OPAALS PROJECT

Contract n° IST-034824

WP6: Socio-Constructivism & Language

DELIVERABLE 6.14.

Using Ontologies to Access e-ContractDocuments. Semantic Search Engine v2.0.

Project funded by the European Community under the "Information Society Technology" Programme

Project Acronym: OPAALS

Deliverable N°: 6.14

Due date: May 31st 2010

Delivery Date: August, 4th 2010

Short Description D6.9 introduced some improvements, both from the legal and the computing point of view to be done in the Contract (Offers) definition and in the treatment of them to be "used" by the computer in order to give a right, quick and optimal answers. This document introduced and based the done changes and briefly describes the adopted solutions. So, from the legal perspective, in D6.12 has been faced the legal problematic of the DSC of ICT-Services, dealing with both a case of inconsistency and a case of insufficiency of General Terms and Conditions regulating ICT-Service Contracts.

Authors	ita: F.J. Lacueva, Erick Mendoza, Jorge I. Vea-Murguia, Javier Val. UniZar: Pedro Bueso, Tamara Martín, Leda Gómez. Diana Vollmer.
Partners contributed:	Universidad de Zaragoza (UniZar), Instituto Tecnológico de Aragón (ITA).
Made available to:	August, 4th 2010

Versioning		
Version	**Date**	**Name, organization**
0.1	21th September 2009	F.J. Lacueva & J.V. Ver-Murguia (ita).
0.2	15th September 2009	F.J. Lacueva (ita).
0.3	12th December 2009	E. Mendoza & F.J. Lacueva (ita).
0.4	12th April 2010	E. Mendoza & F.J. Lacueva (ita).
0.5	29th June 2010	E. Mendoza & F.J. Lacueva (ita). P. Bueso, T. Martín, L. Gómez (UniZar).
0.6	6th July 2010	F.J. Lacueva (ita). P. Bueso, T. Martín, L. Gómez (UniZar).
1.0	4th Aug 2010	F. Zeller (Unikassel). P. Bueso, T. Martín, L. Gómez (UniZar). F.J. Lacueva (ita).

Quality check

Internal Reviewers: Frauke Zeller (Unikassel)

Dependencies:

Achievements*	The following research activities and tasks underlying this Deliverable were **completed**: (1) Semantic Search Engine v2.0 (with a basic client implementation); (2) *Ontologies v2.0:* Creation of ontologies in order to store legal data for new eContract clauses. Trasnlation and semantic search functionalities are covered by ontologies; (3) New clauses definition; (4) *Service discovering:* Analysis on how to discover a service with specific contract details
Work Packages	WP6, WP7, WP8, WP12 (See point 2.3 for details)**.**
Partners	Universität Kassel
Domains	Services - Dynamic Service Composition
Targets	DSC and DE researchers, SMEs in the IST branch
Publications*	The reported work was published on the SourceForge website at: http://opaalstools.sourceforge.net
PhD Students*	F.J. Lacueva
Outstanding features*	(1) Improvement of XLS2LegalXML: usability; introduce the use the ontology to support the label translation and to restrict the vocabulary used by the creator of the document. (2) Development of a Semantical Service Finder in a Digital Ecosystem Platform with some of the next functionalities: translate the labels of the e-Contract Legal-XML document from its original language (German, Spanish, French, English, …) to a common language (for example English); Convert an e-Contract document from its Legal-XML representation to its ontological (OWL) representation; Review Ontology design to simplify it and include new requirements; Create a module to include the deductive service search tools in the service composition platform; A new version of the Finder tool allowing semantical search that is. (3) New release of the ontology for classifying, searching offers and contracts and translating contract. This new release includes original contract clauses specifying designed for DSC of ICT-Services. (4) An XLST document format convertor to include offers in the ontology as instances.
Disciplinary domains of authors*	ITA: F.J. Lacueva, E. Mendoza, J Vea-Murguia, J. Val (Software Engineering). UniZar: P. Bueso (Law), L. Gómez (Law), T. Martín (Law, Business Administration), D. Vollmer (Law).

The information marked with an asterisk (*) is provided in order to address Recommendation n. 4 from the Year 2 review report

Table of contents

Index of Pictures.

Index of Tables.

Executive Summary

In the frame of Task T6.12 (Improvement of Semantic Services Search Capabilities), ITA and UniZar have worked in the improvement of the Phase II prototype allowing semantic location of e-Contract Legal XML services.

On the one hand, ITA has worked in the technical research and developments. First, a state of the art on the schema Translator/Converter tools and algorithms has been created, and Ontologies have been used to translate/convert XML labels and units and the Semantic Reasoning technologies and tools. Then, some of the discovered ideas have been introduced in the search semantic engine prototype: LegalXML2OWL capabilities have been increased, a new LegalOWL translator/converter has been created and a new version of the sFinder module has been released. As collateral results, a new Contract Ontology and some "converter"/"translation" ontologies have been released.

On the other hand UniZar has given legal support to ITA and checked the results of the developments from a legal point of view. This has been reinforced with the analysis of the specific problematic of DSC from the perspective of Contract Law. Two different contractual gaps have been studied: a case of inconsistency of general terms and conditions, i. e., price and currency fixing, and a case of insufficiency of general terms and conditions, i. e., liability clause(s) in case of non fulfilment or defective fulfilment of the contract dynamic composed.

1. Introduction.

During Phase III of Opaals and within the task T6.12 of the WP6, **UniZar** and **ITA** worked on the improvement of the results we obtained in Phase II of the project.

In the former Phase, our work was centered in the business and legal support for defining contracts and offers from the business point of view and we developed the computing support for it. The results of Phase II can be summarized as (see point 2 for a more detailed background):

- The choice of eContract legal XML as the "format" language for contract definition.
- The identification of the common clauses for the definition of contract and offers for the services and product which are more commonly provided by ICT within the region of Aragon.
- A first ontology definition to support reasoning on defined contract offers.
- The development of prototype tools to create, transform and querying eContract offers.

These results were taken as the starting point for our work in Phase III. This document contains the basis and the summary of the work we developed. In opposition to Phase II, this work was centered in the computing support for increasing the semantic querying capabilities of the tools we developed on previous Phase. However, as it was remarked in the conclusion of the deliverable D6.9, some improvements, from the legal point of view, were introduced in the definitions of contracts/offers and in the legal support for the implementation of some computing processes such as the currencies conversion one.

1.1. Document Scope.

As we already said in the previous paragraph, one of the goals of this document is to base, both from the computing and the legal points of view, the developments we realized in order to allow semantic search of offers within the business level of a Digital Ecosystem.

In order to achieve this, from the computing point of view, we document the processes an Offer/Contract (which are defined in eContract LegalXML) should follow to be querying using ontologies:

- As we are on the border between the legal and the computing world we considered it to be very interesting creating a brief introduction to the ontologies and the DL world.
- When we started to work on the document transformation we thought that, as eContrac LegalXML and OWL are both "semi structured" documents, a good way to transform it would be using the XSLT capabilities. Hence, we will briefly introduce document transformation tools.
- As we need to include semantic reasoning, we took a look to the existing ones and we compared them, and based on this comparison we chose the one we use for being integrated in our prototype tools.
- Finally we briefly introduce unit conversion problems and perform a state of the art of ontologies to do it.

From the legal point of view we knew that in order to make DEs acceptable for the SMEs it has to be clarified which are the liabilities of each of the different agents that participate on a composed business process. This supposes to include new clauses within the contract which can be used by the user as determinant of their choices. Moreover in D6.9 we introduced a problem to be solved if DE wants to be accepted by SMEs all around the world: the reference (time) point of a DE should have to be provided in order to allow a correct (and legal) determination of the price of the product or the service to be provided.
Finally we show some learning lessons and things we considered that must to be done in order to improve our R&D undertakings. Moreover we briefly introduce our prototype tools and the new releases of the ontologies.

