptable, the author found that the agents should have the ‘minimum’ necessary intimate knowledge (or knowledge implemented in their algorithms at development phase) about the *overall* environment in which they act. An example of such kind of knowledge is the fact that the Object Agents ‘know’ about the existence of the Directory Facilitator agent and also ‘know’ about the fundamental role of the service provided by the Profile Broker agent. However, in order to be useful and efficient, the agents need to ‘know’ their *local* environment. For example, (i) an Object Agent ‘knows’ that its user’s behaviours are conditioned by a user profile document; (ii) a Service Broker agent ‘knows’ what kinds of Instance Bases are stored locally (i.e. on the computer the agent resides); and (iii) a Service Provider agent ‘knows’ the ontology that defines the representation logic for the kind Instance Base it interfaces. In summary, the

author designs and develops software agents that are aware only of the local environment, with the belief that an overall *intelligence* can spawn from complex local interactions.

5.3.2 The Ontological Plane

Before describing the proposed instantiation of the prototype's Ontology Library, several preliminary specifications or critiques concerning the ontologies development are necessary. Given that, actually, ontologies are describing and storing complex information structures by *establishing agreements*, their implementation and especially their design are highly dependent on the particulars and the negotiations carried out by individuals that performed them. For example a designer could require the detailed description of the product physical structure and a summary description of the manufacturing process, while for a manufacturing engineer this situation could be inversed. How the actual ontology will look like will therefore depends on how the above mentioned individuals are negotiating agreements. Of course, the more individuals are involved, the more difficult could be to establish 'detailed' ontological agreements. Moreover, the ontological purposes which these developers are having in mind subjectively influence the ontologies development process. For example the ontologies proposed in this thesis are primarily intended to demonstrate the sharing and reusability possibilities opened by the usage of ontologies and agents. Therefore, these ontologies, while improbable to contribute to the effort of developing an universally engineering design ontology, are intended to contribute to the process of *starting* the effort of developing an universally engineering design ontology. Given all these theoretical issues concerning the development of ontologies, the author tries to make clear the fact that the form, the structure, the content and the extent of his Ontology Library, which is the agreement established among three IT specialists and two mechanical engineers specialists, is arbitrarily constrained by this thesis purpose and therefore may differ from other Ontology Libraries.

The Ontological Plane consists of the Ontology Library and the distributed Instances Bases (see section 4.4.3). The Ontology Library for the proposed Prototype consists of the following hierarchy of ontologies (see figure 5.3):

- Engineering Design Ontology
- Functional Ontologies

- Structural Ontologies

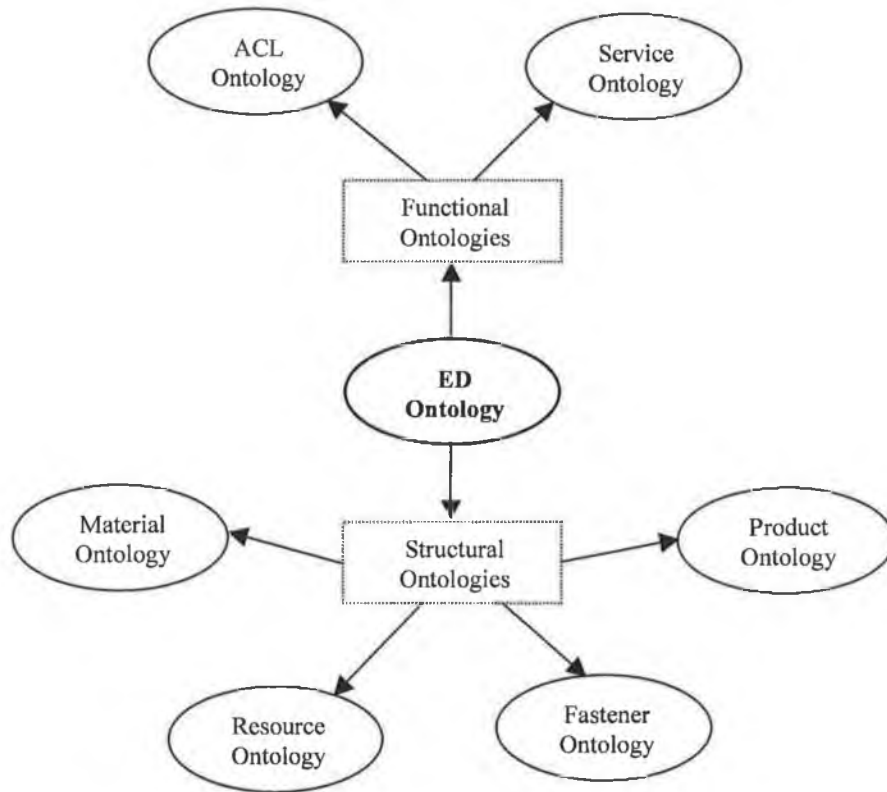


Figure 5.3 Prototype's Ontology Library

The *Engineering Design (ED) Ontology* is the generic ontology that describes the ontologies that make up the hierarchy (in this case the two functional and the four structural ontologies). It also represents the DEDS introductory card for the outside systems.

The *Service Ontology* and *Agent Communication Language (ACL) Ontology* are the *functional* ontologies of the implementation. The software agents use them to implement DEDS behaviors. The ACL Ontology is a FIPA compliant ontology implemented by the JADE framework and is used by agents in order to effectively communicate and exchange messages. The Service Ontology defines and describes the two categories of services which the prototype supplies, i.e. *Browse Service* and *Search Service*. Each category of services further consists of 'atom' services specific to each type of DEDS instance base (e.g. Browse Material service, Browse Product service, Search Product service).

The *Product Ontology*, *Material Ontology*, *Fastener Ontology* and *Resource Ontology* are *structural* ontologies. They define the schema for storing the actual design information structures. The Product Ontology defines the schema of any product structure at the detailed design phase (see figure 5.4).

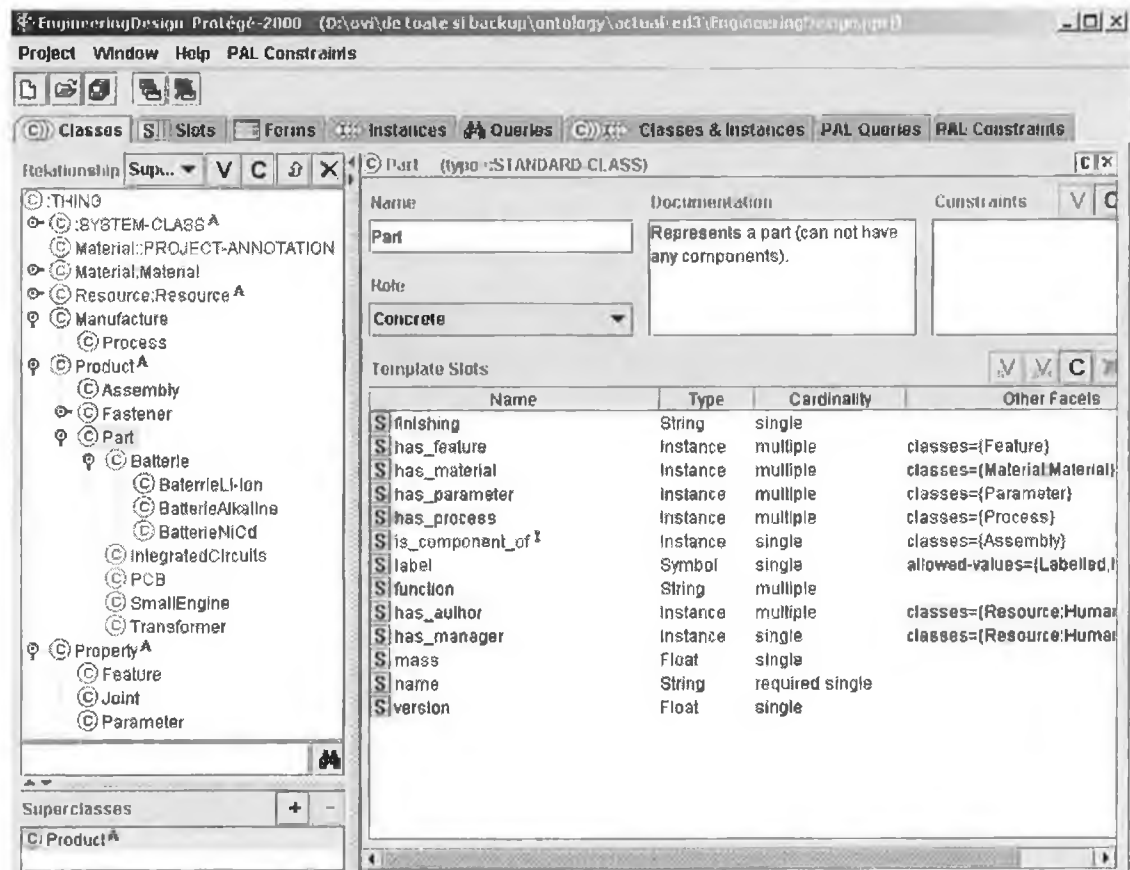


Figure 5.4 The Protégé 2000 view of the Product Ontology.

Each product is viewed as an hierarchy of *assemblies* and *parts*, with each assembly being further made-up of further assemblies (also called subassemblies) and parts. The main constraint defined is that, while a part can be component of an assembly, an assembly cannot be a component of a part. Furthermore the assemblies and the parts are defined in terms of their characteristics (e.g. name, mass, version) and relations (*has_author*, *has_manager*, *has_features*, *has_material*) that can link them to instances from other ontologies (see figure de la sf. de sectiune).

The Material Ontology (see figure 5.5) describes the materials information, needed for designing, in terms of properties such as category/class (e.g. ceramic ferro-metal, fibre, glass, laminate), subcategory/subclass (e.g. carbide and traditional ceramic for ceramic category) and properties (e.g. name, density, colour, texture, impact strength, tensile strength, fatigue, sustainability and environmental issues).

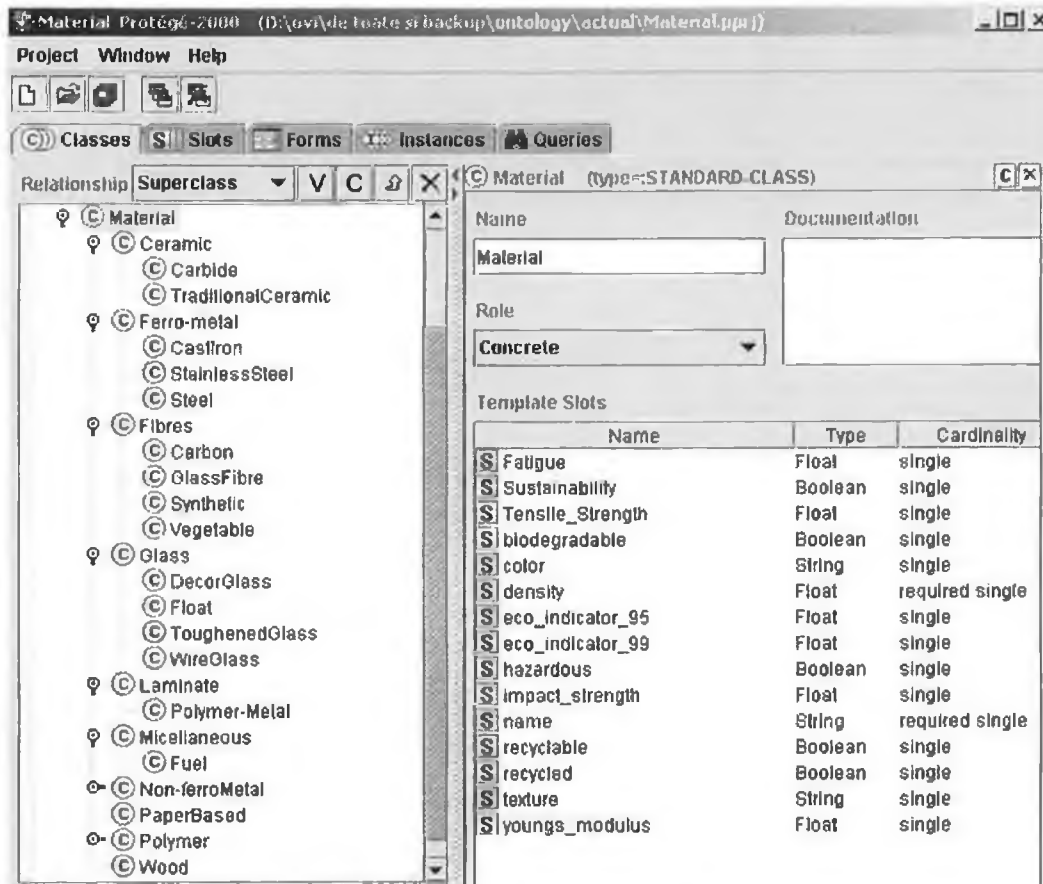


Figure 5.5 The Protégé 2000 view of the Material Ontology.

The Fastener Ontology provides a schema for storing fastener related information structures, such as classes and subclasses of fasteners, properties (e.g. name, mass, type, function, weld). This ontology also provides details for the assembly and disassembly processes, such as the tool needed and time the required.

The Resource Ontology defines the design resources involved in the distributed engineering design activity. At this stages only the human resources have been implemented in terms of characteristics such as name, department, role, e-mail and telephone.

While the Ontology Library defines the logic of storing the DEDS information, the Instance Bases actually store the design information structures. Together they (i.e. Ontology Library and Instances Bases) enable the machines (i.e. software agent systems) to work with complex structures of information. For example, figure 5.6 shows part of a machine-enabled design artefact of a smoke alarm.

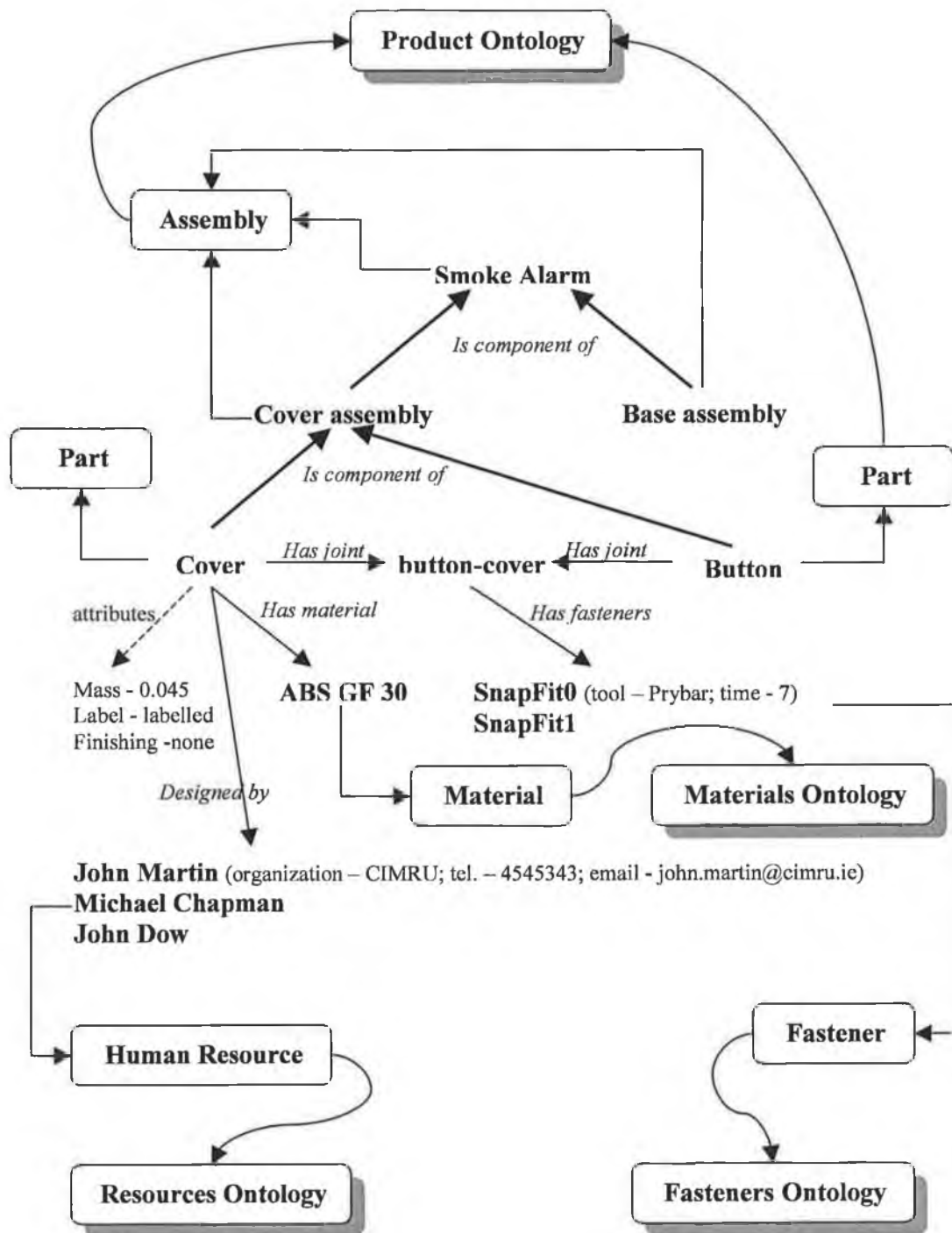


Figure 5.6 A part (two subassemblies and two components) of a Smoke Alarm design information (bold texts represent instances of ontological concepts, i.e. ontological instances).