1.2. Structure of the Document.

We briefly introduce here the structure of D6.14. The first paragraphs are dedicated to define the scope of the document, its structure and a list of acronyms and definitions.
Point 2 of this deliverable extends the work in this 3rd phase: we based our work on the conclusions of D6.9 and tried to establish the relations with other Work Packages of Opaals. We must remark that the some of the technical explanations about ontologies have been moved to Appendix II.
As we are in the interface between the legal and the computing world, we dedicate point 3 to briefly introduce the ontologies and the description language to the non-technical readers of the document. We complement this introduction with a report (available for interested readers in Annex IV) of the existing tools both for defining and using ontologies.
Chapter 4 is dedicated to give legal support of our new ontology definition. It purposes a couple of solutions, regarded as regular issues of a concrete activity sphere in DEs, both to the price and fixing of currency exchange and the liability in DSC.
Point 5 is dedicated to introduce the technologies that allow us to convert, in a distributed way, eContract offers to its ontological representation by using the XSLT language and by choosing the tool we include in our prototype. The information provided in these paragraphs is complemented by the information available in Annex V.
The sixth point is dedicated to introduce ontologies as solvers for the Unit conversion problem. Although we did not need them for querying our contract, we think they are a solution for the currency change problem that chapter 4 solves from the legal point of view.
The previous point finishes the description of the technical background we developed and which is described in chapter 7. Some of the documentation of this point, to make this deliverable more clear, has been moved to Appendix III. Readers looking for a detailed definition of the ontologies, the processes and technical details of the solution are invited to consult it.
The last chapter of the deliverable, number 8, contains the conclusions of our work, the relation it has with works in other areas of the DSC and things we think that can be improved with future works.

Finally we want to signal that Appendix I contains the references to the literature we have consulted for our work. In order to increase the clarity we have classified them as either "computing" or "legal" references.

1.3. Acronyms and Definitions.

API	Application Programming Interface.
coNP	In computational complexity theory, co-NP is a complexity class. A problem $\mathcal{X}$ is a member of co-NP if and only if its complement $\overline{\mathcal{X}}$ is in complexity class NP.
CMR	Carriage of Goods by Road.
Datalog	Datalog is a query and rule language for deductive databases that syntactically is a subset of Prolog. Query evaluation with Datalog is sound and complete and can be done efficiently even for large databases. Query evaluation is usually done using bottom-up strategies.
DE	Digital Ecosystems.
DIG	Description Logic Interface.
DL	Description Logic. Description Logic Languages.
DSC	Dynamic Service Composition.
DTD	Document Type Definition is a set of markup declarations that define a document type for SGML-family markup languages (SGML, XML, HTML).
Dublin Core	The Dublin Core metadata element set is a standard in the fields of library and computer science. It is intended to be used for cross-domain information resource description. It defines conventions for describing things online in ways that make them easy to find. Dublin Core is widely used to describe digital materials such as video, sound, image, text and composite media like web pages. Implementations of Dublin Core typically make use of XML and are Resource Description Framework based. Dublin Core is defined by ISO in ISO Standard 15836 and NISO Standard Z39.85-2007.
Entailment	Entailment or Logical Implication is a logical relation that holds between a set T of propositions and a proposition B, when every model (or interpretation or valuation) of T is also a model of B. In symbols: 1. T $\Rightarrow$B 2. $T \models B$ 3. $T \therefore B$
F-Logic	Frame Logic (or F-logic) provides a logical foundation for frame-based and object-oriented languages for data and knowledge representation
HTTP	Hyper Text Transfer Protocol.

IST	Information Society Technologies.
KR	Knowledge Representation.
KRSS	Description Logic Knowledge Representation System Specification
Legal XML	LegalXML brings legal and technical experts together to create standards for the electronic exchange of legal data. LegalXML is a member section within OASIS[64] the not-for-profit, global consortium that drives the development, convergence and adoption of e-business standards. LegalXML produces standards for electronic court filing, court documents, legal citations, transcripts, criminal justice intelligence systems and others.
Logic Programming (LP)	Logic programming is the use of mathematical logic for computer programming. In this view of logic programming logic is used as a purely declarative representation language and a theorem-prover or model-generator is used as the problem-solver. The problem-solving task is split between the programmer, who is responsible only for ensuring the truth of programs expressed in logical form and the theorem-prover or model-generator, which is responsible for solving problems efficiently.
Metadata	Data about the data. Metadata articulates a context for objects of interest -- "resources" such as MP3 files, library books, or satellite images -- in the form of "resource descriptions". As a tradition, resource description dates back to the earliest archives and library catalogs. The modern "metadata" field that gave rise to Dublin Core and other recent standards emerged with the Web revolution of the mid-1990s.
NeO	Network of Excellence.
NP	In computational complexity theory, NP is one of the most fundamental complexity classes. The abbreviation NP refers to "nondeterministic polynomial time". Intuitively, NP is the set of all decision problems for which the 'yes'-answers have efficiently verifiable proofs of the fact that the answer is indeed 'yes'. More precisely, these proofs have to be verifiable in polynomial time by a deterministic Turing machine. In an equivalent formal definition, NP is the set of decision problems solvable in polynomial time by a non-deterministic Turing machine. The complexity class P is contained in NP, but NP contains many important problems, the hardest of which are called NP-complete problems, for which no polynomial-time algorithms are known. The most important open question in complexity theory, the P = NP problem, asks whether such algorithms actually exist for NP-complete and by corollary, all NP problems. It is widely believed that this is not the case.
OASIS	LegalXML is a member section within OASIS, the not-for-

	profit, global consortium that drives the development, convergence and adoption of e-business standards. Members themselves set the LegalXML agenda, using the open OASIS technical process expressly designed to promote industry consensus and unite disparate efforts. OASIS members participating in LegalXML include lawyers, developers, application vendors, government agencies and members of academia.
OWL	Web Ontology Language.
RDF	Resource Description Framework.
SAWSLD	Semantic Annotation Web Service Description Language.
SMEs	Small and Medium Enterprises
WP	Work Package
WSDL	Web Service Description Language.
XML	Extensible Markup Language.

2. Opaals' Background..

The paragraphs we include in this section try to put in contest D6.12 both on the Opaals NoE and in the Computing and Legal and Regulatory worlds. First, we aligned our prior work within the rest of Opaals'work: we based it on the D6.9 results we wrote in the Phase II of the project and we tried to relate it with other project work packages.
After the introduction of the problems we try to solve, we present the tools, technologies and legal and regulatory support which must be considered or should be used to implement a real solution.

2.1. D6.9 Our Previous Work within the Opaals Phase II.

This Deliverable, D6.12, took its starting point in the Conclusions of D6.9 that we, UniZar and ITA, realised in the Phase II of the project. D6.9 presents an applied approach to Digital Ecosystems, namely a case study of dynamic service composition in the Information Society Technologies (IST) *business domain*. Leaning on an empirical analysis of contract and negotiation processes in the IST sector in the Region of Aragón, this deliverable presents contract descriptions at semantic level in order to construct a test for dynamic service composition (hereafter, DSC). The next paragraph extends this short introduction to the work carried out in the deliverable D6.9, and relating it to the scope of the present deliverable.

D6.9 first introduces an overall description of the IST negotiation and contracting processes and briefly explains the features of the IST business domain in Aragón. This enabled us to present a short domain description as well as to highlight the main features of contracting processes for each selected area: *web and software development*, *hosting* services and *management and maintenance* services. After that, the document introduces the technological background that supports it: DSC, Legal-XML e-Contracts[59],[60], a first version of the Contract Ontology, together with the description of the capabilities implemented by the prototypal tools. Finally, it identifies some basic key features of digital business ecosystems as regulation fields and tries to map the major *legal and regulatory issues* connected to them.

In order to show that technology can support the ideas which are presented in D5.9 (both computing and legal ideas), we developed a prototype of the software infrastructure which should support it. Their main features are summarized in next list:

- To allow the DSC of products based on Web Development, Housing, Hosting and IT Maintenance products.
- To create some basic tools for supporting LegalXML e-Contracts document creation, publishing and search.

- To develop a common ontology supporting the four kinds of contracts, in order to increase the semantic capabilities of the publishing and searching tools (multilingual environments, deductive capabilities, etc).
- To create some test cases in order to probe supposed tools capabilities and, if possible, to detect new functionalities or improvements.

2.2. Conclusions of the D6.9.

The definition of the problem to solve, the implementation of the prototype and the test we performed showed us that there are problems which must be solved before creating new business opportunities from semantic business product or services description in an automatic or semiautomatic way. These problems are related either with the real (human) world, with the legal and regulatory environment (social problems) or the computing world (technical problems). We summarised them here as they become as the requirements or goals for our research.

The first problem we found is intrinsic to the European context in which the project is being developed: if we want to obtain a good SME Business Platform it should allow multilingual participation. That is, the "translation" to a common language (idiomatic) must be solved. This translation to a common language (English might be the best choice) may help not only to the human understanding of the offers but it may simplify some XML[1] file conversion format processes as tags would be written just in one language. We think that some of the capabilities introduced by ontologies will help us to solve it, by the inclusion of synonyms to the concepts expressed in different languages.

The previous problem is closely related with the problem of measurement unit transformation. Ontologies can help to solve it too. An example may help to understand what we mean: if we convert the unit used in an offer to a committed reference unit, then the comparison of offers can be done automatically and in a very easy way. Although at the moment the conversion of units seems an obvious problem, we did not realize this problem until we started to study the contract clauses of the SMEs of Aragón and to develop the templates of the contracts.

Solving the previous problems would allow to different humans sharing understable offers, that is expressed in the same language and with comparable units. Next step is to allow computers to understand offers and to work for humans in the selection of offers. That means to transform a set of Legal-XML e-contracts into an ontological representation of them allowing that reasoning tools can be used to automatic classification and selection of the associated offers. As a consequence, a tool to transform Legal-XML documents to ontology entities is needed. The tool should have to include these entities within the defined ontology (structure) in order to allow user queries.

From the computing point of view, the exposed problems (between others) have to be solved to create a real DE platform supporting DSC at the business level. Obviously, the solution to technical problems should be legally supported and viceversa: as far as technology moves, as many new scenarios appears and new legal problems appears: an iterative cycle of improvements starts. Next paragraphs

introduces legal problems detected in the elaboration of D6.9, which have to be solved, first legally and then from within the computing world.

As far as the legal aspects are concerned, we have seen that both digital ecosystems as such and DBE in particular seem to reproduce, albeit at a lesser scale, basic regulatory problems and issues which are typical for the Internet in general (virtualization, technological turbulence, extraterritoriality, jurisdictional difficulties, multiple normative sources).

How and to what extent these problems will affect DBEs will depend on the features and particular settings of each DBE, but, anyway, an adequate balance between over-regulation and under-regulation must be pursued, for otherwise regulatory attempts are likely to hamper DBE developments. DBE regulation, as a special modality of ICT regulation, requires an active involvement of the DBE actors and stakeholders, but it cannot be carried out regardless of the state legal constraints (legal framework) affecting most of the issues and topics raised by DBE activities.

Therefore, a complex or combined regulatory strategy was suggested: under the EU framework for e-business, co-regulation and codification (regulation-by-technology) seem to be the most adequate strategies. DBE regulatory needs (both community preferences or free decisions and decisions conditioned by external constraints) can be properly met by combining different instruments. In any case, a decision on the basic elements for the introduction of legal and regulatory issues into the DE paradigm must be taken as a relevant part of a general theory on DEs. This shall be done in a way that ensures DE workability in real practices (especially regarding SME business practices) without damaging the architecture and design principles of DEs. Because of its importance, this task is going to be pursued in Phase III in the framework of WP12. However, this deliverable is clearly related to the work done in WP12, as explained below.

2.3. Relations with other Opaals WPs.

This deliverable has relations with the work developed in:

- WP7: the existence of a common business level framework of offers and contracts will help to create and sustain DE communities.
- WP8: obviously the use of open standards and open source tools will support the acceptance of DE by SMEs.
- WP12, precisely with D12.12; the legal background constitutes an example of second-order contractual regulatory issues in an activity sphere consisting in provision of ICT services by means of DSC.

2.4. Problems to Solve.

The final goal of our work within the Phases II and III of Opaals is to allow users to create new Real World Business Offers and Contracts from the ones which are provided by other parties or by the users themselves. This, which is the common way to proceed in the human world, should be performed by a computer: the

computer should provide the contract list which best fits the user demands; in an environment where the offers can be added and removed automatically and which can be expressed in different languages.

From a very high and abstract level the previous problem has many similarities with the problems which are solved or are aimed to be solved by the DSC research community. In consequence we think that following the steps they are following we can obtain good results. The "roadmap" can be summarized as:

1. The best way to integrate services and systems is by using standard formats for the exchanged data. That is, if anyone wants to contribute and use services within a community, data should have to be provided in a standard format. For example, within the Web Services World XML, WSDL and others are used as standard ways to define the structure of files formats. Within Phase II UniZar searched for standard ways to express contracts, and eContract LegalXML was selected as the one best fitting our necessities. eContract DTD, which is expressed in XML, provides an standard way to create the infrastructure of a Real World Contract, that is an eContract document is a semi-structured document.
2. Once message structure and communication channels are established, both in human and computer world, if two peers want to speak they, either explicitly or implicitly, they have to establish the domain of the conversation, and the language they are using.

   Within the WS world the domain is established by creating an ontology. In Gruber's words an ontology is "*An **ontology** is an explicit specification of a conceptualization*"[64]. Ontologies are used to annotate the definition of the provided services and to describe the way in which exchanged data should have to be transformed to be passed to the services provided through a WSDL interface and to be transformed to the standard (ontological) format where they are obtained from. Once the WSDL files are annotated (which can integrate some translations capabilities) they can be queryied in an intelligent and standard way.

   In our case if we want to take advantage of the ontologies' capabilities we should have to transform the eContract LegalXML to a standard ontological representation. To ITA archieve this implies to:

   - Define the concrete domain of what we consider an eContract. That was made in Phase II by the UniZar and ITA teams: we chose the ICT SMEs as the domain to define contracts and take some product and service examples (hosting, housing, network maintenance and web development) in order to define the contract clauses we have to consider.
   - Once the domain was fixed, UniZar selected the main representative clauses and ITA worked on the definition of an ontological representation of the eContracts.
   - Finally the results of our work were:
     - A first version of the eContract Ontology. for
     - A set of tools which can be used to define eContract, to query for contracts. by users and to look for contract which best fix their

As we indicated in the conclusions of the Phase II work, although we advanced a bit in reducing the gap between the Real World and Web Serviced contract, there stille

is a long road to go both in the legal and in the computing dimension regarding the problem:

- The semantic annotation of the eContract should have to be provided in an automatic way in order to make this sustainable and useful both for humans and computers. It supposes to redefine the ontology and to transform the eContracts documents to a set of entities and relations between them that can be used to reason about them.
- From a more legal point of view, the interaction of peers (i. e., people, enterprises, …), which offer services and products from different places with different currencies and regulatory frameworks, opens new issues to be solved:
    - How does a price value have to be interpreted? How does the equivalence value has to be fixed?
    - The problem of the currency conversion shows a subjacent problem which has to be solved both in the legal and the computer world to connect user demands and providers' offers: both have to share the units in which the request and the answer are expressed or an automatic process of unit conversions has to be created.
    - What about liability issues if something or someone does not work in accordance with the clauses of the contract resulting from the DSC?
- To realize the first point of this list, from the computing point of view, we first should have to solve some subproblems:
    - In order to make queries easier all the contracts should have to be expressed in the same human/natural language, that is they have to be "translated" to a common language.
    - A reasoner module should have to be integrated within the infrastructure to allow "intelligent" queries.
    - A human interface should have to be provided to manage contracts (add, remove, update) and to allow queries.
- From a legal point of view, contractual gaps resulting from inconsistencies or contradictions and from insufficiencies of the general terms and conditions of the different offers taking part in a DSC must be filled in, in order to keep or increase legal certainty i. e. trust in the result of DSC.