In figure 5.6 the thick straight arrows represent structural relationships (i.e. *is component of* relationships), the normal arrows represent functional relationships (e.g. *has material, has fastener*), the elbow arrow connectors (e.g. between **Cover Assembly** and **Assembly**) stand for 'is instance of' relation (e.g. **Cover Assembly** is an instance of **Assembly** concept) and the curved arrow connectors (e.g. between **Assembly** and **Product Ontology**) stand for 'is concept from' relation (e.g. **Assembly** is a concept described in the **Product Ontology**). In this example, all the design information concerning the *Smoke Alarm* is explicitly represented and stored in instance bases in a form defined by the Ontology Library and, therefore is ready to be used by software components (i.e. software agents).

5.3.3 The Software Agents Plane

The first phase in the validation of the OSA system consisted of enabling the *access to* and the *sharing* of design information structures (by means of ontologies and their instance bases). The next necessary phase focused on the development and implementation of software agents that (i) *integrate* the DEDES actors in this pool of information (by means of OSA services) and (ii) *regulate* and *control* the system services.

The Prototype implements the following integrative mechanisms:

- Object Agents developed only for the human users (i.e. *username:MyAgent*) integrates the DEDES human actors;
- Service Broker and Service Provider agents (Browse and Search services for Product, Material, Fastener and Resource ontologies) integrate the instance bases;

The Prototype also uses the following regulation and control agents:

- Profile Broker agent that retrieves the available services based on the Service Ontology definitions and Requester (i.e. *username:MyAgent* agents) credentials. The Personal Profile service has not been implemented.
- Service Advertiser agent is provided by the JADE framework through the Directory Facilitator;

At its activation, an *username:MyAgent* agent sends a request message to the *Profile Broker*. Based on the received username, the Profile Broker agent retrieves the available services for this particular agent and sends them back to the *username:MyAgent* agent. In

accordance with the returned messages, the username:MyAgent activates its GUI and waits for the user to request services. Figure 5.7 shows the GUI of Cami's Object Agent.

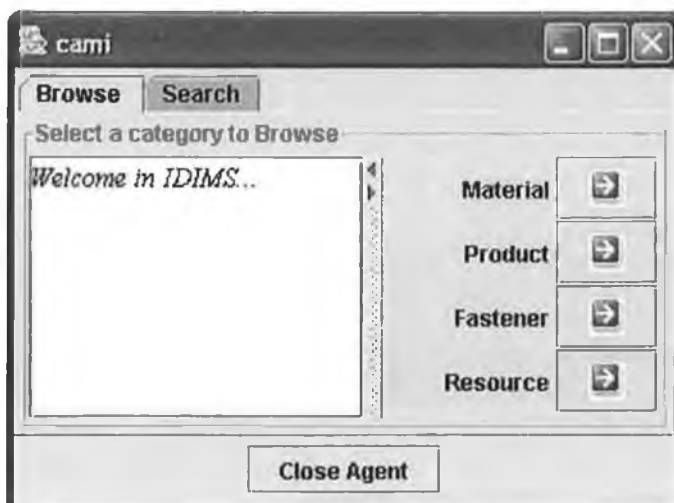


Figure 5.7 The GUI of cam:MyAgent agent.

The *Service Broker* agents are instantiated on each machine that stores Instance Bases. While its main behaviors are the same (register its services with the *Directory Facilitator*, instantiate and execute *Service Provider* agents) for each instance of the agent, some differences that depend on the kind(s) of Instance Base(s) with which it interfaces, exists. As already mentioned the Prototype provides Browse and Search services for each kind of structural Instance Base. This two categories of services for four kinds Instance Bases equals eight atomic services (i.e. browse_product, search_product, browse_material, search_material, browse_fastener, search_fastener, browse_resource, search_resource). Each of the above eight mentioned services are actually provided by a specific implementation of the Service Provider. Therefore, the local Instance Base(s) is(are) interfaced by two Service Broker agents, i.e. Browse Broker and Search Broker which will instantiate and lunch Instance Base specific Provider Agents (i.e. browse_product, search_product, browse_material, search_material, browse_fastener, search_fastener, browse_resource or search_resource Provider agent). After performing specific operations on the particular Instance Base, the Service Provider agent will return the result to the username:MyAgent, the requester agent.

Further clarification of the behaviors implemented by the OSA Prototype will be described in two scenarios, i.e. *request for browse* and *request for search* services.

Request for Browse Service

Suppose that an engineer designer wishes to browse the Material Base. At this time he/she has to explicitly tell his/her agent to do it (i.e. mouse click on the Material button under the Browse tab, see figure 5.8).

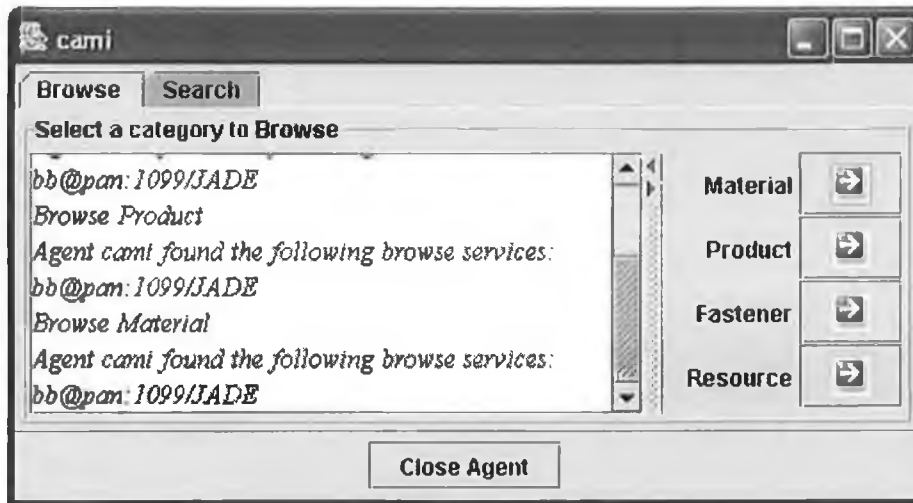


Figure 5.8 Cami has requested the Material Browse service

Once the proper broker agent(s) is(are) identified (from the Directory Facilitator), the operations performed by the Prototype to deliver the service are presented in figure 5.9

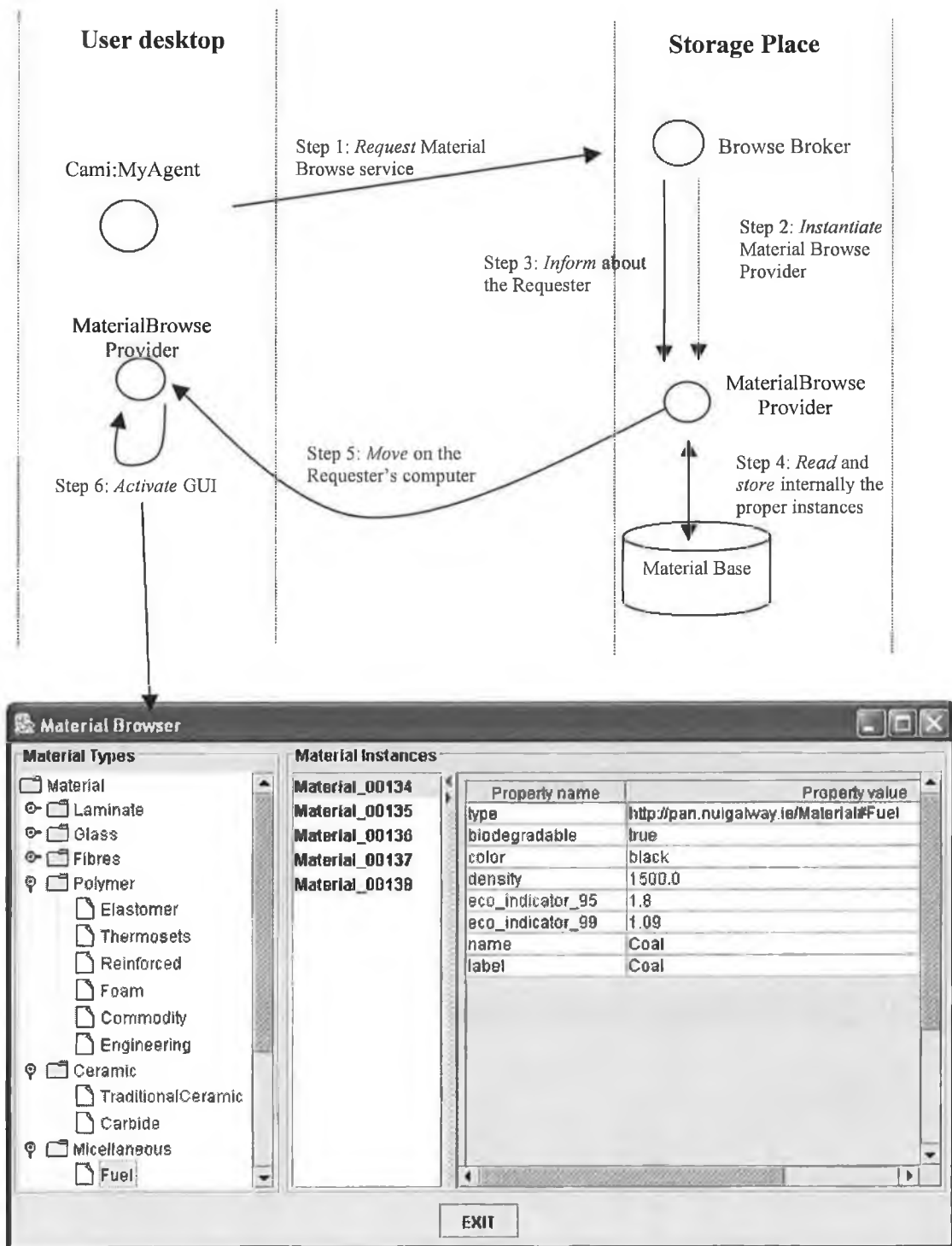


Figure 5.9 The material_browse Service Provider's GUI presented to the requester.

Of course, for each Material Base stored within the system, a material_browse Service Provider will move on the Requester's computer and will display its GUI, dealing in this

way with the distribution of design information. Therefore, it is possible for the user to have more than one browsing window for materials instances.

Request for Search Service

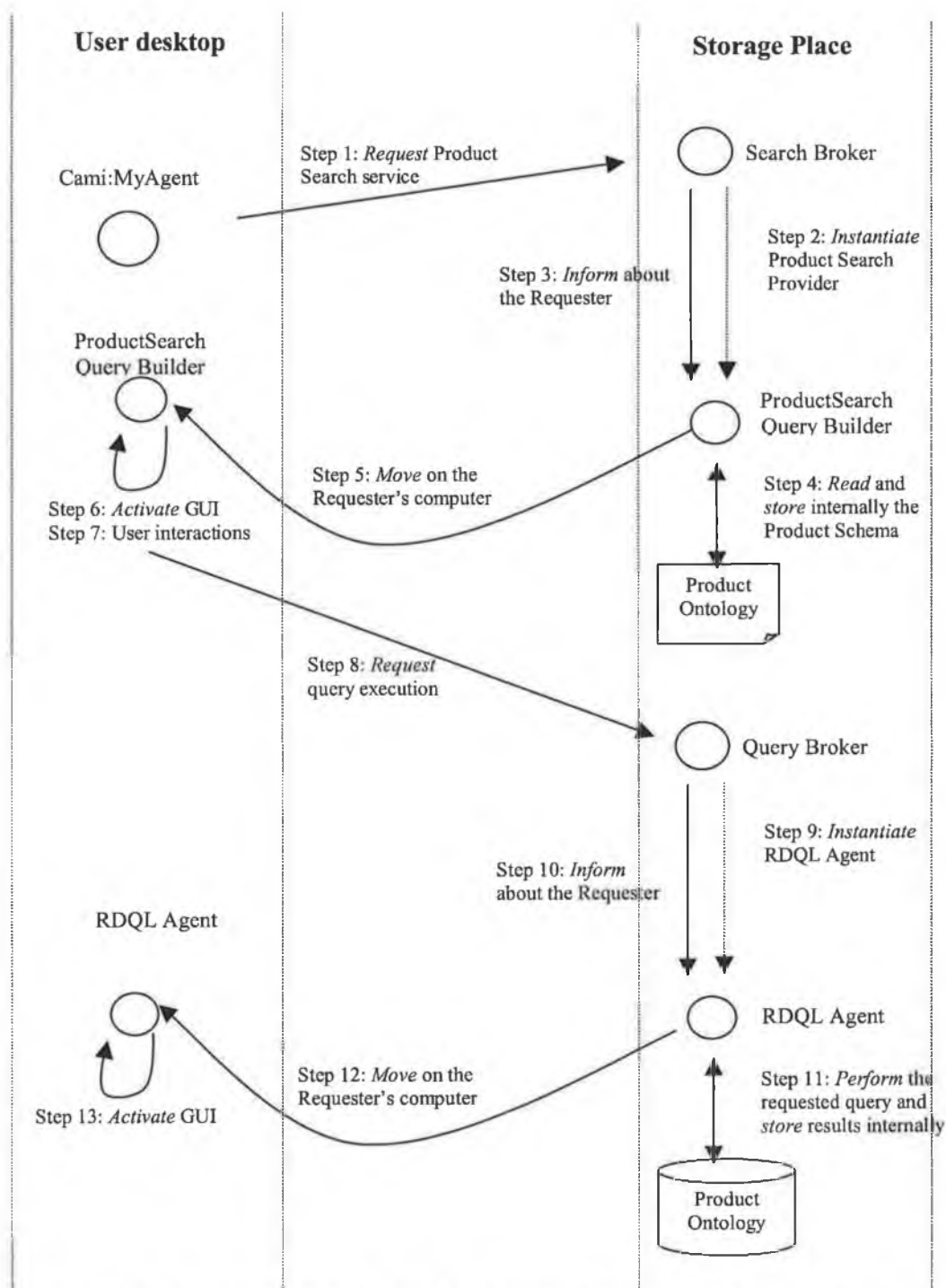


Figure 5.10 The providing of the SearchProduct service by the OSA Prototype

The providing of the Search Service has the same internal logic as for the Browse Service (the Search Broker agent instantiates and activates the provider of the service, which in turn will perform specific operations and will move the result on the Requester's computer). However, the inter-agents interactions and behavior are a little bit more complex, as can be seen in figure 5.10.

The GUIs presented to the Requester of the search_product service (step 6,7 and step 13) may look like the ones in the figure 5.11.



Figure 5.11 The GUIs presented to the user came by the ProductSearch Query Builder agent and RDQL agent

The complexity of the Search Service arises from the already mentioned approach to the development of software agents, i.e. as much as possible the agents should 'know' only about their local environment. Therefore, since the search service requires Requester's inputs accordingly to ontological descriptions, there was the need for a ProductSearch Query Builder agent, which will instantiate and activate the RDQL agent (which will provide the result of the service). In this case two agents play the role of the Service Provider agent from the OSA model, i.e. the ProductSearch Query Builder agent and RDQL agent. However, this *complex* behavior is the result of interactions between *simple* agents.

5.4 Conclusions

In this chapter the author introduced the OSA system built along the architectural specifications as synthesized and described in chapter four (see section 4.4). This OSA system actually acts as an instantiation of the proposed architectural framework. By providing specific DEDS *services*, the OSA system intends (i) to enable the reusing and sharing of design information structures, (ii) to integrate the DEDS objects in a cooperation enabled environment, and (iii) to employ non-human software-based control mechanisms for its administration.

Furthermore, based on the deduced specifications of the OSA system, a prototype consisting of instantiations of the kinds of ontologies and software agents characterized in the architectural framework model (figure 4.5) and more specifically identified by the OSA model (figure 5.2) in terms of OSA services, has been proposed. The prototype has been implemented using Protégé 2000 tool for the designing and developing of the Ontology Library and JADE framework and JAVA programming language for the implementation of the OSA software agents.

The author contends that, beside the logic of thought that led to the proposed architectural framework, the specified OSA system and its implemented prototype further validate the viability and practicability of the architectural framework for the DEDS.