Solving the former list of issues is the objective of UniZar and ITA within the Task 6.12 of Opaals' Phase III.

3. Technical Background.

3.1. What is an Ontology?

The Semantic Web vision, as articulated by Tim Berners-Lee, is of a Web in which resources are accessible not only to humans, but also to automated processes, automated "agents" roaming the web performing useful tasks such as improved search (in terms of precision) and resource discovery, information brokering and information filtering. The automation of tasks depends on elevating the status of the web from machine-readable to something we might call machine-understandable. The key idea is to have data on the web defined and linked in such a way that its meaning is explicitly interpretable by software processes rather than just being implicitly interpretable by humans [46].

Gruber [61] defined that an "***Ontology is an explicit specification of a conceptualization"***. An ontology defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, databases and applications that need to share domain information (a domain is just a specific subject area or area of knowledge, like medicine, tool manufacturing, real estate, automobile repair, financial management, etc.). Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among.. They encode knowledge in a domain and also knowledge that spans domains. In this way, they make that knowledge reusable.

In consequence, the word ontology has been used to describe artifacts with different degrees of structure. These range from simple taxonomies (such as the Yahoo hierarchy), to metadata schemes (such as the Dublin Core [60]), to logical theories.

The Semantic Web needs ontologies with a significant degree of structure. These need to specify descriptions for the following kinds of concept:

- Classes (general things) in the many domains of interest,
- The relationships that can exist among things,
- The properties (or attributes) those things may have.

Ontologies are usually expressed in a logic-based language, so that detailed, accurate, consistent, sound and meaningful distinctions can be made among the classes, properties and relations. Some ontology tools can perform automated reasoning using the ontologies and thus provide advanced services to intelligent applications such as: conceptual/semantic search and retrieval, software agents, decision support, speech and natural language understanding, knowledge management, intelligent databases and electronic commerce.

Ontologies figure prominently in the emerging Semantic Web as a way of representing the semantics of documents and enabling the semantics to be used by web applications and intelligent agents. Ontologies can prove very useful for a community as a way of structuring and defining the meaning of the metadata terms that are currently being collected and standardized. Using ontologies, tomorrow's applications can be "intelligent", in the sense that they can more accurately work at the human conceptual level.

Ontologies are critical for applications that want to search across or merge information from diverse communities. Although XML DTDs and XML Schemas

are sufficient for exchanging data between parties who have agreed to definitions beforehand, their lack of semantics prevent machines from reliably performing this task given new XML vocabularies. The same term may be used with (sometimes subtle) different meaning in different contexts and different terms may be used for items that have the same meaning. RDF and RDF Schema begin to approach this problem by allowing simple semantics to be associated with identifiers. With RDF Schema, one can define classes that may have multiple subclasses and super classes and can define properties, which may have sub properties, domains and ranges. In this sense, RDF Schema is a simple ontology language. However, in order to achieve interoperation between numerous, autonomously developed and managed schemas, richer semantics are needed. For example, RDF Schema cannot specify that the Person and Car classes are disjoint, or that a string quartet has exactly four musicians as members. OWL was created to complete these faults. We invite to interested readers to take a look to appendix A.II of this document where we include a short introduction to OWL and OWL2.
In order to understand one of the criteria this bases our choice of the tool to implement our reasoning module we now briefly introduce the Description Logic.

3.2. What is Description Logic?

Description Logics (**DLs**) is the name for a family of **knowledge representation** (**KR**) formalisms that represent the knowledge of an application domain (the "world") by first defining the relevant concepts of the domain (its terminology) and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description).

As the name Description Logics indicates that one of the characteristics of these languages is, unlike some of their predecessors, that they are equipped with formal, logic-based semantics. Another distinguished feature is the emphasis on reasoning as a central service: reasoning allows one to infer implicitly represented knowledge from the knowledge that is explicitly contained in the knowledge base.

Description Logics support inference patterns that occur in many applications of intelligent information processing systems and which are also used by humans to structure and understand the world: classification of concepts and individuals. Classification of concepts determines **subconcept/superconcept** relationships (called **subsumption** relationships in DL) between the concepts of a given terminology and thus allows one to structure the **terminology** in the form of a **subsumption hierarchy.** This hierarchy provides useful information on the connection between different concepts and it can be used to speed-up other inference services. Classification of **individuals** (or **objects**) determines whether a given individual is always an instance of a certain concept (i.e., whether this instance relationship is implied by the description of the individual and the definition of the concept). It thus provides useful information on the properties of an individual. Moreover, instance relationships may trigger the application of rules that insert additional facts into the knowledge base.

Description Logics are descended from so-called "**structured inheritance networks**", which were introduced to overcome the ambiguities of early semantic networks and frames and which were first realized in the system Kl-One[87]. The following three ideas, first put forward in Brachman's work on structured inheritance networks, have largely shaped the subsequent development of DLs:

- The basic syntactic building blocks are atomic concepts (unary predicates), atomic roles (binary predicates) and individuals (constants).
- The expressive power of the language is restricted in that it uses a rather small set of (epistemologically adequate) constructors for building complex concepts and roles.
- Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures. In particular, subsumption relationships between concepts and instance relationships between individuals and concepts play an important role: unlike IS-A links in Semantic Networks, which are explicitly introduced by the user, subsumption relationships and instance relationships are inferred from the definition of the concepts and the properties of the individuals.

3.2.1. Definition of a Basic Formalism.

A KR system based on Description Logics provides facilities to set up knowledge bases, to reason about their content and to manipulate them. The next picture sketches the architecture of such a system.

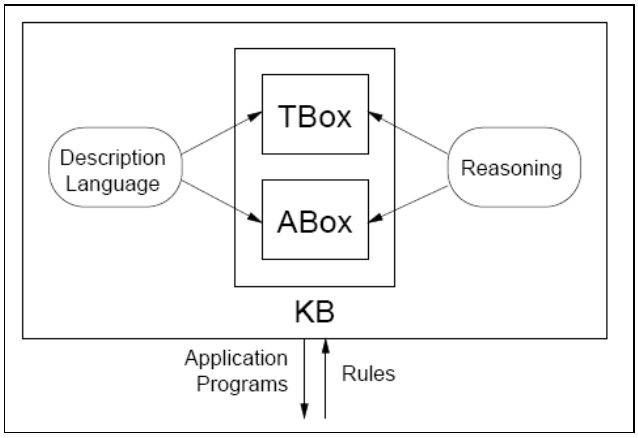


Fig. 1.- Architecture of a KR System Based on a Description Logics.

A knowledge base (KB) comprises two components, the T-Box and the A-Box. The **T-Box** introduces the **terminology**, i.e., the vocabulary of an application domain, while the **A-Box** contains **assertions about named individuals** in terms of this vocabulary.

The **vocabulary** consists of **concepts**, which denote **sets of individuals** and **roles**, which denote **binary relationships between individuals**. In addition to atomic

concepts and roles (concept and role names), all DL systems allow their users to **build complex descriptions of concepts and roles**. The T-Box can be used to assign names to complex descriptions. The language for building descriptions is a characteristic of each DL system and different systems are distinguished by their description languages. The description language has a model-theoretic semantics. Thus, **statements in the T-Box and in the A-Box can be identified with formulae in first-order logic** or, in some cases, a slight extension of it.