In the next chapter the author will summarize the research he carried out and the results obtained. Based on the research results further developments and recommendations for future work will be proposed.

Chapter 6

Conclusions and Future Work

6.1 Research Summary

6.2 Research Results

6.3 Further Development and Recommendations for Future Work

6.4 Final Remarks

6.1 Research Summary

The object of this research is the distributed engineering design, understood both as a process and as an organization. Based on the literature review, the author characterizes the *process* of designing as an information transformation activity performed and supported by designers' problem solving and decision making skills (Luckman 1984; Pugh 1991; Hubka and Eder 1996; Gero 2000) (Roche 1999). The input consists of abstract statements of design requirements and the output represents detailed information that specifies the product (Hubka and Eder 1996; Chira, Chira et al. 2003). Besides the accomplishment of the initial requirements under the initial constraints, any end product generally has some other characteristics such as: it responds to consumer demand, it is economically manufactured, and it fulfills a human need (Feilden 1963; Pugh 1991; Lang, Dickinson et al. 2002). Moreover, the engineering design process represents the starting phase of a process consisting of a set of interrelated phases that also include manufacturing, supply, use, maintenance and disposal (Eder 1998; Roche 1999; Lang, Dickinson et al. 2002).

Concerning its organization, the author found that the key characteristics of the distributed engineering design are (i) the inherent *distribution* of design resources, especially of designers (Cutkosky, Englemore et al. 1997) (Olsen, Cutkosky et al. 1994; Siemieniuch and Sinclair 1999) and design information (Cross 1994; Pahl and Beitz 1996) (Bertola and Teixeira 2003); (ii) the importance of effective *teamwork* and collective effort (Olsen, Cutkosky et al. 1994; Cross and Cross 1995) (Siemieniuch and Sinclair 1999) (Patel, D'Cruz et al. 1997); (iii) the critical need for a robust *cooperation* process (Lawson 1990; Brereton, Cannon et al. 1994; Olsen, Cutkosky et al. 1994; Harvey and Koubek 1998; MacGregor 2002), and (iv) the *role of the computer* which acts not only as a tool but also as a engineering design workplace (MacGregor 2002).

Focusing on the role that information structures play (in distributed engineering design), the author categorized the problems associated with distributed engineering design, as follows:

- The *quantity* of information structures that designers need to handle (e.g. search, identify, retrieve, use/process, store) is already burdensome and is increasing at a rapid pace.
- Designers' knowledge and, therefore, performance depend on the readily available information, which is not always of the appropriate *quality*.
- The above problems result in an unsatisfactory *cooperation* within the organization and, consequently the intensification and recurrence of the same problems.

These shortfalls of distributed engineering design outline the drivers for this research. Therefore, it has been proposed to find and describe, based on current advances in computer-related technologies, an architectural framework that would characterize a software system for improving the quantitative and qualitative access of designers to however complex distributed design information structures.

Given the identified characteristics of the distributed engineering design (both as process and organization), the author contended that distributed engineering design is an inherently complex organization where the study of its separate parts (besides being a burdensome task) will not provide an understanding of the whole. Therefore, the author explicitly proposed a more holistic approach to studying the distributed design organization, in a bid to identify patterns rather than isolate phenomena or key behaviors rather than local actions. This appropriate approach had been identified in the sciences of General System Theory and Cybernetics Systems, sciences of whose results and methods have been used along the development of this research.

6.2 Research Results

This thesis aimed to discover and to provide knowledge for the design and the development of a machine enabled semantic framework that would improve the qualitative and quantitative management of design information structures within the distributed engineering design organization. In order to achieve this overall aim, the research was carried out along six main objectives (see section 1.3). In the following the author will present the research results for each objective (as discussed in chapter one).

Objective 1. To investigate and characterize the engineering design process performed in a distributed environment and its problematic aspects;

- A literature-based characterization of the concept of **engineering design** (the process, the designing actor, the objectives and the life-cycle information aspects) and its understanding in a **distributed environment** context (distribution, teamwork, cooperation, the role of the computer) has been synthesized.
- A summarization model of the above has been introduced to describe the distributed design organization as the result of complex cooperation enabled inter/intra-actions among engineering designers, design methods, methodologies and tools, and ICT support.
- It has been found that there are significant inadequacies in the management of design information structures at both quantitative and qualitative levels. This

situation may result in an unsatisfactory cooperation within the distributed design organization and, consequently the intensification and recurrence of the same undesired situations.

Objective 2. To research and study alternative theories for thinking and modelling the distributed engineering design process;

- Based on the results to date and on a set of subjective and objective criteria General Systems Theory and Cybernetics have been proposed to provide the scientific tools for further investigating the research domain.
- It has been argued and demonstrated that the distributed engineering design is an open cybernetic system, i.e. the Distributed Engineering Design System (or DEDES), made of three main subsystems (i.e. Human, Infrastructure and Engineering Design Model systems).
- A model of the DEDES (figure 2.4) that summarizes and focuses the research. In short, the DEDES is modelled as an organized collection of humans, machines and design methodologies working together to transform information-based inputted requirements and constraints into appropriate product specifications information.
- A minimal set of requirements for a solution system that will improve the information-related problematic aspects of the DEDES has been proposed. These requirements comprise of (i) reusable, shared and formal design information structures, (ii) integrative mechanisms capable of translations and mappings between different contexts and (iii) control mechanisms able to regulate the functionality of the system.

Objective 3. To investigate current research in information and knowledge management for identifying supporting technologies for a possible solution to the identified problematic aspects (from objective 1);

- The research associated with this thesis has identified **ontologies** as the solution for organizing the system's information resources. Ontologies describe concepts and relations assumed to be always true independent from a particular domain by a community of actors (humans or machines) that commit to that view of the world. Therefore, by specifying content specific agreements, ontologies are facilitating information sharing and reuse among systems that submit to the same ontology/ontologies by the means of ontological commitments. However, the form,

the content and the power of ontologies are critically determined by process of negotiating commitments among the concerned actors.

- The **agent-based systems** can provide the autonomous, proactive and cooperative hard working helper that can both integrate disparate components and regulate DEDES behaviors (or functionalities). Considered an important new direction in software, agents and MAS provide techniques to manage the inherent complexity of the software systems and are appropriate for domains in which data, control, expertise and resources are inherently distributed.
- It has been found that ontologies and Agent-based Systems can enable the development of software tools to support the inherent complexity and inherent distribution of the DEDES.

Objective 4. To analyze the requirement needs for a solution according to the findings from previous objectives, i.e. the driving problems (from point 1), the research and therefore the thinking approach (from objective 2), and available supporting technologies (from objective 3);

- Given that the DEDES has been explicitly described in the terms of General Systems Theory and Cybernetics, it was allowed and made possible to use probably one of the most powerful tools in system's research, i.e. Systems Thinking. This cognitive and applied mode of thinking further helped in reaching the subsequent results.
- The need for semantic support is critical for improving the performance of the engineering design process.
- The main pattern of a DEDES consists of a hierarchy of fractal-like cooperation processes that stretch and reach any DEDES component at any level of granularity. In this light, the DEDES cooperation processes have a critical role in enabling semantics within DEDES.
- The use of ontologies and software agents can provide the needed technological support for enabling semantics within DEDES.

Objective 5. To synthesize the architectural framework along the identified supporting technologies (from objective 3);

- At this point, the research results converged into the specification of an architectural framework (figure 4.5). This architectural framework defines and characterizes the computational context in which the solution software system acts

in order to improve the manner in which the engineering design process is carried out within a distributed organization. The proposed framework has been described along its four structural layers (i.e. Object, Integration, Schema and Instance layers) and its two functional planes (i.e. Ontological and Agent System planes).

Objective 6. To instantiate a software system along the underlying computational context as described by the architectural framework (from point 5).

- The ontology-based software agent system (or OSA) for supporting an quantitatively and qualitatively improved designers' access to complex design information structures.
- Validation of the architectural framework through OSA system.
- The prototype of the OSA system.

6.3 Further Development and Recommendations for Future Work

The author suggests that further development to be categorized on three levels, as follows:

1. the *research* level
2. the proposed *architectural framework* level
3. the *OSA system* level

At the *research* level, based on the results of this research, the author recommends the use of General System Theory and Cybernetics for 'in-house' study of the distributed engineering design. While this research constructs and provides a generic view of the DEDES, more detailed views are also necessary for fully understanding the structures and the interactions within the DEDES, such as:

- Designers' negotiations strategies;
- Adequate understanding of each of the phases of the engineering design (i.e. the Engineering Design Model system);
- Designer – Computer interactions and Designer – Computer – Designer interactions;
- Further research in software agent paradigm for identifying the appropriate agent architectures and languages to be used for implementing DEDES behaviours;
- Further investigation in ontologies necessary for the design and development of an Ontology Library aligned to recognized standards.

At the *architectural framework* level, the author recommends the followings:

- The use of the results of the above suggestions for a more specific description of the kinds and behaviours of agents and subsequent ontologies needed for controlling and implementing DEDS functionalities or services, e.g. support for all the stages of design for all the design models used within the enterprise;
- To characterize the conditions for enabling the interoperability among existent design applications and tools (e.g. CAD, PDM), if possible without human mediation;
- The design and description of an additional layer (or functionality) for integrating the DEDS in the wider system of *product development* and even into the *global business* system.

Based on the recommendations from the research and architectural framework levels, the author identified that, at the *OSA system* level, the following implementations are necessary:

- Dedicated agents or agent systems and ontologies for each of the design model used;
- Object agents capable of collaborating and of performing negotiations;
- Intelligent, or at least, adaptable human interfaces;
- Ontologies that adhere to established standards, so interactions between multiple extended enterprises can be enabled;
- Discovery of more DEDS services;
- The production/creation of more ontological instances (so a critical mass of information structures is obtained and the full benefits of this approach is more easily measured);

A special suggestion, materialized from the author experience gained during this research, is the recommendation of ‘plugging-in’ the DEDS architecture to the future Semantic Web. The Semantic Web (SW) is an emerging concept that launches the idea of having data on the web defined and linked in a way that it can be used by people and processed by machines (Berners-Lee 1998; Decker, Harmelen et al. 2000; Fensel 2000; Ramsdell 2000; Berners-Lee, Hendler et al. 2001; Dumbill 2001; Hendler, Berners-Lee et al. 2002) in a “wide variety of new and exciting applications” (Swartz and Hendler 2001). It develops “languages for expressing information in a machine processable form” (Berners-Lee 1998), so as to enable the machine to participate and help inside the information space

(Benjamins, Contreras et al. 2002): "The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users" (Berners-Lee 1998). The author believes that the DEDES will benefit from the large amount of machine enabled information structures that will make the next generation Web. Moreover, while the proposed architectural framework has been developed for Intranet (where policies for representation of information can be enforced), the SW can open the vastness and variety of Internet to the DEDES objects.

6.4 Final Remarks

This object of the investigation of this thesis is the distributed engineering design domain. A key characteristic of this domain is its highly inherent heterogeneity. Under the banner of distributed engineering design are classified organic (e.g. humans) and inorganic (e.g. computers) physical entities, conceptual entities (e.g. engineering design models, design information), physical phenomena (e.g. cooperation between designers, negotiations) and all sorts of relations (e.g. human-to-computer interaction, human-to-human computer mediated interaction, application-to application interaction). The author identifies this plane as the *reality level* because it encompasses all the domain components as they are *in the real world*. A *representation* of this first level has been necessary in order to handle this complex diversity, so the research could go further. Therefore, in chapter two, the second plane called *first-order representation*, used informal and semi-formal forms of human language (in particular of the English language) and diagrams to catch the key characteristics of the domain and to model it. As a result, models of the distributed engineering design organization were proposed (see figure 2.2 and figure 2.4), models that enabled and supported the systems analysis and systems thinking carried out in agreement with the focus of the research. The resulting conclusions, followed by a requirement analysis made necessary a third plane, i.e. a representation of the representation of reality (called for this reason the *second-order representation level*). The second-order representation level further formalizes the first-order representation level towards a fully formal, self-explanatory and semantically enabled code or codes of signs (see figure).

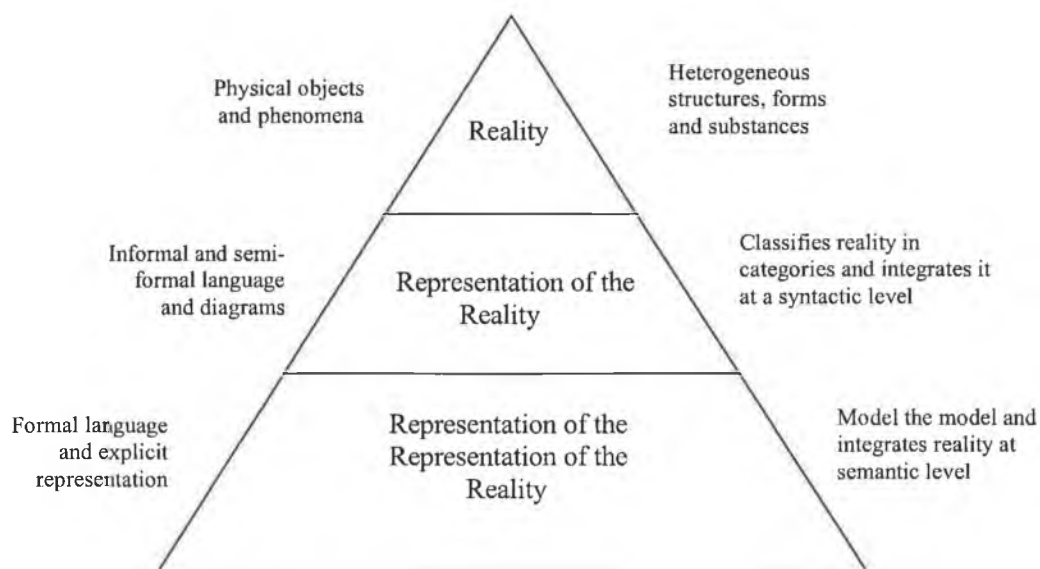


Figure 6.1 Research progression from implicit and heterogeneous knowledge towards explicit and homogeneous knowledge.

The upper part of the triangle, which stands for the real world of distributed engineering design domain, signifies that the unmediated by signs knowledge that the DEDS (of course, the author refers especially to the human components of the DEDS) has about itself is fragmented and depends on the personal visual and tactile perceptions. Moreover this knowledge is too often implicit. Therefore, design knowledge is stored on often 'unreliable memories' (i.e. human memory), it is difficult to be communicated among designers and, it is even more difficult to be shared with the (advanced) design tools.

Once systems of signs (such English language and diagrams) are used to stand for the real objects and their relations the knowledge can be shared, exchanged and improved by involving (besides human perceptions) intellectual capabilities. At this level the quantitative and qualitative access to knowledge is improved, depending not solely the perceptual capabilities, but also on the storage capacities, personal expertise and competence to interpret codes of signs. The information available is still implicit, depending in this way on the personal experience.

Furthermore, at the third level, of knowledge about knowledge, the need for perceptions and interpretations of different systems of signs are by-passed by a single system of signs capable of describing itself. The author contends that this is the level where the human designers, the design tools, the design methodologies and the design information structures can truly interoperate.

For example, at the first level a computer is a physical object made of wires, circuits, plastic and so on. At the second level, computer is a word (in English language is word computer) that implicitly represents the physical entity. Further, at the third level, computer is the explicit and formal knowledge that can be represented about the physical entity (i.e. it is an ontological concept).

Finally, the present research work can also be viewed as a systematic reduction of heterogeneity. Therefore at the third level distributed engineering design domain is intended to be a corpus of explicit and formally represented homogenous knowledge.