A DL system not only stores terminologies and assertions, but also offers services that reason about them. Typical **reasoning** tasks for a terminology are to determine whether a description is **satisfiable** (i.e., non-contradictory), or whether one description is more general than another one, that is, whether the first subsumes the second. Important problems for an A-Box are to find out whether its set of assertions is **consistent**, that is, whether it has a model and whether the assertions in the A-Box entail that a particular individual is an instance of a given concept description. Satisfiability checks of descriptions and consistency checks of sets of assertions are useful to determine whether a knowledge base is meaningful at all. With subsumption tests, one can organize the concepts of a terminology into a hierarchy according to their generality. A concept description can also be conceived as a query, describing a set of objects one is interested in. Thus, with instance tests, one can retrieve the individuals that satisfy the query.

As it was presented in [65] the T-Box and A-Box reasoning task can be summarized as showed in next lists. The T-Boxes are the following:

- **Satisfiability** checks whether a class *C* can have instances according to the current ontology.
- **Subsumption** checks whether a class *D* subsumes a class *C* according to the current ontology. Property subsumption is defined analogously.
- As a representative reasoning task we consider classification of the ontology, i.e. computing the complete subsumption hierarchy of the ontology.

A-Box reasoning tasks usually come into play at runtime of the ontology. Reasoning tasks typically considered for A-Boxes are the following:

- **Consistency** checks whether the A-Box is consistent with respect to the T-Box.
- **Instance Checking** checks whether an assertion is entailed by the A-Box.
- **Retrieval Problem** retrieves all individuals that instantiate a class *C*, dually we can find all named classes *C* that an individual *a* belongs to.
- **Property Fillers t**etrieves, given a property *R* and an individual *i*, all individuals *x* which are related with *i* via *R*. Similarly we can retrieve the set of all named properties *R* between two individuals *i* and *j*, ask whether the pair *(i, j)* is a filler of *P* or ask for all pairs *(i, j)* that are a filler of *P*.
- **Conjunctive Queries** are popular query formalism capable of expressing the class of selection/projection/join/renaming relational queries.

R. Shearer and col. in [81] well summarized the relation between Ontologies and Description Logig Knowledge bases. Given an OWL ontology O, it can be divided into three parts: the property axioms, the class axioms and the facts. These correspond to the T-Box T and A-Box A of a Description Logic knowledge base.

In any application, a KR system is embedded into a larger environment. Other components interact with the KR component by querying the knowledge base and by modifying it, that is, by adding and retracting concepts, roles and assertions. A restricted mechanism to add assertions are rules. Rules are an extension of the logical core formalism, which can still be interpreted logically. However, many systems, in addition to providing an application programming interface that consists of functions with a well-defined logical semantics, provide an escape hatch by which application programs can operate on the KB in arbitrary ways.

3.3. Reasoning with Ontologies.

The **standardization** of OWL[65] has led to the development and adaption of a wide range of tools and services, including reasoners such as FaCT++ [53], Racer-Pro [49], Pellet [44][56] and KAON2[37]. Reasoners are often used with editing tools, e.g., Protégé [42] and Swoop [164], in order to compute the class hierarchy and alert users to problems such as inconsistent classes.
One of the key benefits that a formal language standard provides to users of web-based technologies is *interoperability*. In the case of OWL, this should mean that users can load ontologies from the internet and use them in applications, possibly answering queries against them using one of the available OWL reasoners. One of the goals of **standardization is that the result of this process is independent of the chosen reasoner**.
Another important issue in order to achieve standardization and interoperability with different reasoning systems is the definition of standardized APIs to access to its functionalities.
In Annex IV we perform an study of the different tools for editing and reasoning with ontologies together with some interesting initiatives to increase and improve reasoning capabilities of the actual tools. The aim of this state of the art is to base our choice of the reasoning we integrated in the new tool for contract recovering which is explained in next paragraph.

3.3.1. Reasoner Selection.

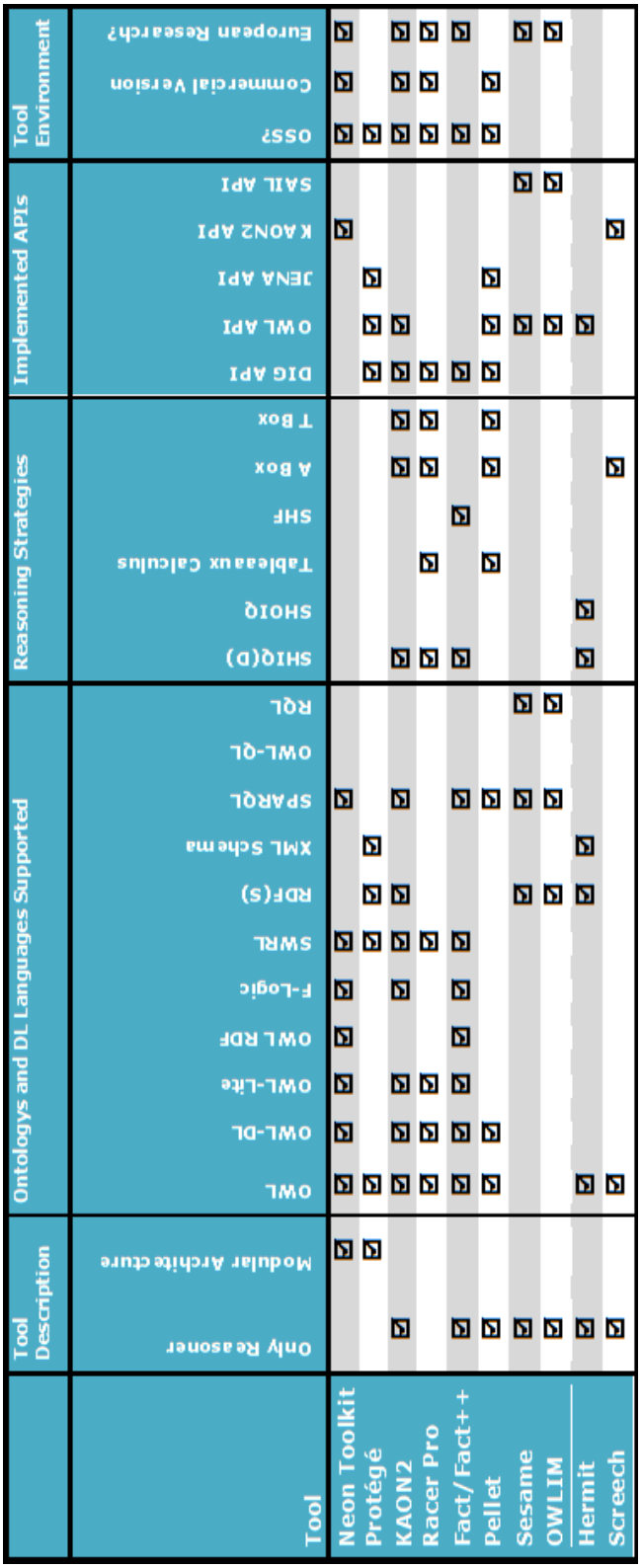
To select the reasoner we used we first chose the language in which the ontologies are going to be created. This choice came from the PHASE II of the OPAALS project and OWL was selected. It is a standard coming from a recommendation of the W3C and in consequence is being supported, developed and improved for many different entities from the enterprise to the academic world. In consequence one of the selection criteria for the Reasoner is that it has to support OWL to express the knowledge and for reasoning about it. In particular and to allow the use in future releases of the module under development supporting OWL 2 may be valuable, too.
The previous paragraph introduces another important feature the selected reasoner should have to provide: it has to implement some of the “standard” APIs described in previous paragraphs, DIG, Jena or OWL-API. Using a “standardized” reasoner

will allow to change it in the feature with a bit of effort. Obviously it is really important the selected API should have to be supported for a community of researchers and developers. Only DIG and OWL-API are still being developed at the moment and, if we compare both of them the second one seems to be defined to use its implementations as service provider. In consequence a second (main) dimension to consider in the selection of the reasoner is that it has to provide an OWL-API interface.

The next table summarized the main features of the different tools and reasoners we introduced in previous paragraphs. They are divided in five groups:

1. **Tools Description**: as we evaluated both ontology developing tools and only reasoning tools we classified them according to this, moreover we considered if they are a modular architecture or not.
2. **Ontologies and DL Languages Supported:** these columns evaluate the reasoners' capabilities to define ontologies and query languages.
3. **Reasoning Strategies:** defines the reasoning algorithms they implement.
4. **Implemented APIs:** evaluate which are the "standard" APIs they implement.
5. **Tool Environment:** determines if they are OSS, if they have a commercial release and if they are (partially) developed in Europe.

In order to choose "our" reasoner we considered mainly the two last groups of features. This reduces the reasoners to: FACT/++, Pellet, Racer Pro and Hermit. As we want to choose a reasoner that provides OWL-API interface, we discard FACT/++ and Racer Pro. Both Pellet and Hermit are available as Protégé plug-ins, but given that **Pellet** implements conjunctive queries, we decided to choose this one.



Tool	Tool Description: Only Reasoner	Tool Description: Modular Architecture	Ontologys and DL Languages Supported: OWL	OWL-DL	OWL-Lite	OWL RDF	F-Logic	SWRL	RDF(S)	XML Schema	SPARQL	OWL-QL	RQL	Reasoning Strategies: SHIQ(D)	SHOIQ	Tableaux Calculus	SHF	A Box	T Box	Implemented APIs: DIG API	OWL API	JENA API	KAON2 API	SAIL API	Tool Environment: OSS?	Commercial Version	European Research?
Neon Toolkit		☑	☑	☑	☑	☑	☑	☑	☑		☑									☑					☑		
Protégé		☑	☑	☑	☑			☑	☑	☑	☑									☑	☑	☑			☑		
KAON2	☑		☑	☑	☑		☑	☑	☑		☑			☑		☑		☑	☑	☑	☑				☑	☑	☑
Racer Pro			☑	☑	☑			☑	☑		☑			☑		☑		☑	☑	☑					☑	☑	☑
Fact/Fact++	☑		☑	☑	☑	☑	☑	☑			☑			☑			☑			☑					☑		
Pellet	☑		☑	☑	☑						☑					☑		☑	☑	☑	☑	☑			☑		
Sesame	☑		☑						☑		☑	☑	☑								☑			☑	☑		
OWLIM	☑		☑						☑		☑	☑	☑								☑			☑	☑		
Hermit	☑		☑						☑	☑				☑	☑						☑						
Screech	☑		☑											☑				☑					☑				

Table 1.- Reasoners Compartion.

4 Legal Background.

4.1. Preliminary remarks.

4.1.1. Legal concept and nature of Dynamic Service Composition.

Dynamic Service Composition (DSC) is possibly the most attractive and striking feature of Digital Ecosystems (DE), and especially of business ecosystems and business software systems in general[97]. We could find some notions about the concept of DSC in systems and computing engineering, as "the process of creating new services at run-time from a set of service components, that supports business agility, flexibility and availability"[98].

From a socio-economic viewpoint, one of the key challenges of DSC is to increase competition by articulating collaboration among different Small and Medium enterprises (SMEs). This can be described as a "collaborative competition process". Regarding this, the reason why SMEs introduce information technologies in their business processes is that SMEs are looking for obtaining simultaneously flexibility, adaptability and efficiency. Precisely, all these advantages will be successfully managed by implementing DEs. Providers will be able to offer their goods and services on Internet at a minimal cost through a Peer to Peer (P2P) platform which allows connecting SMEs and individuals for concluding simple or complex transactions in an open and decentralised environment. Definitely, agents who take part into DSC in a DE environment will be able to develop and extend their value chains through the network, interacting among them to achieve different aims which increase their competitive advantages[99], as creating and offering new products or services adapted to the particular user requests, promoting strategic agreements, reducing transaction costs, sharing knowledge, etc.

However, we could not find so easily a concept to define this phenomenon for legal purposes. There are not legal references for this kind of innovative process of composing complex services or applications, but certainly we could extract some ideas to describe its legal nature. Thus, DSC is conceived as an open source, distributed software platform based on Internet technologies, which allows to create electronic transactions that include complex contract operations in a realistic and legally-safe environment. Its purpose is composing new integrated services, by combining existing product or service offers from a plurality of different providers on the runtime upon a previous user request. In other words, DSC allows SMEs to work jointly for creating, integrating and providing services more efficiently and effectively.

In D6.9 we focused on structuring contracts for providing ICT-Services in order to allow their formalization in a language which is suitable for DSC based on Semantic Web. In D12.12 we want to go a step forward. Therefore, this section of the deliverable will focus the attention on new legal issues arising from the implementation of DSC. We will give an approach on two particular issues. First, the determination of price through the fixing of the exchange rate of the currencies used by the different provider's co-offerors. Second, the delimitation of liability when participating a plurality of providers in the execution of complex composite services.

4.1.2. Regulation of Digital Ecosystems and Dynamic Service Composition.

As said above, the analysis of the two questions which has been chosen do not solve all legal questions which may arise in DSC. In fact, there are many other issues which have to be considered in order to give enough legal certainty to a contract entered by means of DSC; i. e., data protection issues, use of e-signature, use of e-invoice, specific ADR schema – some of this issues belong to the community sphere, some of them to the activity sphere, as identified in D12.12.

However, here we focus on issues which are mainly related to Contract Law, because DSC is a clear case of necessity of second-order contract rules, as anticipated in M4.12: DSC leads to scenarios in which one user can simultaneously enter different contracts with different providers under potentially inconsistent contractual conditions, so that second-order solutions must be reached to address this issue [100]. This means, that freedom of contract can not be an absolute value, it must be balanced with some degree of standardization of the contractual terms and conditions; in other case, a situation similar to the "battle of the forms"[101] is going to happen, a situation which means a high risk of made legally inoperative a contract entered by means of a DSC, as far as a non-depurated combination of the general terms and conditions established separately by the each of the participating providers may give rise to contractual rights and/or obligations which are incoherent or contradictory, and this might affect the efficacy and the validity of the contract itself. Therefore, there is a need for shared general terms and conditions to avoid such risks. This is the case of the example related to price in different currencies and the need for fixing the currencies exchange rate(s): it is a matter of coordination of a basic element of the contract, i. e. price.

But inconsistency is not the only risk from the point of view of Contract Law when considering DSC. A contract entered by means of a DSC will give rise to specific contractual issues which are not attended by the general terms and conditions established separately by the each of the participating providers, as far as they are not considering such constellation. Hence, there is a great risk of insufficiency of regulation of a contract entered by means of a DSC if we just rely on a non-depurated combination of the general terms and conditions of the providers involve. And this is going to undermine the legal certainty, again, and so the trust of users in such a contract. Liability in case of non-fulfilment or wrong fulfilment is one, or probably, the main issue from this point of view.

Of course, all this shall be done respecting regulation on competition and fair trading, that means, the standardisation shall be limited to the minimum needed for the good functioning of DSC.

4.2. Price and Fixing of Currency Exchange Rate.

4.2.1. Legal Framework for Payment and Price.

The operations coordinated i.e. the contracts entered by means of DSC imply an economic value in their final concretion as consideration to the given services. Taking into account the idea that providers who offer DSC-based services can trade in different currencies, it is necessary to establish a system that determines the value of the currency in which the price must have to become effective and the amount of money that has to be paid by the user who requested services dynamically composed.