References

- Ackoff, R. L. (1981). Creating the Corporate Future. New York, John Wiley & Sons.
- Ahn, S.-H., S. Roundy, et al. (1999). "Design Consultant": a Network-Based Concurrent Design Environment. International Mechanical Engineering Congress & Exposition, Nashville, Tennessee.
- Anumba, C. J., O. O. Ugwu, et al. (2002). "Collaborative design of structures using intelligent agents." Automation in Construction **11**: 89-103.
- Arias, E., H. Eden, et al. (2000). "A. Gorman and E. Scharff. "Transcending the Individual Human Mind - Creating Shared Understanding through Collaborative Design." ACM transactions on Computer-Human Interaction **Vol. 7, No. 1**: 84 - 113.
- Arpírez, J. C., O. Corcho, et al. (2001). WebODE: a Workbench for Ontological Engineering. First International Conference on Knowledge Capture (K-CAP'01), Victoria (B.C.), Canada.
- Artala, A., E. Franconi, et al. (1996). "Part-Whole Relations in Object-Centered Systems: an Overview." Data and Knowledge Engineering **20(3)**: 347-383.
- ASC American Society for Cybernetics. **2003**.
- Backlund, A. (2000). "The definitoin of *system*." Kybernetes **29(4)**: 444-451.
- Barrow, J. (1992). Theories Of Evervthing, Random House UK Distribution.
- Bartlett, G. (2001). Systemic Thinking. a simple thinking technique for gaining systemic focus. The International Conference on Thinking "Breakthroughs 2001", Auckland, New Zealand.
- Bear, S. (2002). "What is cybernetics." Kybernetes **31(2)**: 209-219.
- Bellifemine, F., G. Caire, et al. (2003). "JADE A White Paper." EXP **3(3)**.
- Bellifemine, F., G. Caire, et al. (2003). JADE Administrator's Guide.
- Bellifemine, F., G. Caire, et al. (2003). JADE Progarmmer's Guide.
- Bellifemine, F., A. Poggi, et al. (1999). JADE - A FIPA-compliant agent framework. Proceedings of PAAM'99, London.
- Benjamins, V. R., J. Contreras, et al. (2002). Six Challenges for the Semantic Web. International Semantic Web Conference (ISWC2002), Sardinia, Italia.
- Benjamins, V. R., D. Fensel, et al. (1998). Knowledge Management through Ontologies. Second International Conference on Practical Aspects of Knowledge Management (PAKM98), Basel, Switzerland, 29-30 Oct. 1998.
- Bennett, S., S. McRobb, et al. (1999). Object-Oriented System Analysis and Design using UML. London, McGraw-Hill.
- Bergamaschi, S., S. Castano, et al. (1998). An Intelligent Approach to Information Integration. Formal Ontology in Information System. N. Guarino. Amsterdam, IOS Press.
- Berners-Lee, T. (1998). Semantic Web Road Map.
<http://www.w3.org/DesignIssues/Semantic.html>, World Wide Web Consortium.
- Berners-Lee, T., J. Hendler, et al. (2001). "The semantic web." Scientific American **284(5)**: 34-43.
- Bertalanffy, L. V. (1976). General System Theory: Foundations, Development, Applications, George Braziller.
- Bertola, P. and J. C. Teixeira (2003). "Design as a knowledge agent

- How design as a knowledge process is embedded into organizations to foster innovation." Design Studies 24(2): 181-194.
- Blazquez, M., M. Fernandez, et al. (1998). Building Ontologies at the Knowledge Level using the Ontology Design Environment. 11th Knowledge Acquisition Workshop, KAW98, Bamff, Canada.
- Boer, S. J. D. (1989). Decision Methods and Techniques in Methodical Engineering Design, University of Twente.
- Borst, P., H. Akkermans, et al. (1997). "Engineering Ontologies." International Journal of Human-Computer Studies 46(Special Issue on Using Explicit Ontologies in KBS Development): 365-406.
- Bradshaw, J. M. (1997). An Introduction to Software Agents. Software Agents. J. M. Bradshaw. Cambridge, MIT Press.
- Brazier, F. M. T., B. M. Dunin-Keplicz, et al. (1997). "DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework." International Journal of Cooperative Information Systems 6(Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems): 67-94.
- Brazier, F. M. T., L. V. Moshkina, et al. (2001). "Knowledge level model of an individual designer as an agent in collaborative distributed design." Artificial Intelligence in Engineering 15: 137-152.
- Brennan, A. (1996). A Graphical User Interface Design Tool to Facilitate Managerial Learning. CIMRU, Univeristy College Galway.
- Brereton, M. F., D. M. Cannon, et al. (1994). Collaboration in Engineering Design Teams: Mediating Design Progress through Social Interaction. Proceedings of the Design Protocol Analysis Workshop, Faculty of Industrial Design Engineering, Delft University of Technology, the Netherlands.
- Caire, G. (2002). JADE Tutorial: Application-Defined Content Languages and Ontologies.
- Carver, N., V. Lesser, et al. (1993). Distributed sensor Interpretation: Modeling Agent Interpretations in DRESUN, UMass Technical Report, UMCS 93-75.
- Chaib-draa, B. (1996). "Interaction Between Agents in Routine, Familiar and Unfamiliar Situations." International Journal of Intelligent & Cooperative Information Systems 5(1): 1-25.
- Chaib-draa, B. and F. Dignum (2002). "Trends in Agent Communication Language." Computational Intelligence 18(2).
- Chen, L. and S. Lee (2002). "A Computerized Team Approach for Concurrent Product and Process Design Optimization." Computer Aided Design 34(1): 57-69.
- Chikofsky, E. J. and J. H. C. II (1990). "Reverse Engineering and design recovery: A taxonomy." Software Magazine: 13-17.
- Chira, C. (2002). Design, Development and Testing of a CAD Integrated Design for Environment Software Tool. Science. Galway, Galway Mayo Institute for Technology.
- Chira, O., C. Chira, et al. (2003). An agent-based approach to knowledge management in distributed design. 10th ISPE International Conference on Concurrent Engineering: Research and Applications, Madeira Island, Portugal.
- Chu, E., K. Srihari, et al. (1996). "Distributed Artificial Intelligence in Process Control." 19th International Conference on Computers and Industrial Engineering.

- Corning, P. A. (2001). "Control information" The missing element in Norbert Wiener's cybernetic paradigm?" Kybernetes 30(9/10): 1272-1288.
- Coyne, R. D., M. A. Rosenman, et al. (1990). Knowledge based Design Systems, Addison Wesley.
- Crabtree, R. A., M. S. Fox, et al. (1997). "Towards an Understanding of Collaborative Design Activities." Research in Design Engineering 9: 70-84.
- Cross, N. (1994). Engineering Design Methods, J. Wiley & Sons.
- Cross, N. (2000). Design as a Discipline. Doctoral Education in Design: Foundations for the Future. D. Durling and K. Friedman. Stoke-on-Trent, Staffordshire University Press.
- Cross, N. and A. C. Cross (1995). "Observations of teamwork and social process in design." Design Studies 16(2): 143-170.
- Cutkosky, M. R., R. S. Englemore, et al. (1997). PACT: An Experiment in Integrating Concurrent Engineering Systems. Readings in Agents. M. N. Huhns and M. P. Singh. San Francisco, CA, USA, Morgan Kaufmann: 46-55.
- Decker, S., F. v. Harmelen, et al. (2000). "The Semantic Web - on the respective Roles of XML and RDF." IEEE Internet Computing.
- DeLoach, S. A. (1999). Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems. Agent-Oriented Information Systems (AOIS) '99.
- Draper, F. and M. Swanson (1990). "Learner-directed systems education. A successful example." System Dynamics Review 6(2): 209-213.
- Dumbill, E. (2001). Building the Semantic Web. <http://www.xml.com/pub/a/2001/03/07/buildingsw.html>, XML.com.
- Durfee, E. H. (2001). "Scaling Up Agent Coordination Strategies." IEEE Computer 34(7): 39-46.
- Durfee, E. H. and V. R. Lesser (1991). "Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation." IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Distributed Sensor Networks SMC-21(5): 1167-1183.
- Eder, W. E. (1998). "Design Modelling - A Design Science Approach (And Why Does Industry Not Use It?)." Journal of Engineering Design 9(4).
- Feilden, G. B. R. (1963). Engineering Design. London, Report of Royal Commission - HMSO.
- Fensel, D. (2000). Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Berlin, Springer.
- Fernandez, M., A. Gomez-Perez, et al. (1997). METHONTOLOGY: From Ontological Art Towards Ontological Engineering Workshop on Ontological Engineering. Symposium on ONtological Engineering of AAI, Standford, California.
- Fernandez-Lopez, M. (2001). "Overview Of Methodologies for Building Ontologies." Intelligent Systems 16(1): 26-34.
- Fernandez-Lopez, M., A. Gomez-Perez, et al. (1999). "Building a Chemical Ontology Using Methontology and the Ontology Design Environment." IEEE Intelligent Systems and their applications January/February: 37-46.
- Fikes, R., Farquhar, A. (1999). "Distributed Repositories of Highly Expressive Reusable Ontologies." IEEE Intelligent Systems 14(2): 73-79.

- Finger, S. and J. R. Dixon (1989). "A Review of Research in Mechanical Engineering Design - Part 1- Descriptive, Prescriptive and Computer Based Models of the Design Process." Research in Engineering Design, Springer: 51-67.
- Finin, T., R. Fritzson, et al. (1994). KOML as an Agent Communication Language. Proceedings of the Third International Conference on Information and Knowledge Management.
- Finkelstein, L. and A. C. W. Finkelstein (1983). Review of Design Methodology. IEE Proceedings.
- Franklin, S. and A. Graesser (1996). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996, Berlin, Germany.
- Gaines, B. (1997). "Editorial: Using Explicite Ontologies in Knowledge-based System Development." International Journal of Human-Computer Systems 46: 181.
- Gammack, J. and S. Poon (1999). Communication Media for Supporting Distributed Engineering Design. 32nd Hawaii International Conference on System Sciences, Hawaii.
- Gasser, L. (1998). Social conceptions of knowledge and action: DAI foundations and open systems dynamics. Readings in Agents. M. N. Huhns and M. P. Singh, Morgan Kaufmann Publishers.
- Genesereth, M. R. and S. P. Ketchpel (1994). "Software Agents." Communications of the ACM. ACM Press.
- Genesereth, M. R. and N. J. Nilsson (1987). Logical Foundations of Artificial Intelligence, Morgan Kaufmann Publishers.
- Gero, J. (2000). "Computational Models of Inovative and Creative Design Process." Technological Forecasting and Social Change 64: 183-196.
- Geyer, F. (1994). The Challenge Of Sociocybernetics. 13th World Congress of Sociology, Bielefeld.
- Gomez-Perez, A. (1998). Knowledge Sharing and Reuse. The Handbook on Expert Systems. Liebowitz, CRC Press.
- Gomez-Perez, A. (1999). "Ontological Engineering: A State Of The Art." Expert Update. Ontono 2(3): 38-43.
- Gomez-Perez, A., N. Juristo, et al. (1995). Evaluation and assessment of knowledge sharing technology. Towards Very Large Knowledge Bases - Knowledge Building and Knowledge Sharing. N. J. Mars. Amsterdam, IOS Press: 289-296.
- Green, S., L. Hurst, et al. (1997). Software Agents: A review. Dublin, Intelligent Agents Group, Trinity College Dublin, Broadcom Eireann Research Ltd.
- Gruber, T. R. (1991). The Role of Common Ontology in Achieving Shareable, Reusable Knowledge Bases. Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference, San Mateo, Morgan Kaufmann, 1991.
- Gruber, T. R. (1993). "A Translation Approach to Portable Ontology Specification." Knowledge Aquisition 5(2): 199-220.
- Gruber, T. R. (1995). "Toward Principles for the Design of Ontologies Used for Knowledge Sharing." International Journal of Human and Computer Studies 43(5/6): 907-928.

- Gruber, T. R., J.M. Tenenbaum, J.C. Weber (1992). Toward a Knowledge Medium for Collaborative Product Development. Artificial Intelligence in Design, Pittsburg, USA, Kluwer Academic Publishers.
- Gruninger, M. and M. S. Fox (1994). The Role of Competency Questions in Enterprise Engineering. IFIP WG5.7 Workshop on Benchmarking - Theory and Practice, Trondheim, Norway.
- Gruninger, M. and M. S. Fox (1995). Methodology for the Design and Evaluation of Ontologies. IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing, Montreal, Quebec, Canada.
- Guarino, N. (1995). "Formal Ontology, Conceptual Analysis and Knowledge Representation." International Journal of Human and Computer Studies 43(5/6): 625-640.
- Guarino, N. (1997). Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. Summer School on Information Extraction, Frascati, Italy, July 14-19.
- Guarino, N. (1997). "Understanding, Building and Using Ontologies: A Commentary to "Using Explicit Ontologies in KBS Development." International Journal of Human and Computer Studies 46: 293-310.
- Guarino, N. (1998). Formal Ontology and Information Systems. Formal Ontology in Information Systems. FOIS'98, 6-8 June 1998., Trento, IOS Press,.
- Guarino, N., S. Borgo, et al. (1997). Logical Modelling of Product Knowledge: Towards a Well-Founded Semantics for STEP. European Conference on Product Data Technology (PDT Days 97), Sophia Antipolis, France.
- Guarino, N., M. Carrara, et al. (1994). Formalizing Ontological Commitments. National Conference on Artificial Intelligence, AAAI 94, Seattle, Morgan Kaufmann.
- Guarino, N. and P. Giaretta (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. N. Mars. Amsterdam, IOS Press: 25-32.
- Hales, C. (1987). Analysis of the Engineering Design Process in an Industrial Context. Department of Engineering. Cambridge, University of Cambridge.
- Harvey, C. M. and R. J. Koubek (1998). "Toward a Model of Distributed Engineering Collaboration." Computers & Industrial Engineering 35(1-2): 173-176.
- Hawking, S. (2001). The Universe in a Nutshell, Bantam.
- Hendler, J., T. Berners-Lee, et al. (2002). "Integrating Applications on the Semantic Web." Journal of the Institute of Electrical Engineers of Japan 122(10): 676-680.
- Heylighen, F. **Web Dictionary of Cybernetics and Systems**. Principia Cybernetica Web. F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels. 2003.
- Heylighen, F. and C. Joslyn (1992). What is Systems Theory? Principia Cybernetica Web. F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels. 2003.
- Heylighen, F. and C. Joslyn (2001). Cybernetics and Second-Order Cybernetics. Encyclopedia of Physical Science & Technology. R. A. Meyers. New York, Academic Press.
- Heylighen, F., C. Joslyn, et al. (1993). Principia Cybernetica Web. 2003.

- Heylighen, F., C. Joslyn, et al. (Oct 1, 1993 (created)). What are Cybernetics and Systems Science? Principia Cybernetica Web. F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels.
- Hirsch, B. (2000). "Extended Products in Dynamic Enterprises", *E-Business: Key Issues, Applications and Technologies*,: 622-628.
- Ho, J. and R. Tang (2001). "Towards an Optical Resolution to Information Overload : An Infomediary Approach." ACM.
http://protege.stanford.edu/doc/users_guide/index.html. 2003.
- <http://www.fipa.org> Foundation for Intelligent Physical Agents.
- Hubka, V. and E. Eder (1987). "A Scientific Approach to Engineering Design." Design Studies 8(3): 123-137.
- Hubka, V. and E. Eder (1996). Design Science. Springer-Verlag.
- IBM (2003). IBM Lotus Sametime. 2003.
- IEEE96 (1996). IEEE Standard for Developing Software Life Cycle Processes. New York (USA), IEEE Computer Society.
- Iglesias, C. A., M. Garijo, et al. (1999). A Survey of Agent-Oriented Methodologies. Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages.
- Jagdev, H. and J. Browne (1998). "The Extended Enterprise-A context for Manufacturing." Production Planning and Control 9(3): 326-339.
- Jennings, N. R. (2000). "On agend-based software engineering." Artificial Intelligence.
- Jennings, N. R., K. P. Sycara, et al. (1998). "A Roadmap of Agent Reasearch and Development." Journal of Autonomous Agents and Multi-Agent Systems 1(1): 7-36.
- Jennings, N. R. and M. Wooldridge (1998). Applications of Agent Technology. Agent Technology: Foundations, Applications, and Markets. N. R. Jennings and M. Wooldridge, Springer-Verlag.
- Joslyn, C. (1992). The Nature of Cybernetic Systems. Principia Cybernetica Web. F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels. 2003.
- Kemerling, G. (2002). Philosophy Pages. 2003.
- Kimura, F. (1997). Inverse manufacturing: From Products to Services. Managing Enterprises - Stakeholders, Engineering, Logistics and Achievement First International Conference Proceedings, MEP Ltd, London,.
- Kinny, D., M. Georgeff, et al. (1996). A Methodology and Modelling Technique for Systems of BDI Agents. Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, Springer.
- Kolb, D. (1984). Experiential Learning: Experience as the Source of Learning and Development. Prentice-Hall.
- Labrou, Y., T. Finin, et al. (1999). "Agent Communication Languages: The Current Landscape." IEEE Intelligent Systems.
- Lang, S. Y. T., J. Dickinson, et al. (2002). "Cognitive factors in distributed design." Computers in Industry 48: 89-98.
- Lawson, B. (1990). How Designers Think 2nd Ed.
- Lazansky, J., O. Stepankova, et al. (2001). "Application of the multi-agent approach in production planning and modelling." Engineering Applications of Artificial Intelligence 14(3): 369-376.

- Lesser, V. and D. Corkill (1981). "Functionally Accurate, Cooperative Distributed Systems." IEEE Transactions on Systems, Man, and Cybernetics **SMC-11(1)**: 81-96.
- Lesser, V. R. (1999). "Cooperative Multiagent Systems: A Personal View of the State of the Art." IEEE Transactions on Knowledge and Data Engineering **11(1)**.
- Love, T. (2002). "Constructing a coherent cross-disciplinary body of theory about designing and designs: some philosophical issues." Design Studies **23(3)**: 345-361.
- Luckman, J. (1984). An Approach to the Management of Design. Developments in Design Methodology. N. Cross. London, John Wiley & Sons Ltd: 83-97.
- MacGregor, S. P. (2002). "New Perspectives for Distributed Design Support." The Journal of Design Research **2(2)**.
- Maes, P. (1995). "Artificial Life meets Entertainment: Lifelike Autonomous Agents." Communications of the ACM. ACM Press **38(11)**: 108-114.
- Martin, F. J., E. Plaza, et al. (1998). Java Interagents for Multi-Agent Systems. Software Tools for Developing Agents.
- McGee, J. and L. Prusak (1993). Managing Information Strategically: Increase Your Company's Competitiveness and Efficiency by Using Information as a Strategic Tool.
- Mena, E., Kashyap, V., Illarramendi, A., Sheth, A. (1998). Domain Specific Ontologies for Semantic Information Brokering on the Global Information Infrastructure. Formal Ontology in Information Systems. N. Guarino. Amsterdam, IOS Press.
- Merriam-Webster (2003). Merriam-Webster's Collegiate Dictionary, 11th edition, Merriam-Webster, Inc.
- Miller, G. A., C. Fellbaum, et al. WordNet 1.7.1, Cognitive Science Laboratory - Princeton University. **2003**.
- Mulej, M., M. Vezjak, et al. (1999). APPLIED SYSTEMS THINKING AND THE LAW OF REQUISITE HOLISM. 7th Interdisciplinary Information Management Talks, Zadov, Czech Republic.
- Nakakoji, K., Y. Yamamoto, et al. (1998). "From Critiquing to Representational Talkback: Computer Support for Revealing Features in Design." Knowledge-Based Systems Journal **11(7-8)**: 457-468.
- Neches, R., R. Fikes, et al. (1991). Enabling Technology For Knowledge Sharing. AI Magazine. **12**: 36-56.
- Noy, N. F. and C. D. Hafner (1997). "The State of the Art in Ontology Design - A Survey and Comparative Review." AAAI: 53-74.
- Noy, N. F. and D. L. McGuinness (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Stanford, CA, 94305, Stanford University.
- Nwana, H., L. Lee, et al. (1996). "Coordination in Software Agent Systems." BT Technology Journal **14(4)**: 79-88.
- Nwana, H. and M. Wooldridge (1996). "Software Agent Technologies." BT Technology Journal **14(4)**: 68-78.
- Nwana, H. S. (1996). "Software Agents: An Overview." Knowledge Engineering Review **11(3)**: 1-40.
- Nwana, H. S. and D. T. Ndumu (1999). A Perspective on Software Agents Research. Ipswich, British Telecommunications Laboratories.

- Odell, J. (2000). Agent Technology - Green Paper, OMG - Agent Platform Special Interest Group.
- Oliveira, E., K. Fischer, et al. (1999). "Multi-agent systems: which research for which applications." Robotics and Autonomous Systems 27: 91-106.
- Olsen, G. R., M. Cutkosky, et al. (1994). Collaborative Engineering based on Knowledge Sharing Agreements. 1994 ASME Database Symposium, Minneapolis, MN.
- Ossimitz, G. (1997). The Development Of Systems Thinking Skills Using System Dynamics Modeling Tools. 2003.
- Pahl, G. and W. Beitz (1996). Engineering a Systematic Approach, Springer.
- Pahng, F., N. Senin, et al. (1997). Modeling and Evaluation of Product Design Problems in a Distributed Design Environment. DETC'97: 1997 ASME Design Engineering Technical Conferences, Sacramento, California.
- Patel, U., M. J. D'Cruz, et al. (1997). "Collaborative Design for Virtual Team Collaboration : A Case Study of Jostling on the Web." ACM.
- Pena-Mora, F., K. Hussein, et al. (2000). "CAIRO: a Concurrent Engineering Meeting Environment for Virtual Design Teams." Artificial Intelligence in Engineering 14: 202-219.
- Pollard, E. and H. Liebeck (2000). The Oxford Paperback Dictionary. New York, Oxford University Press Inc.
- Poslad, S., P. Buckle, et al. (2000). The FIPA-OS Agent Platform: Open Source for Open Standards. Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, UK.
- Pugh, S. (1991). Total Design: Integrated Methods for Successful Product Engineering, Addison-Wesley Publishing UK.
- Ramsdell, J. D. (2000). A Foundation for a Semantic Web.
- Rao, A. S. and M. P. Georgeff (1995). BDI Agents: From Theory to Practice. Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA.
- Richmond, B. (1994). System Dynamics/Systems Thinking: Let's Just Get On With It. International Systems Dynamics Conference, Sterling, Scotland.
- Roche, T. (1999). Development of a Design for the Environment Workbench. CIMRU, Industrial Engineering Dept. Galway, UCG.
- Rosnay, J. d. (1979). The Macroscopic: A New World Scientific System. New York, Harper & Row.
- Rosnay, J. d. (1997). Analytic vs. Systemic Approaches. Principia Cybernetica Web. F. Heylighen, C. Joslyn and V. Turchin, Principia Cybernetica, Brussels. 2003.
- Russell, S. and P. Norvig (2003). Artificial Intelligence: A Modern Approach, 2/E, Prentice Hall.
- Sclater, N., H. Grierson, et al. (2001). "Online Collaborative Design Projects: Overcoming Barriers to Communication." International Journal of Engineering Education 17(2): 189-196.
- Shannon, C. E. and W. Weaver (1963). The Mathematical Theory of Communication. Chicago, University of Illinois Press.
- Shintani, T., T. Ito, et al. (2000). Multiple negotiations among agents for a distributed meeting scheduler. Proceedings of the Fourth International Conference on MultiAgent Systems.

- Shoham, Y. (1998). Agent-oriented programming. Readings in Agents, Elsevier Science. **Artificial Intelligence 60 (1993)**.
- Siemieniuch, C. E. and M. Sinclair (1999). "Real-time collaboration in design engineering: an expensive fantasy or affordable reality?" Behaviour & Information Technology 18(5): 361-371.
- Simon, H. A. (1996). The Sciences of the Artificial. Cambridge Mass., MIT Press.
- Skyttner, L. (1996). "General systems theory: origin and hallmarks." Kybernetes 25(6): 16-22.
- Smith, R. P. and J. A. Morrow (1999). "Product development process modeling." Design Studies(20): 237-261.
- Snow, C. P. (1993). The Two Cultures. Cambridge, Cambridge University Press.
- Sowa, J. F. (2000). Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, CA, Brooks Cole Publishing Co.
- Spyns, P., R. Meersman, et al. (2002). Data Modelling versus Ontology Engineering, ACM SIGMOD Record. **31**.
- Sterman, J. D. (1991). A Skeptic's Guide to Computer Models. Managing a Nation: The Microcomputer Software Catalog. G. O. Barney, W. B. Kreutzer and M. J. Garrett. San Francisco, Westview Press: 209-229.
- Studer, R., V. R. Benjamins, et al. (1998). "Knowledge Engineering: Principles and Methods." Data and Knowledge Engineering 25(1-2): 161-197.
- Swartz, A. and J. Hendler (2001). The Semantic Web: A Network of Content for the Digital City. Proceedings Second Annual Digital Cities Workshop, Kyoto, Japan.
- Thoben, K.-D. (2002). Extended Products: Evolving Traditional Product Concepts. 7th International Conference on Concurrent Enterprising.
- Thoben, K.-D., F. Weber, et al. (2002). "Barriers in Knowledge Management and Pragmatic Approaches." Studies in Informatics and Control 11(1).
- Tomiyama, T. (1994). The Technical Concept of Intelligent Manufacturing Systems (IMS). Tokyo, University of Tokyo.
- Tsvetovaty, M., M. Gini, et al. (1997). "MAGMA: An agent-based virtual market for electronic commerce." Journal of Applied Artificial Intelligence.
- Uschold, M. (1996). Building Ontologies: Towards a Unified Methodology. Expert Systems '96, the 16th Annual Conference of the British Computer Society Specialists Group on Expert Systems, Cambridge, UK, 16-18 December 1996.
- Uschold, M. (1998). "Knowledge level modelling : concepts and terminology." The Knowledge Engineering Review 13(1): 5-29.
- Uschold, M. and M. Gruninger (1996). "Ontologies:Principles, Methods and Applications." The Knowledge Engineering Review 11(2): 93-136.
- Uschold, M. and M. King (1995). Towards a Methodology for Building Ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing" IJCAI-95.
- Van de Riet, R., Burg, H., Dehne, F. (1998). Linguistic Issues in Information System Design. Formal Ontology in Information System. G. Nicola. Amsterdam, IOS Press.
- VanCuilenburg, J. J., O. Scholten, et al. (1991). Stiinta Comunicarii.
- Viano, G. (2000). Adaptive User Interface for Process Control based on Multi-Agent approach. AVI 2000, Palermo, Italy.

- Weber, R. (1997). Ontological Foundations of Information Systems. Melbourne, Coopers and Lybrand.
- Weiss, G. (1999). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. London, MIT Press.
- Werkman, K. J. (1990). Multiagent Cooperative Problem-Solving through Negotiation and Sharing of Perspectives. DAI-List, <http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/pubs/lists/dai-list/dailist/006.10may90>.
- Wooldridge, M. (1998). "Agent-based computing." Interoperable Communication Networks 1(1): 71-97.
- Wooldridge, M. (1999). Intelligent Agents. The MIT Press.
- Wooldridge, M. and P. Ciancarini (2001). Agent-Oriented Software Engineering: The State of the Art. Agent-Oriented Software Engineering. P. Ciancarini and M. Wooldridge, Springer-Verlag. **AI Volume 1957**.
- Wooldridge, M., N. R. Jennings, et al. (2000). "The gaia Methodology for Agent-Oriented Analysis and Design." Autonomous Agents and Multi-Agent Systems **Kluwer Academic Publishers**(3): 285-312.
- Zlotkin, G. and J. S. Rosenschein (1989). Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains. The Eleventh International Joint Conference on Artificial Intelligence, Detroit, Michigan.
- Zlotkin, G. and J. S. Rosenschein (1996). "Mechanism Design for Automated Negotiation, and its Application to Task Oriented Domains." Journal of Artificial Intelligence 86(2): 195-244.

Annex 1

The Semantic Web

- 1. Introduction**
- 2. Background**
- 3. The structure of the Semantic Web**
- 4. Final Remarks**

1. Introduction

One of the most successful stories of the information age is the story of the World Wide Web (WWW). The WWW was developed in 1989 by Tim Berners-Lee to enable the sharing of information among geographically dispersed teams of researchers within the European Laboratory for Particle Physics (CERN). The simplicity of publishing on WWW and the envisioned benefits attracted an increasing number of users from beyond the boundaries of the research community. The WWW grew rapidly to support not only information sharing between scientists (as it was intended), but to support information sharing among different kind of people communities, from simple homepages to large business applications. The Web became an “universal medium for exchanging data and knowledge: for the first time in history we have a widely exploited many-to-many medium for data interchange” [Decker, Harmelen et al. 2000]. The WWW is estimated to consist from around one billion documents and more than 300 millions users access them, and these numbers are growing fast [Fensel 2001; Benjamins, Contreras et al. 2002]. Soon, as a “medium for human communication, the Web has reached critical mass [...] but as a mechanism to exploit the power of computing in our every-day life, the Web is in its infancy” [Connolly 1998; Cherry 2002]. On one hand it became clear that while the Web enables human communication and human access to information it lacks of tools or technologies to ease the management of Web’s resources [Fensel 2000; Fensel 2001; Palmer 2001]. On the other hand it is impossible to build *semantical* tools that will spot and know the difference between for example *a book by* and *a book about*. But also from the human user point of view, the WWW has become an immense haystack of data [Ewalt 2002]. The time has come “to make the Web a whole lot smarter” [Connolly 1998], so dynamic generated data (e.g. data from pages generated from databases) can join the Web [Benjamins, Contreras et al. 2002; Hendler, Berners-Lee et al. 2002]. In other words it is time to upgrade the Web from “giving value to human eyeballs” to a Web where “the interesting eyeballs will belong to computers” (Prabhakar Raghavan, chief technology officer at Variety Inc as cited by [Ewalt 2002]).

2. Background

The Semantic Web (SW) is an emerging concept that launches the idea of having data on the web defined and linked in a way that it can be used by people and processed by machines [Berners-Lee 1998; Decker, Harmelen et al. 2000; Fensel 2000; Ramsdell 2000; Berners-Lee, Hendler et al. 2001; Dumbill 2001; Swartz and Hendler 2001; Hendler, Berners-Lee et al. 2002] in a “wide variety of new and exciting applications” [Swartz and

Hendler 2001]. It develops “languages for expressing information in a machine processable form” [Berners-Lee 1998], so to enable the machine to be able to participate and help inside the information space [Benjamins, Contreras et al. 2002]:

“The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users.” [Berners-Lee 1998]

The SW will not be a separate Web, but the extension of the current one [Berners-Lee, Hendler et al. 2001; Swartz and Hendler 2001; Ewalt 2002; Hendler, Berners-Lee et al. 2002]. The WWW is primary a medium of documents for people rather than a medium of data and information than can be processed automatically. The SW will upgrade it to a new medium adequate for both people and machines [Berners-Lee, Hendler et al. 2001; Benjamins, Contreras et al. 2002; Westoby 2003]. The new environment will be more effective for its users by automating or enabling the processes that are currently difficult to perform: “locating content, collating and cross-relating content, drawing conclusions from information found in two or more separate sources” [Dumbill 2001]. These objectives are made possible/reachable by giving structure to the rich information contained in documents all over the Web [Berners-Lee, Hendler et al. 2001].

The goals of the SW related research are summarized by Koivunen as follows [Koivunen 2001]:

1. “Design the technologies that support machine facilitated global knowledge exchange.”
2. “Making cost-effective for people to record their knowledge.”
3. “Focus on machine consumption.”

After the WWW any new web-based medium (no matter if it will upgrade or replace WWW) has to fulfil a new set requirements used for exchanging data on the web [Decker, Harmelen et al. 2000]:

1. *Universal expressive power*: meta-data languages should be enabled to express any kind of data.
2. *Support for Syntactic Interoperability*: the structures that represent data should be easily readable so applications (such as parsers) can exploit them.
3. *Support for Semantic Interoperability*: unknown data should come with semantics, or it should be possible to be mapped to known data.

Once the goals and the requirements have been identified, a set of principles – emerged from the advantages, the mistakes and the shortcomings of the WWW - to guide the SW

research and development have been proposed by the World Wide Web Consortium (W3C) as follows [Koivunen 2001; Westoby 2003]:

Principle 1: Everything can be identified by Uniform Resource Identifiers (URI).

This principle gives a measure of the things that can be a part of SW, that is anything conceivable. From physical objects to human beings and from simple words to complex conceptual structures, everything can be on the SW as long as an URI have been associated to it. There is no restriction concerning the permissible part of the Web URI namespace to be used by public.

Principle 2: Resources and links can have types.

The first impression a user may have about the WWW is its vastness and derived from here its complexity. In fact the WWW structure is quite simple, consisting in a collection of *resources* (e.g. web documents) and *links* that bind different resources to each other (see Figure 1a).

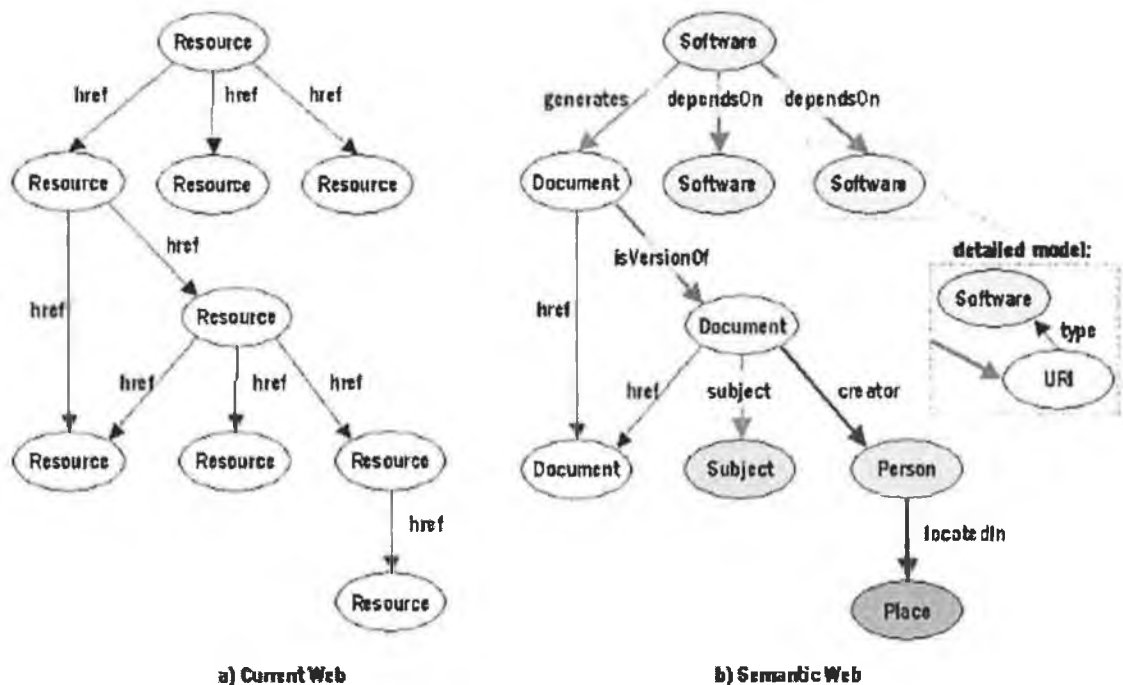


Figure 1. WWW structure vs. SW structure [Koivunen 2001]

On one hand this kind of structure (because of its simplicity) is easy to be implemented, so almost everyone can create, publish and link resources. On the other hand (because of its poor granularity) its difficult if not impossible to know beforehand what a resource refer at and what is a meaning of a link. This result in a lack of automated tools to help humans locate, use or share a specific resource. The SW structure, in turn, gives the possibility of typing the resources and links (see Figure 1b), so more information about resources and link to be available beforehand. In this way it enables the development of tools that would

automate some if not all of the web-related process (i.e. finding information, sharing and reuse).