Legally, the issue of determination of price and, with it, of fixing the rate of change applicable to the currency in which the payment has to be made, rests on the previous determination of the movement and, to some extent, the place of completion of the contracts entered electronically by means of DSC.

We depart from the assumption – this should be established at the time to access as provider of services into the DSC – that the creation of the web page that gives the essential information on the contractual offer (description of the services, price and conditions of payment, etc.) and the incorporation of such essential information into the Internet in the context of a DSC implies for the co-offering provider the will to remain loyal to them and to the results of such a DSC, and that it does not mean a simple advertising communication of his services[102].

If we refer to the legal frame of the EU, the Directive 2000/31/EC on electronic commerce makes no explicit statement on this subject-matter. Due to this we will make use of the Comparative Law and certain transverse European and International regulations on the moment of completion when the contract is entered into by parties who do not meet personally, instead entering the contract from a distance.

So, Article 18 of the United Nations Convention on Contracts [103] for the International Sale of Goods establishes on the matter that:"

1. *A statement made by or other conduct of the offer indicating assent to an offer is an acceptance. Silence or inactivity does not in itself amount to acceptance.*
2. *An acceptance of an offeor becomes effective at the moment the indication of assent reaches the offeror. An acceptance is not effective if the indication of assent does not reach the offeror within the time he has fixed or, if no time is fixed, within a reasonable time, due account being taken of the circumstances of the transaction, including the rapidity of the means of communication employed by the offeror. An oral offer must be accepted immediately unless the circumstances indicate otherwise.*
3. *However, if, by virtue of the offer or as a result of practices which the parties have established between themselves or of usage, the offeree may indicate assent by performing an act, such as one relating to the dispatch of the goods or payment of the price, without notice to the offeror, the acceptance is effective at the moment the act is performed, provided that the act is performed within the period of time laid down in the preceding paragraph.*

Article 2.1.6. of UNIDROIT Principles Of International Commercial Contracts [104] determines that: "(Mode of acceptance)

1. *A statement made by or other conduct of the offeree indicating assent to an offer is an acceptance. Silence or inactivity does not in itself amount to acceptance.*
2. *An acceptance of an offer becomes effective when the indication of assent reaches the offeror.*
3. *However, if, by virtue of the offer or as a result of practices which the parties have established between themselves or of usage, the offeree may indicate assent by performing an act without notice to the offeror, the acceptance is effective when the act is performed."*

Finally, the article 2:205 and 2:206 in The Principles Of European Contract Law [108] comes near to an equivalent solution:

- Article 2.205, *"Time of Conclusion of the Contract (1) If an acceptance has been dispatched by the offeree the contract is concluded when the acceptance reaches the offeror. (2) In case of acceptance by conduct, the contract is concluded when notice of the conduct reaches the offeror. (3) If by virtue of the offer, of practices which the parties have established between themselves, or of a usage, the offeree may accept the offer by performing an act without notice to the offeror, the contract is concluded when the performance of the act begins."*
- Article 2.206, *"Time Limit for Acceptance (1) In order to be effective, acceptance of an offer must reach the offeror within the time fixed by it. (2) If no time has been fixed by the offeror acceptance must reach it within a reasonable time. (3) In the case of an acceptance by an act of performance under art. 2.205 (3), that act must be performed within the time for acceptance fixed by the offeror or, if no such time is fixed, within a reasonable time."*

Therefore, "contracts made by means of automatic devices" can be regarded to be concluded since the acceptance has been stated [106], not in the extreme sense of the theory of the emission of assent, according to which it would be enough that the offeree states his assent in any way to be concluded, but in accordance to the criteria of the reception of the assent by the offeror. Hence, it has to be regarded as the moment of perfection of the contract of the services offered and requested by means of a DSC when the user-offeree accepts the offer which results of such DSC, where the offer includes the services required, and the assent of the offeree is received by the providers-offerors. But it must be pinpointed that, in order to regard the contract as binding, there is no need for the offerors to know the content of the assent, as far as the user-offeree can not modify the terms of the offer obtained by means of such a DSC.

Consequently, "the order and the reception are considered to be realized when the parties to whom they are directed have the possibility of accessing them" [107]. Thus, if the contract is concluded directly by means of a DSC, it will be understood that the contractual order or assent is made when, once accepted by the user-offeree

of such a DSC, the offer which he receives (which includes the services required) can be acceded by their providers-co-offerors, it is said, at the moment at which the user-offeree clicks on "acceptance" by "ok" or "buy" in the interface of contracting of the DSC [108]. In conclusion, the acceptance or assent is considered to be known when it is received in the address (i.e., server) of the offerors and, is considered received, when the offerors can have access to its content.
From what has been discussed so far it can be said that an essential element of the web page of the DSC – where essential data and necessary information of the contract can be seen – is a technical device so that the acceptance can take place through an electronic route. According to this, by "clicking" in such a device and accepting the offer, at this moment there will take place automatically the conversion of the currencies of the prices of the services dynamically composed, being translated and reflected in an unique price, depending on the currency that the users and the offerors agree to use in the e-contract.
Therefore, the conversion of the price into a concrete currency will be made precise in this moment, paying attention to the information of values to be issued every day and published across diverse means belonging to diverse official entities; to such an effect, it is proposed to resort to the type of change fixed by the European Central Bank [130].

4.2.2. Clause on Price and Fixing of Currency Exchange Rate. Definition for the Demo.

1. The contract shall regarded as concluded when the providers-co-offerors receive the user-offeree's acceptance to the offer set out by means of the dynamic services composition by the providers-co-offerors; from that moment, the terms of the offer can not be modified.
2. The user-offeree can determine the currency of payment at the time prior to acceptance of the offer, choosing among the options presented. [i. e., OPTIONS: EUR – USD – GBP]
3. The price will be determined in the currency choose by the user-offeree. The conversion of the currencies corresponding the prices of the different services dynamically composed will be fixed at the time of conclusion of the contract, as determined in section 1.
4. The present value will be the result of applying the rate fixed by the European Central Bank, issued and published daily on [130]

4.3. Contractual Liability in Dynamic Service Composition.

4.3.1. Legal Framework for Contractual Liability in Dynamic Service Composition.

4.3.1.1. First approach to contractual transactions created by DSC.

As it has been introduced above (sec. 2.4.1), DSC allows to create complex obligational relations (*relación jurídico-obligatoria compleja*). This kind of relationship does not have a lineal structure – not only there are numerous agents

taking part, but there is a plurality of obligations as well. This structure refers to the so-called collective obligation (*obligación subjetivamente compleja*). The Doctrine defines collective obligations as those relationships for which each provider would execute one activity or task different from the others, in spite of being them all interconnected to obtain a common result which requires organization and coordination [109].

We should point out the importance of assuming the organization and coordination risks. Only if these risks are assumed by the group of providers who offer their services, we can consider the relationship being a collective obligation – in this case it would exist an obligational relation and the risk would belong to the entire group. Otherwise, if the user that made a request for a complex service is the one who assumes these risks, he will have composed his own integrated service, and therefore, it would exist a plurality of obligational relations which work independently, not depending on each other.

The aim of such obligation relations is to provide an integrated service which is the result of the composition of already existing goods and/or services. Hence, another relevant aspect that we should consider is customer satisfaction. Customers look for a complex service which better fits their needs and requirements. Therefore, customers pursue to receive the entire service requested, not being enough a partial fulfilment of the service. In other words, the customer only will be satisfied if each provider fulfils his own part, that is, if the service is comprehensively executed. The last approach contributes to the notion of unity or integration. Naturally, unity or integration must be understood in both economic and legal layers: unity of (a complex) service and unity of contract.