Principle 3: Partial information is tolerated.

One of the most important characteristics of the web is scalability, which is closely related to the ability of dynamic evolving of the WWW. Because of it the WWW grew beyond any expectation, becoming one of the most important communication/business environment of today's world. The price paid for this is the link integrity [Koivunen 2001]. This means that resources from the WWW may appear, evolve (e.g. change its content while keeping the same URI) and disappear dynamically without any possibility for the links that point to such resources to be "announced". That is why there are links that point to inexistent locations (also called 404 links) and links that point to wrong resources. But this is a price worth paying. For these reasons, SW will also have to deal with the resource's life cycle (e.g. creation, changing, decaying) and the tools built for SW should tolerate it and function in this kind of dynamic resources.

Principle 4: There is no need for absolute truth.

As in today's WWW, there is nothing to apriory guarantee the truth or the value of truth of some information within SW. No matter if the particular resource is data or information or knowledge, nobody and nothing should enforce some kind of rules to guarantee that resource is true in some particular system. The value of truth should remain at application level. This means that the particular application - based on some kind of label-kind of information about the resource and about the place it came from - should decide how trustworthiness an input is. In this way the principles 3 and 4 implement in SW one of the most cherished principle of WWW: the freedom of information.

Principle 5: Evolution is supported.

Generally speaking information evolves as human understanding evolves. This means that it (i.e. information) sustains a progressive change and development. SW has to be able to deal with this phenomenon by enabling processes such as some kind of version control (i.e. adding information without having to change or delete the old one), translation among different communities (i.e. translation from one language to another, synonymy – "postal code" is equivalent to "ZIP code", and so on), combination of information that may arise from different/distributed resources, and so on [Koivunen 2001].

Principle 6: Minimalist design.

"The Semantic Web makes the simple things simple, and the complex things possible" [Koivunen 2001] by standardizing no more than is necessary with the desiderate that "result should offer much more possibilities than the sum of the parts"[Koivunen 2001].

3. The structure of the Semantic Web

The general structure/schema of the SW, also known as *the layer cake*, developed by Tim Berners-Lee, the inventor of WWW, presents the most important elements of the system. The foundation consists of new web languages such as metadata languages (e.g. XML and RDF) and furthermore, to languages which allow ontologies, rules, proofs and logics to be realised at a web-wide scale [Swartz and Hendler 2001](Figure 2):

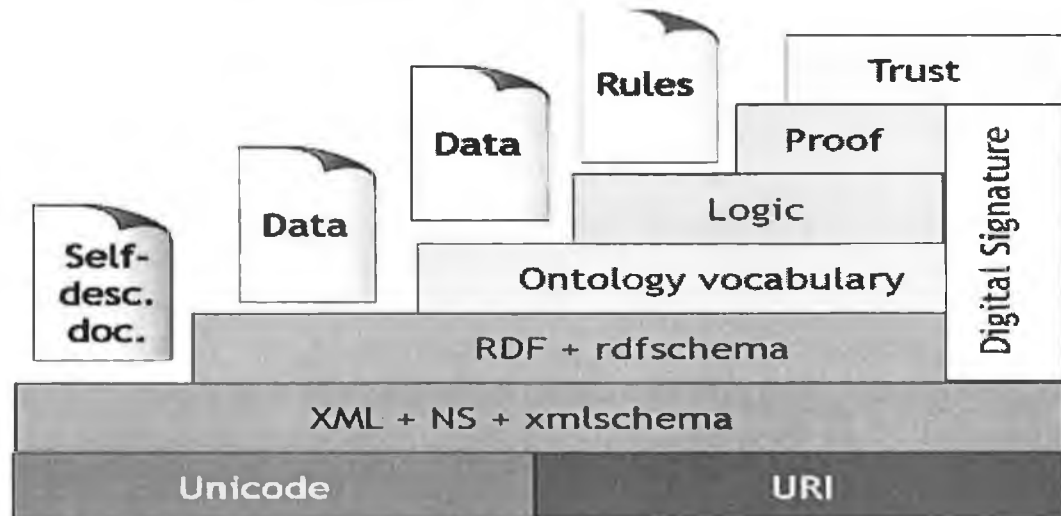


Figure 2. *The Layer Cake* after Tim Berners-Lee [Koivunen 2001; Swartz and Hendler 2001]

This layered structure of the SW is not the definitive model of the SW, but is intended to be a prototype, an idealized diagram. It is designed in such way that “each layer gives progressively more value” [Dumbill 2001]. The architecture of the SW starts with the foundation of URIs and Unicode that provide the following layers with an alphabet (i.e. Unicode) and a technology for identifying resources on web (i.e. URI). The XML layer adds the syntactic interoperability and introduces the RDF + rdfschema layer as the data interoperability layer [Dumbill 2001]. The ontology layer deals with the description of objects and the relations among them and is the key layer of the attempt to achieve a shared meaning of a domain of interest. The logic layer provides languages for enabling reasoning on data as they are structured in the lower levels, while the proof layer offers techniques for describing the steps taken for reaching a conclusion from the facts. On the top of the architecture lies the trust layer that provides the means of weighing the value of information, of deductions made and so on. The Digital Signature layer accompanies all the layers that are dealing with data/information/knowledge and represents a way of assuring the provenance of a certain resource. Each of these layers will be detailed in the followings.

Unicode

The Unicode establishes the alphabet of any information system. It has exactly the same role as, for example, the alphabet of the English language, i.e. it sets a set of adequate characters to be used for constructing words and sentences. Because the computers are working with numbers, the Unicode provides a unique number for every character, independent of platform, application or language. Hence, Unicode is a “character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages of the modern world” [Consortium].

Uniform Resource Identifier (URI)

“A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource” [Berners-Lee, Fielding et al. 1998]. In other words an URI is simply a web identifier [Palmer 2001; Swartz 2002].

To paraphrase the World Wide Web Consortium (W3C), the Internet space is inhabited by many points of content. An URI is the way you identify any of these points of content, whether it is a page of text, a video or sound clip, an image, or a program. An URI typically describes [Berners-Lee, Fielding et al. 1998]:

- The mechanism used to access the resource
- The specific computer where the resource is
- The specific name of the resource on the computer

The URI is the foundation of the Web [Swartz and Hendler 2001]. Anything that has an URI is *on the Web* and anything can have a URI. One of the most familiar form of URI is the Uniform Resource Locator (URL), that is the address of a web page, like: <http://pan.nuigalway.ie/PublicDocuments/>, which lets any computer with a browser and access to Internet to locate a specific resource (in this case the public documents from the computer named *pan* from National University of Ireland Galway network). The syntax of URI's is governed by IETF, who published RFC 2396 [Berners-Lee, Fielding et al. 1998] as the general URI specification. The W3C maintains a list of URI schemes at <http://www.w3.org/Addressing/schemes>.

Extensible Markup Language (XML)

XML [<http://www.w3c.org/XML/>; Bray, Paoli et al. 2000] was designed to be a simple way of sending *meaningful* documents across the Web and it is considerate to be the standard for data interchange on the Web. It provides a syntax for structuring and

granulating data according to specific needs [Westoby 2003], such as having a *book* structure described by its *name*, *author*, *publisher* and *ISDN*, as follows:

```
Book {Name;  
      Author;  
      Publisher;  
      ISDN;}
```

XML allows anyone to create arbitrary document layout/structure in which to write his/her own document [Decker, Harmelen et al. 2000; Swartz and Hendler 2001; Cherry 2002]. XML “is the evolutionary successor to HTML” [Connolly 1998] and is enhancing HTML (the standard mark-up language for WWW) by adding structure to the document. In a HTML document the information is “structured and shared in forms that facilitate its display for human consumption” [Ramsdell 2000] (e.g. sections, paragraphs, lists, tables), but this structure does not provide any help to software tools [Heflin 2001]. In turn, a XML document gives a way of building software tools that understand the information it contains. Of course, a software program will not truly *understand* the document, but it is easy to code software tools to work with XML documents in more specific contexts than with HTML documents. Also it will be easier for a human to read and understand a XML document than a HTML one. For example: in “The Analytical Language of John Wilkins” Borges describes a probably inexistent Chinese encyclopaedia called the *Celestial Emporium of Benevolent Knowledge*, which divides animals into classes, as illustrated in figure 3.

“On those remote pages it is written that animals are divided into:

1. those that belong to the Emperor
2. embalmed ones
3. those that are trained
4. suckling pigs
5. mermaids
6. fabulous ones
7. stray dogs
8. those included in this classification
9. those that tremble as if they were mad
10. innumerable ones
11. those drawn with a very fine camel's hair brush
12. others
13. those that have just broken a flower vase
14. those that resemble flies from a distance.”

Figure 3. A classification of animals (from [Borges 1984]).

Publishing this typology on the Web using the standard web language (i.e. HTML) would require something like the following code:

```
<p> On those remote pages it is written that animals are divided into:</p>
<ol>
  <li>those that belong to the Emperor </li>
  <li>embalmed ones </li>
  <li>those that are trained </li>
  <li>suckling pigs </li>
  <li>mermaids </li>
  <li>fabulous ones </li>
  <li>stray dogs </li>
  <li>those included in this classification </li>
  <li>those that tremble as if they were mad </li>
  <li>innumerable ones </li>
  <li>those drawn with a very fine camel's hair brush </li>
  <li>others </li>
  <li>those that have just broken a flower vase </li>
  <li>those that resemble flies from a distance. </li>
</ol>
```

Here is the document marked-up using XML:

```
<example>
  In
  <book>
    <title>The Analytical Language of John Wilkins</title>
    <author>J. L. Borges </author>
  </book>
  <cite>On those remote pages it is written that animals are divided into: </cite>
  <animal>
    <classes>
      <class id=1> those that belong to the Emperor</class>
      <class id=2> embalmed ones </class>
      <class id=3> those that are trained </class>
      <class id=4> suckling pigs </class>
      <class id=5> mermaids </class>
      <class id=6> fabulous ones </class>
      <class id=7> stray dogs </class>
      <class id=8> those included in this classification </class>
      <class id=9> those that tremble as if they were mad </class>
      <class id=10> innumerable ones </class>
      <class id=11> those drawn with a very fine camel's hair brush </class>
      <class id=12> others </class>
      <class id=13> those that have just broken a flower vase </class>
      <class id=14> those that resemble flies from a distance </class>
    </classes>
  </animal>
</example>
```

The items located between the signs < and > in XML version are called tags. A full set of tags (including the opening and closing tags) plus their content is called an *element* and descriptions as *id=6* are called *attributes*. It is obvious that in the XML document the

information is more flexibly structured (i.e. a basic XML data-model consists of a labelled tree) not only for humans to understand what a certain document is referring to, but also for software programs to work with. While in HTML is impossible to say something about the content of the document (this is possible only if a human *knows* about the context of which the document is part of), a lot about it can be extracted solely from the structure of tags in XML document. Furthermore constraints on tags may be enforced (e.g. constraints on the range of values or on the types some attributes/tags may have) through the use of Document Type Definition (DTD) files. Because of some technical limitations of DTD's, W3C proposes the use of XML Schema instead. Besides a number of advantages the main role of XML Schema is the same with the DTD, and this is to define a grammar for XML documents [Decker, Harmelen et al. 2000].

Through the concept of "XML Namespaces" each element and attribute has an URI associated. Anyone can create XML tags and the correspondent URIs and mix them with tags created by others. In this way a *low level* sharing and reuse of structured data (i.e. information) becomes possible. Generally, XML helps humans and software programs to predict what information might lie "between the tags" (this depends on the design skills of the human who creates the tags), but XML can only help. However, the XML data-model presents some disadvantages when used in the context of SW. For an XML processor, <sentence> and <p> and <animal> are all equally (and totally) meaningless. Moreover, different developers may choose different words for expressing the structure of same data (e.g. <author> vs. <authorname>, <classification> vs. <taxonomy>). This has direct consequences for the average ability of the software tools that can be designed for working with XML documents. And this is because XML cannot add semantics to data (cannot convey an arbitrary meaning) [Heflin 2001; Westoby 2003].

Resource Description Framework (RDF)

Collaboration on the Web, distributed information and knowledge, and in general any application designed for a distributed environment requires rich data. Data-exchange flow is currently very limited, consisting of tab-delimited dumps or product-specific tables. Specific XML formats for each exchange task improves the situation, but is far away from solving the problem because the XML data model is too low-level [www.semanticweb.org]. Usually the humans have to browse, filter and process the received data. After that the result has to be coded into specific structures that would allow some specific application to use it as an input. After the input data is processed, an output flow is generated. Usually this flow would be automatically coded in a document of some

sort or would require further processing from the humans in order to be used as input data for another application. This need for human's intervention is considered as a waste of resources. For this reason, the need for a new data model paradigm that would allow applications to exchange data semantics without human intervention has become stringency. Resource Description Framework (RDF) [Lassila and Swick 1999] implements such a meta-data model that "gives a way to make statements that are **machine-processable**" [Fensel 2000; Swartz and Hendler 2001], and its motivation is to provide a standard for semantical description of resources on the Web [Decker, Harmelen et al. 2000; Palmer 2001; Cherry 2002]. Of course, the computer will not truly *understand* a statement, but "it can deal with it in a way that seems like it does" [Swartz and Hendler 2001].

In the SW Activity Statement [<http://www.w3.org/2001/sw/Activity>], RDF is viewed as the language designed for the SW in the same way that HTML is the language of WWW. Moreover "RDF is an infrastructure that enables the encoding, exchange and reuse of structured metadata" [Fensel 2000].

RDF follows the W3C design principles [<http://www.w3.org/Consortium/#web-design>]:

1. *Interoperability*: Specifications of the Web's languages and protocols must be compatible with one another and allow (any) hardware and software used to access the Web to work together.
2. *Evolution*: The Web must be able to accommodate future technologies. Design principles such as simplicity, modularity, and extensibility will increase the chances that the Web will work with emerging technologies such as mobile Web devices and digital television, as well as others to come.
3. *Decentralization*: Decentralization is without a doubt the newest principle and most difficult to apply. To allow the Web to "scale" to worldwide proportions while resisting errors and breakdowns, the architecture (like the Internet) must limit or eliminate dependencies on central registries.

A RDF statement is like a simple sentence, except that instead of words it uses URIs. The basic RDF model contains only two concepts [Berners-Lee 1998]:

1. **Assertion** - a positive statement or declaration (often without support or reason);
2. **Quotation** - an assertion about assertion (it comes from RDF's property of being a data about data, i.e. metadata, language)

Each RDF statement is a triplet subject-predicate-object or object(O)-attribute(A)-value(V) [Decker, Harmelen et al. 2000; Westoby 2003]. This triplet is commonly written as A(O,V), such as in the following example:

ReallyLikes(Camelia, to paint)

Syntactically speaking, anything that has an URI can be a subject, or a predicate, or an object, but for the RDF statement to make sense the semantic aspect is also important. In a RDF the predicate links the object to subject or, in real words it says something about something. The figure 4 shows an example meaning: *Camelia really likes to paint*.

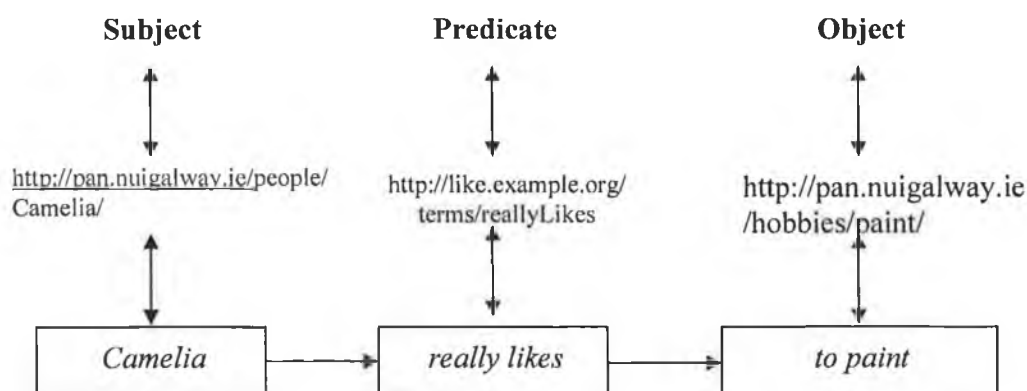


Figure 4. The Subject-Predicate-Object triplet.

Anyone can create information and label it through the use of URIs and write RDF statements to give sense to the separate pieces of information. Moreover, because of the data-oriented programming paradigm that dominates the world of IT, there are thousands of databases containing machine-processable information. Usually, the data contained in these databases is used only locally not because of the technological limitations, but because of the wide variety of ways of coding data in databases. The primary purpose of RDF is to provide a data-model for meta-data that would enable the description of resources in a standard manner without making any assumptions about a particular application domain. In this way it is possible to built intelligent programs that can “begin to fit the data together” [Swartz and Hendler 2001] from the various databases, according to specific needs. The RDF data model, however, “provides no mechanisms for declaring these properties, nor does it provide any mechanisms for defining the relationships between these properties and other resources” [Brickley and Guha 2002], where properties represent attributes of resources or relationships between resources. This is the place where RDF Schema comes into action. RDF Schema is for RDF what XML Schema is for XML [Decker, Harmelen et al. 2000], which means that it “defines not only the properties of the resource (e.g., title, author, subject, size, colour, etc.) but may also define the kinds of

resources being described (books, Web pages, people, companies, etc.)” [Brickley and Guha 2002] (e.g. with RDF Schema one can say that "Dalmatian" is a type of "Dog", and that "Dog" is a sub class of animal).

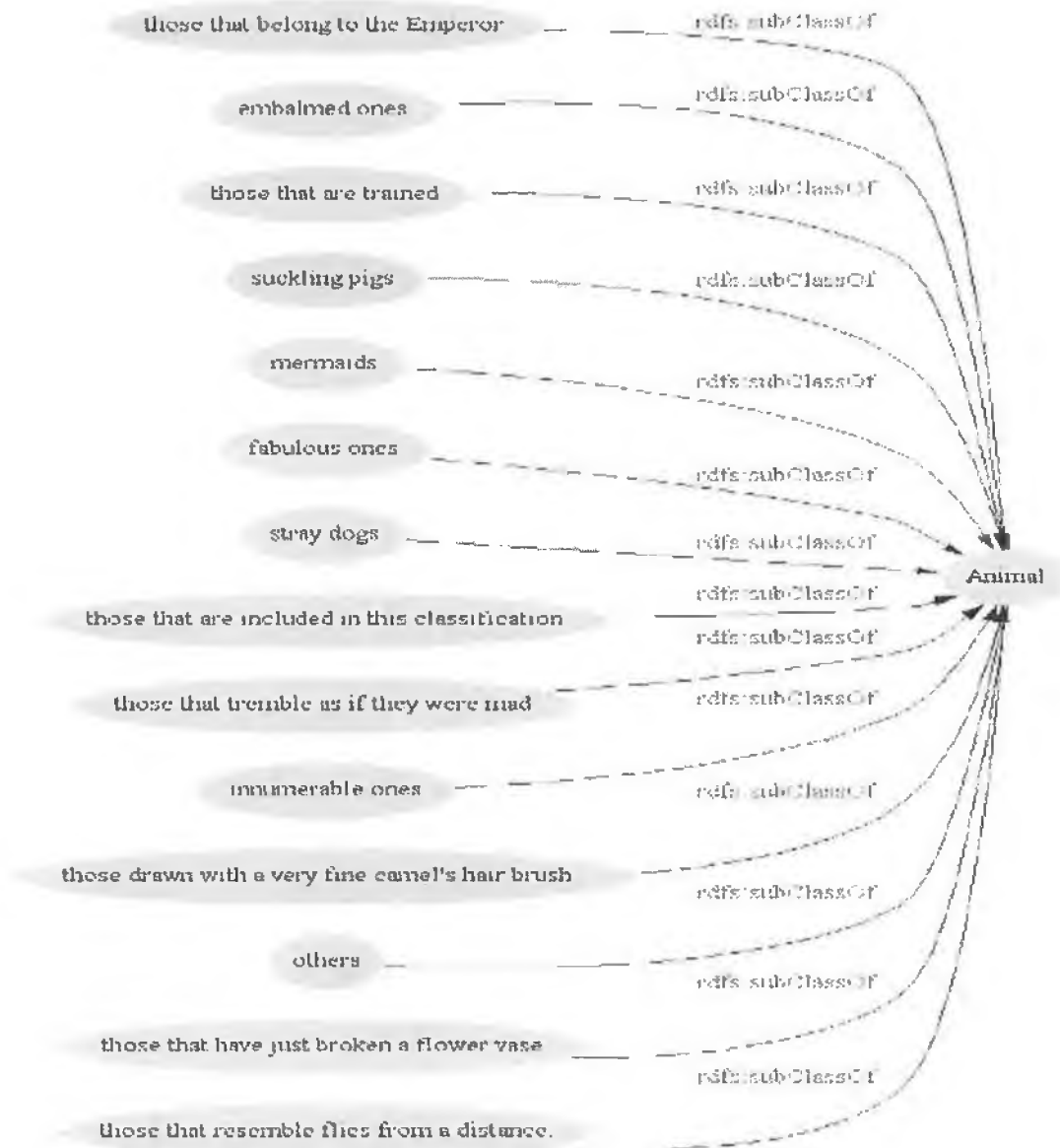


Figure 5. RDF model of a simple vocabulary [Koivunen 2001]

For example (see Figure 3) the RDF model of a simple classification uses the "subClassOf" property/predicate to relate every category/subject to the object being classified (see Figure 5) [Koivunen 2001]. In this way, a structured model of the information about the classification of animals is produced. The RDF source code [<http://www.w3.org/2001/09/01-borges/taxon.rdf>] that implements the model is provided in the appendix. This kind of *machine-readable* information can be accessed directly by

applications (without human intervention), can be integrated with other resources (similarly encoded) and processed. The result will also be encoded in RDF style so it can be further used/reused.

Generally, RDF is accepted as being the language for representing formal data on the Web [Decker, 2000 #83; [Palmer 2001]], with the specification that it has to be enriched so it can represent any kind of complex data structure.

Ontologies

In order to build programs that parse RDF modelled information from databases or documents, it has to be assumed that data is nearly perfect modelled [Berners-Lee, Hendler et al. 2001; Swartz and Hendler 2001]. This means that every concept is uniquely defined and interpreted in the same way by all the members of the web community. This assumption is not possible and comes in contradiction with the freedom that Web provides and should provide. Moreover, there is no way for a computer or human to figure out what a specific term means, or how it should be used. The ontologies (see subchapter 1) overcome these difficulties by providing a way to describe the meaning and the relationships of terms [Gruber 1993; Guarino 1997; Fensel 2000; Swartz and Hendler 2001; Ding, Fensel et al. 2003] so that a shared understanding or a consensus to be reached among people and machines. The most typical Web ontology has a taxonomy and a minimal set of inference rules [Hendler 1999; Swartz and Hendler 2001; Cherry 2002] and usually represents a 'small' specialised world of a community of interest or a Domain Model [Decker, Harmelen et al. 2000; Ding, Fensel et al. 2003]. The SW is predicted/envisioned to be an anarchic web of such small ontologies created by different communities and referenced by/pointed to each other in the way the documents on the current web are [Hendler 1999; Fensel 2000].

At the ontology layer the world of discourse is created by defining the classes of objects and the relationships among them [Berners-Lee 1998; Fensel 2000; Berners-Lee, Hendler et al. 2001]. This description, in a RDF-based language, gives the computers the ability to identify equivalent concepts (meanings) even if different communities use different identifiers (situation that will most probably arise) to point to them [Berners-Lee, Hendler et al. 2001; Swartz and Hendler 2001].

Ontologies establish a joint terminology between members of a community of interest [www.semanticweb.org]. These members can be human or automated agents. To represent a conceptualisation a representation language is need. Several representation languages for representing meaning and structure content [Benjamins, Contreras et al. 2002] have been defined (usually XML-based and RDF-based – see Figure 6), as follows: SHOE, Ontology

Exchange Language (XOL), Ontology Markup Language (OML and CKML), Resource Description Framework Schema Language (RDFS), and Riboweb. A new proposal extending RDF and RDF Schema, which will most probably enforce the new standard in creating ontologies, is DARPA Agent Markup Language with Ontology Inference Layer (DAML+OIL) jointly developed by a group of scientists from Europe and United States of America.

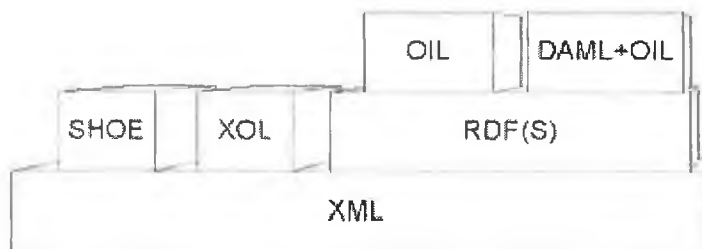


Figure 6. Semantic Web languages pyramid [Benjamins, Contreras et al. 2002].

The ontology is a very powerful concept of the SW because allow and require information to be structured and organised in classes, subclasses and relations (such as inheritance and equivalence) [Hendler 1999; Fensel 2000; Berners-Lee, Hendler et al. 2001]. Furthermore, the inference (i.e. deriving new data from data that is already know [Palmer 2001]) rules added to the logic of the ontologies supply further power.

Using a representation language based on the RDF data-model anyone can create a world of discourse/RDF schema/ontology. For the entire system (i.e. Semantic Web) to function, an important requirement is the existence of a minimum set of rules for converting a document in one RDF schema into another one [Berners-Lee 1998; Westoby 2003]. For example a schema might state that (after [Swartz and Hendler 2001]):

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

# A creator is a type of contributor:
dc:creator rdfs:subClassOf dc:contributor .
```

If some application wants to gather the authors and the contributors to various papers it uses this vocabulary to understand the information it finds. If a newcomer will want to create RDF documents, he/she will probably not know about dc:creator, so he/she will make up his own term, such as ed:hasAuthor.

The existing term:

```
<http://aaronsw.com/> is dc:creator of  
  <http://logicerror.com/semanticWeb-long>
```

The new term:

```
<http://logicerror.com/semanticWeb-long>  
  ed:hasAuthor <http://aaronsw.com/> .
```

Normally, the original program would simply ignore these new statements, since it can't understand them. However, it is a possibility to bridge the gap between these two worlds, by providing information on how to convert between them:

```
# [X dc:creator Y] is the same as [Y ed:hasAuthor X]  
  dc:creator daml:inverse ed:hasAuthor .
```

Since the program understands DAML ontologies, now it can take this information and use it to process all of the hasAuthor statements it couldn't understand before [Swartz and Hendler 2001].

Some applications of this level (layer) of SW can be summarized as follows [Berners-Lee 1998; Berners-Lee, Hendler et al. 2001]:

- Cross-linking different databases independently created and published on the Web by semantic links that allow queries on one database to be automated converted into queries on other database;
- Improving the accuracy of the Web searches (the search program is looking at the pages that are referring a specific concept and not some ambiguous keywords);
- Relating information found on some Web page to the associated knowledge structures and inference rules.

Logic

The logical layer brings the power of logic into SW. It enables (together with the proof layer) SW applications to make assumptions and prove them. In other words it allows the computer to make inferences and deductions [Palmer 2001; Swartz and Hendler 2001], such as:

If *a* implies *b* and *b* implies *c* then ***a* implies *c***.

or

My dog's name is Lucky and

Dog is an animal

Then **Lucky is an animal**.

In general, from a popular point of view, the term *semantic* in the SW context is understood as being “machine processable” or “machine understandable” [W3C]. Farrugia stresses that this is too limiting (since to a certain extent this is already achieved) and the term *semantic* should be understood also as “logical (model-theoretic) semantics” [Farrugia 2001]. A model theory is “a formal semantic theory which relates expressions to interpretations” [W3C]. This technique is used for “specifying the semantics of a formal language”, that is linking the use of terms of the formal language to their definitions, and its main utility is “to provide a technical way to determine when inference processes are valid, i.e. when they preserve truth” [W3C]. Generally, a logic consists of “a syntax (or deductive system) and an appropriate semantics (or model theory)” [Farrugia 2001].

The logic that has to be added to SW documents should allow the followings [Berners-Lee 1998]:

- rules of deduction of one type of document from a document of another type;
- checking of a document against a set of rules of self-consistency;
- resolution of a query by conversion from terms unknown into known terms.

For all of these to be possible, Berners-Lee identifies two sub-layers of the logical layer (**quotation** already being in the ontology representation language as presented at the RDF layer) [Berners-Lee 1998]:

1. **Predicate logic** layer – introduces logical operators (e.g. not, and, or, xor)
2. **Quantification** layer – introduces the universal quantifier (e.g. for all *x*, *y*(*x*))

Moreover, Farrugia points out that is important for SW to support not only one logic, but different logics (depending on the particular purposes and goals of a specific application) to be used for different kinds of reasoning (see Figure 7)

For example, the main focus concerning the logic layer of the SW is to implement a logic (Partial First Order Logic) that would allow the *reasoning* application to verify that “one concept or class subsumes another” [Farrugia 2001; Westoby 2003]. But there are other kinds of reasoning such as to reason about “what might possibly be the case” [Farrugia 2001]. This could be supported if specific modal operators (from modal logic) dealing with possibility and necessity are added to the classical propositional logic [Farrugia 2001].

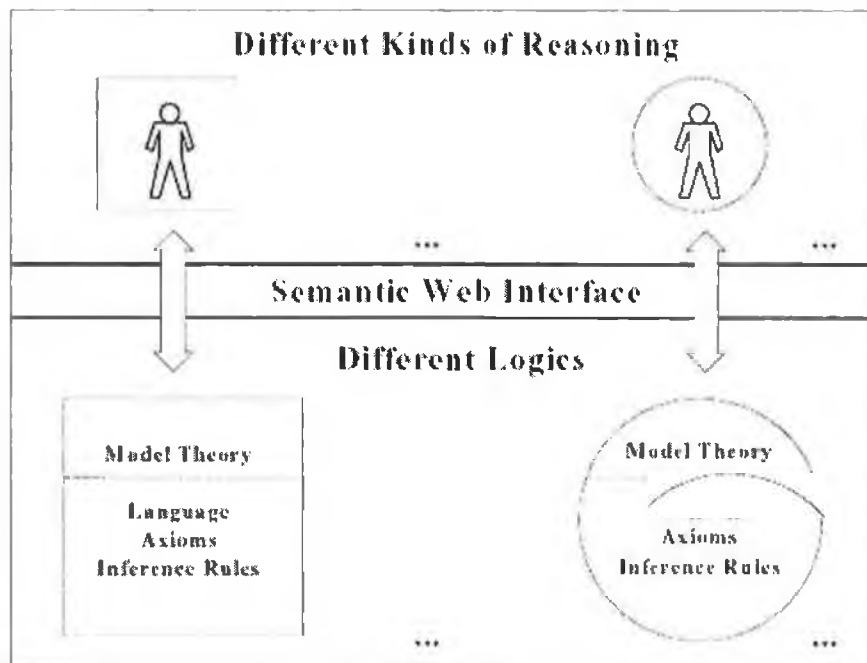


Figure 7. Different logics for different kinds of reasoning [Farrugia 2001]

Proof

Once systems that follow logic, are built, “it makes sense to use them to prove things” [Swartz and Hendler 2001]. Different Web communities can write logic statements. A program can follow these semantic links to begin to prove facts. For example:

Fact: Lucky is an animal.

Proof: 1. Dog is an animal.

2. Lucky is a dog.

Of course, this is a simple example. In a real application to prove a fact could require to follow thousands of links, which is a difficult task, but to check the proof becomes very easy. This will allow “to build a Web of information processors” [Swartz and Hendler 2001]:

“Some of them could merely provide data for others to use. Others would be smarter, and could use this data to build rules. The smartest would be heuristic engines, powering “intelligent agents” which follow all these rules and statements to draw conclusions, and place their results back on the Web as proofs as well as data or query answers like those shown in the introduction.” [Swartz and Hendler 2001]

Trust

One of the main principles of the Web and inherited by the SW is the freedom of the Web participant. This can be translated in anyone can say anything. The negative aspect, or the price to be paid is the problem of trust: whom to trust? The answer to this problem is the concept of trust.

A human or an agent will trust data coming from verified sources. Anything else can be considerate suspicious and sent to further analyses. Because it is almost impossible to directly trust enough *data makers* for a fairly complex application to work, the concept of “Web of Trust” [Swartz and Hendler 2001] was developed. This concept can be represented as an oriented graph where the vertices are people or companies on the Web and the weights of the edges are the degrees of trust (see Figure 8). The graph is constructed starting with a *generator* vertice that is the person/company, which generated the *trust measuring*. The following vertices to be added are the persons (or digital signatures) in which the *generator* trusts with the specific degrees of trust labelling the edges. After that for each added vertice new vertices will be added corresponding to the trust of that vertice, and so on, as shown in figure below.

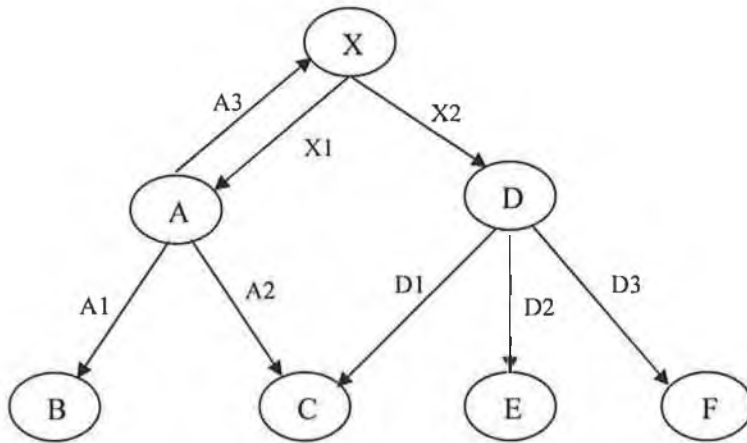


Figure 8. Graph representation of the Web of Trust

Explanation: X trusts - A with the trust degree X1,
- D with the trust degree X2,
A trusts - B with the trust degree A1,
- C with the trust degree A2,
- X with the trust degree A3,
D trusts - C with the trust degree D1,
- E with the trust degree D2,
- F with the trust degree D3,
and so on ...

In addition to trust, a Web of Distrust can also be built. The reason is to differentiate between new information and false information. New information, with no *trust path* possible to be found, identified by an application may be trusted more than information known to be false [Swartz and Hendler 2001].

The application may now take all the factors into account when deciding how trustworthy a piece of information is, but it can present to the user (a human or another application) a simple or complex explanation, so the user can decide about the trustworthiness of the information. [Swartz and Hendler 2001].

Digital Signature

Because on the Web anybody can say anything, it is also possible that anybody can pretend to be somebody else. This is also true in the case of information. For different reasons, the source of some information can be advertised to be different than the true one. This uncertainty can lead to a general atmosphere of distrust inside the web communities with direct consequences on the development of the SW because of its data-oriented design. SW depends on data and this is why fabricated data may lead to serious malfunctions.

The answer to this problem is the Digital Signature that provides proof that a certain person wrote or agrees with a document or statement [Berners-Lee, Hendler et al. 2001; Swartz and Hendler 2001; Westoby 2003]. Digitally signing all the RDF statements makes possible a way of knowing which data or knowledge to trust and which not.

The Digital Signature (“DSig”) was proposed by the W3C Digital Signature Working Group (www.w3.org/DSig/Overview.html) as a “*standard format for making digitally-signed, machine-readable assertions about a particular information resource*”. DSig project provides a mechanism to make the statement such as:

signer believes *statement* about *information resource*

The Digital Signature is a must for development of the SW. That is why it is involved in the design of almost all layers of the SW architecture.

4. Final Remarks

Generally, when speaking about data semantics within the SW, only the textual forms of hypermedia are being considered [Westoby 2003]. This is somehow odd in an environment that would inherit a very rich Web of images, sounds and videos. SW also has to take into

account the semantic markup of all multimedia data, being that the audio-video content is at least equally expressive with the text content. An example of semantic markup for an image/picture, as pictured by Les Carr [Westoby 2003], is presented in Figure 9.

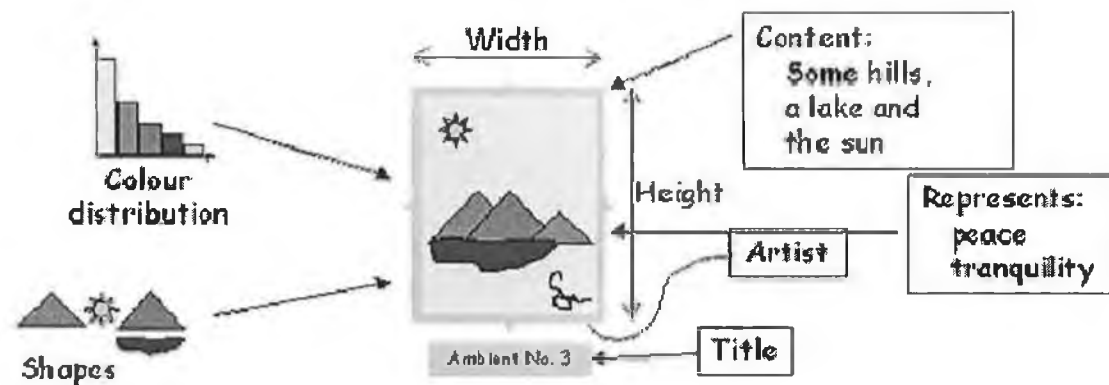


Figure 9. A painting example and associated metadata [Westoby 2003]

As seen in the figure a multimedia metadata has to consider defining different *classes* of data (e.g. colour metadata, shape metadata, text metadata, and so on). The markup is similar for audio and video semantics [Westoby 2003]. Moreover, SW is also interested in all kinds of data, which is anything that can be published in the Web. In other words anything that can be identified by an URI (i.e. is a resource) will be markup-ed in SW style, e.g. databases, services, applications [Hendler, Berners-Lee et al. 2002], address books, organization charts, newsgroups [Cherry 2002]. In the far future is predicted that SW will be technologically enabled to deal with even more resources such as sensors, personal devices and household applications [Hendler 1999; Hendler, Berners-Lee et al. 2002; Westoby 2003].

While the SW can represent the solution to the stringent problem of information management in an interoperable environment, it is not yet functional. Anyway, The first three layers (i.e. Unicode layer, URI layer and XML + XML Schema layer) are functional and important advantages have been made towards the definition and implementation of the RDF + RDF Schema layer and ontology layer. Moreover, its implementation and development does not depend solely on the researchers that are investigating it. In order to achieve a *Semantic Web* it is essential to attract a critical mass of users to exploit it (e.g. publish meta-data, create applications for SW, and so on). In the case of WWW this happened because of its simplicity. With SW things are more complex and would require familiarity with knowledge representation languages such as RDF and extensions of it. That is why an important research area within the SW is the development of user-friendly tools and environments for dealing with meta-data, otherwise “create a technology for page

markup that's difficult to grasp and nobody will use it" [Dumbill 2001; Koivunen 2001]. Benjamin et al have identified six challenges that SW has to complete before becoming functional, as follows [Benjamins, Contreras et al. 2002]:

1. The availability of content. The WWW relays on the huge amount of data published. Things are the same with the SW: in order to talk about a SW revolution high quality and diverse semantic content (e.g. upgraded static HTML pages, multimedia, web services and so on) has to be available.
2. Ontology availability, development and evolution. Ontologies are the backbone of the SW architecture; so a big effort has to be invested in their creation, change management, mapping and evolution.
3. Scalability. As the WWW, the SW has to be able to sustain a huge growth.
4. Multilinguality. The access to the semantic content should be language independent (here language refers to the native language of the human user).
5. Visualization. It is expected that the amount and quality of information will be much higher on the SW than on the WWW. Hence a parallel research has to be carried out in the Human-Computer Interaction area, so that SW browsers will be able to handle the visualization of the new Web.
6. Stability of Semantic Web Languages. For the research in SW to continue it is identified as very important the process of standardization of the representation languages.

So, how the SW will look like? An answer is given in Figure 10, where "the bottom side of the figure shows the current web with its different types of resources: static pages, dynamic pages, web services and multimedia, all of them can be in different languages. Dedicated editors and wrappers aggregate the information in those sources into semantic indexes. A routing mechanism establishes and maintains the relation and communication between the various indexes. Software applications (agents) access SW content through the routing mechanism. Since semantics are represented using ontologies, an ontology lifecycle model forms a central component of the architecture" [Benjamins, Contreras et al. 2002].

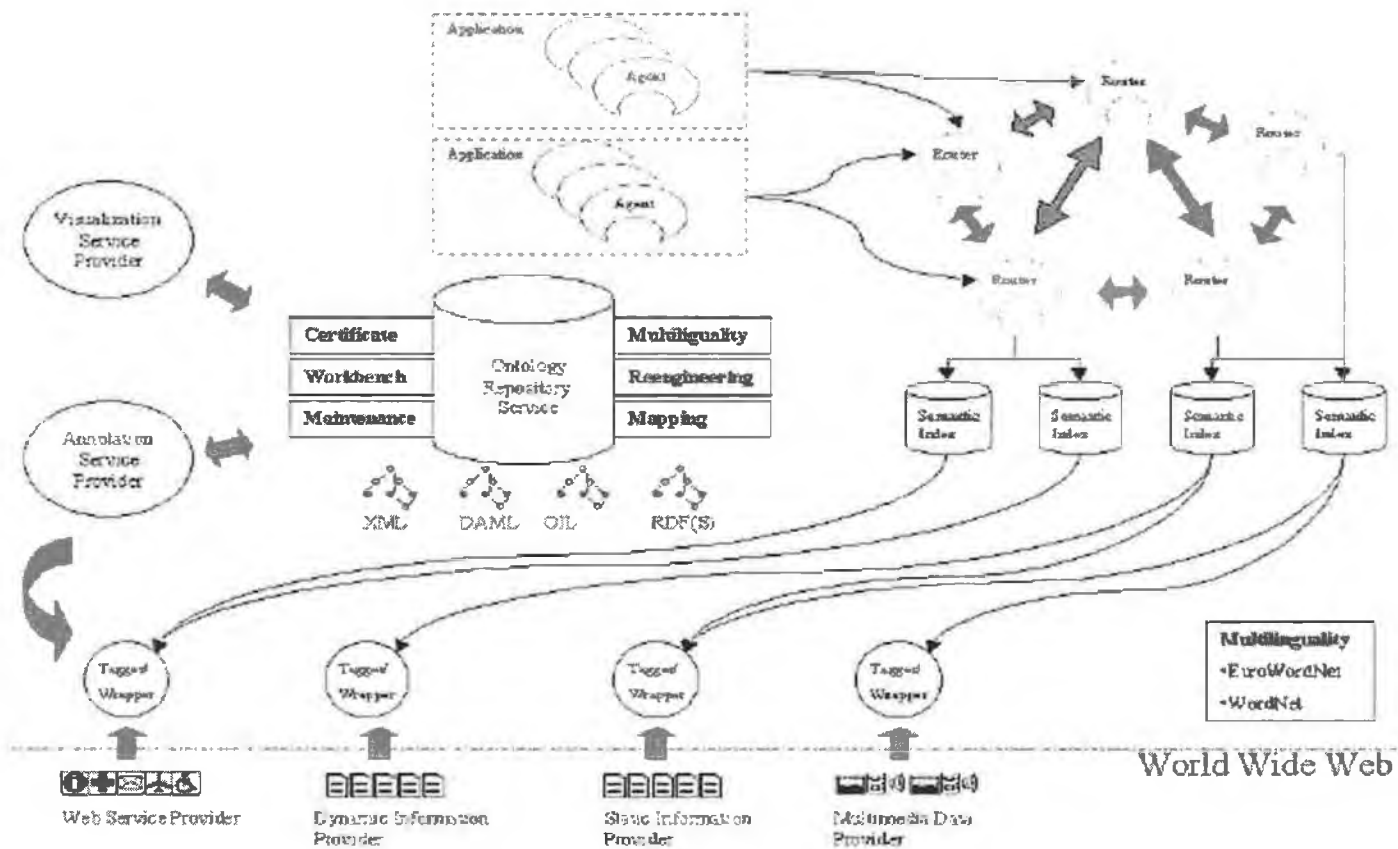


Figure 10. High-level architecture of the Semantic Web construction technology [Benjamins, Contreras et al. 2002]

Benjamins, V. R., J. Contreras, O. Corcho and A. Gomez-Perez (2002). Six Challenges for the Semantic Web. International Semantic Web Conference (ISWC2002), Sardinia, Italia.

Berners-Lee, T. (1998). Semantic Web Road Map.

<http://www.w3.org/DesignIssues/Semantic.html>, World Wide Web Consortium.

Berners-Lee, T., R. Fielding, U. C. Irvine and L. Masinter (1998). Request for Comments: 2396 - Uniform Resource Identifiers (URI). World Wide Web Consortium -

<http://www.ietf.org/rfc/rfc2396.txt>.

Berners-Lee, T., J. Hendler and O. Lassila (2001). "The semantic web." Scientific American **284**(5): 34-43.

Borges, J. L. (1984). Other Inquisitions 1937-1952, Univ of Texas Pr; ; Reprint edition (December 1984).

Bray, T., J. Paoli, C. M. Sperberg-McQueen and E. Maler (2000). Extensible Markup Language (XML) 1.0. **2003**.

Brickley, D. and R. V. Guha (2002). Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation 27 March 2000. **2003**.

Cherry, S. M. (2002). "Semantic Web: Weaving a Web of Ideas." IEEE Spectrum **39**(9): 65-69.

Connolly, D. (1998). The XML Revolution. <http://www.nature.com/nature/webmatters/>.

Consortium, U. "The Unicode Standard."

Decker, S., F. v. Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein and S. Melnik (2000). "The Semantic Web - on the respective Roles of XML and RDF." IEEE Internet Computing.

Ding, Y., D. Fensel and H.-G. Stork (2003). *The Semantic Web: from Concept to Percept.* 2003.

Dumbill, E. (2001). *Building the Semantic Web.*

<http://www.xml.com/pub/a/2001/03/07/buildingsw.html>, XML.com.

Ewalt, D. M. (2002). *The Next Web.* InformationWeek.

Farrugia, J. (2001). Logics for the Semantic Web. The First Semantic Web Working Symposium (SWWS '01), Stanford University, California, USA.

Fensel, D. (2000). Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce. Berlin, Springer.

Fensel, D. (2001). Ontologies: Dynamic Networks of Formally Represented Meaning. International Semantic Web Working Symposium (SWWS), July 30 - August 1, 2001, Stanford University, California, USA.

Gruber, T. R. (1993). "A Translation Approach to Portable Ontology Specification." Knowledge Acquisition 5(2): 199-220.

Guarino, N. (1997). "Understanding, Building and Using Ontologies: A Commentary to "Using Explicit Ontologies in KBS Development." International Journal of Human and Computer Studies 46: 293-310.

Heflin, J. D. (2001). *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment.* Faculty of the Graduate School, University of Maryland, College Park.

Hendler, J. (1999). *Is There an Intelligent Agent in Your Future?*

<http://www.nature.com/nature/webmatters/agents/agents.html>, Nature Webmatters.

Hendler, J., T. Berners-Lee and E. Miller (2002). "Integrating Applications on the Semantic Web." Journal of the Institute of Electrical Engineers of Japan 122(10): 676-680.

<http://www.w3.org/2001/09/01-borges/taxon.rdf>.

<http://www.w3.org/2001/sw/Activity>.

<http://www.w3c.org/XML/>.

Koivunen, M.-R. (2001). W3C Semantic Web Activity.

<http://www.w3.org/Talks/2001/1102-semweb-fin/Overview.html>, World Wide Web Consortium.

Lassila, O. and R. R. Swick (1999). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999. **2003**.

Palmer, S. B. (2001). The Semantic Web: An Introduction. **2003**.

Ramsdell, J. D. (2000). A Foundation for a Semantic Web.

Swartz, A. (2002). The Semantic Web (for Web Developers). **2003**.

Swartz, A. and J. Hendler (2001). The Semantic Web: A Network of Content for the Digital City. Proceedings Second Annual Digital Cities Workshop, Kyoto, Japan.

W3C "RDF Semantics."

W3C "Semantic Web Activity Statement."

Westoby, L. (2003). A Pedantic Web: The trouble with hypermedia evolution. 3rd Annual CM316 Conference on Multimedia Systems, Southampton University, UK.

www.semanticweb.org. **2002**.