Here, the Doctrine refers to the economic aspect of integration, i. e., unity of service, in order to explain the utility or satisfaction obtained by customers when a service is comprehensively executed by providers. Thus, it will frustrate the economic business purpose in case of not fulfilment or partial fulfilment by providers. On the other hand, when analysing the legal aspect of integration, i. e., unity of contract, we must consider this relation as an indivisible opus, whose breach will generate the consequent liability [110]. According to LARENZ, "it is one legal relation which agglutinates several obligations on a superior entity".

The thesis mentioned above, which refers to the integrated contracts, has been confirmed by the courts (so, STS de March 15, 1982 on Art. 1137 of the Spanish Civil Code) as far as integrated contracts are regarded as a case of *joint and several liability*, that is, a designation of liability by which members of a group are either individually or mutually responsible to a party in whose favour a judgment has been awarded. The relevant question for providers to be jointly and severally liable is to determine the final purpose of the obligation relation and the different interests which are protected and satisfied by that relation [111].

4.3.1.2. Legal liability models

From now, we should analyse different types of contracts in order to find the most appropriated one or ones to subsume a contract entered by means of DSC. Our purpose is to describe a possible liability system which could be applicable to dynamic service providers.

As we mentioned above, DSC is not specifically regulated by Law, therefore the contracts which are concluded by this process will be so-called atypical contracts. Atypical contracts are a clear result of freedom of contract, whose limits are the binding legal provisions, the morality and the public order (so, Art. 1255 of the Spanish Civil Code).
The need of flexible arrangements makes it necessary to develop new types of contract in order to achieve economic growth, adaptation of business strategies and productivity; and in this context, agents celebrate contracts adapted to their particular circumstances by free will. Naturally, these new contractual forms do not have a typical legal categorisation, but they are contracts validly and effectively celebrated and therefore, they will be complied within the framework determined by the terms and conditions agreed by parties; where parties do not specifically agree, the contractual regulation will be determined by the general rules of Contract Law and the analogical application of the rules governing contracts of the same nature. As we can infer, it is a difficult undertaking to determine the applicable Law for atypical contracts, thus, our mission will be to clarify those extremes which conform the legal background in order to reinforce trust in the contractual result of a DSC.
It is inside the service provision field where the number of atypical contracts are multiplied, and it is not possible to lead them to the limited regulation which, in the case of Spain, the Civil Code establishes. Also, it is common to find complex service contracts as a consequence of the particular characters of the service offers. However, we should consider that in theses cases they are not absolute atypical contracts, because most of them would have been created as an union of several typical contract arrangements. Then, we could determine applicable regulation analysing jointly theses typical contracts [112].
The electronic transactions celebrated by means of a DSC could be subsumed in the category of contracts for service provision. However, its object is to provide an integrated service entirely. Because of this, it is not accurate to deal with such contracts as tying agreements [113]. In this sense, to determine the liability in this kind of complex transactions, we should have in consideration both general contractual rules and similar specific rules for service provision. Here, we could identify, at least, two contractual models which we could analyse in order to clear the question of the providers' liability. First, we will focus our attention on the contract of carriage of goods by road, and second on the package travel contract.

a) Liability model in the contract of carriage of goods by road

When referring to contracts of carriage of goods, we should notice that there are many types of them depending on the modals (by road, by sea, by air…) and the number of carriers executing this activity. Before analysing the different forms of liability that we could afford depending on carriage's nature, we should remark that participation of a plurality of carriers will not always be considered as a carriage performed by several successive carriers or a multimodal carriage. It could be possible that a plurality of different carriers executes the same carriage but performing their own stretch independently. The last case is the so-called fractionated carriage (*transporte fraccionado*): the issuing carrier (*cargador*) concludes so many carriage contracts as carriers perform on the route, therefore the

obligational relation only exists between each carrier and the issuing carrier but not among one carrier from another. In other words, it does not exist any liability for the different carriers connected to each other. Thus, each carrier will be responsible for his own stretch against the issuing carrier.
Despite of that, the carriage of goods performed by several successive carriers will consist of the addition of several partial carriages, conceiving the whole under the same contract [114]. Evidently, this kind of carriage is more frequent than another, because it only exists one agreement among the issuing carrier and the plurality of carriers. The Doctrine differences between two models of carriage which may be relevant in the case of contracting by means of a DSC: the carriage performed by successive carriers and the multimodal carriage. Therefore, the models consisting in carriage with reshipping and carriage with sub carriage are not going to be analysed [115].

i) Provisions relating to Carriage performed by successive carriers

As we anticipated before, all carriers who participate in the execution of the carriage are included in the same contract, therefore they are jointly and several liable (that means, they all, as singular contractual party, are liable without distinction for the whole damage caused) against the issuing carrier or the consignee for the integral service, despite of the fact that their respective activity is limited to a stretch of the route. This modality generates an obligational relation between the issuing carrier (or consignee) and the plurality of carriers; and also generates another internal relation between each carrier to another. The different carriers are *successively included* in the contract. We should notice that this last particular characteristic has no place in the DSC field, because in this case providers would be *simultaneously and jointly* included in the same contract for service provision [116].
To delimitate the provider's liability, we should analyse two perspectives. First, we will determine the ones who are called by Law to assume responsibility *ad extra*, that is, against the issuing carrier or the consignee; and in which cases they are called to assume it. Second, we will examine the internal relations among the different providers in order to determine the one who must assume definitely the liability *ad intra*.
Actually, to establish a joint and several provider liability represents an advantage for the customer, who will be able to claim for the whole damage at the same time against several of these providers. Therefore, to base joint and several liability systems on the notion of liability extension approaches such liability model to its authentic economic function [117].
We will analyse the Convention on the contract for the International Carriage of Goods by Road (CMR) [118]. Particularly, we will have consideration for the carriage performed by several successive carriers. Despite the literal text of the abolished Spanish Law on carriage of goods (Art. 373 of Commerce Code of 1885) which referred to successive carriers and that was different from the established Law on the Convention, it was interpreted by the Supreme Spanish Court as a joint and several liability case, therefore, in a similar way to the international convention. The recently passed Spanish Law of carriage of goods in force (*Ley 15/2009, de 11*

de noviembre, del contrato de transporte terrestre de mercancías) follows exactly the CMR liability model.

The International Conventions on carriage (so CMR, but 1929 Warsaw Convention for International Air Transport, too) establish a liability that arises from wrongful acts, that is, carriers shall be liable for the total or partial loss of the goods and for damage occurring between the time when they take over the goods and the time of delivery, as well as for any delay in delivery [119]. Thus, a carrier shall be responsible for damage caused by his fault and negligence and also he shall be responsible for the acts of omissions of his agents and servants and of any other person of whose services he makes use of for the performance of the carriage, when such agents, servants or other persons are acting within the scope of their employment, as if such acts or omissions were his own. Even, he shall be responsible when the loss or damage arises from special risks, if he is unable to prove his prudent action. The carrier shall, however, be relieved of liability if the loss, damage or delay was caused by the wrongful act or neglect of the claimant, by the instructions of the claimant given otherwise than as the result of a wrongful act or neglect on the part of the carrier, by inherent vice of the goods or through circumstances which the carrier could not avoid and the consequences of which he was unable to prevent.

The Convention of Carriage said that legal proceedings in respect of liability for loss, damage or delay may only be brought against several of these successive carriers. Particularly, against the first carrier, the last carrier or the carrier who was performing that portion of the carriage during which the event causing the loss, damage or delay occurred. This phenomenon could be a so-called "soft joint and several liability system" (*solidaridad imperfecta*), as the lawsuit shall not be brought against any carrier, but only against the first, the last or the one who was performing the carriage when the damage occurred.

The abolished Spanish Law on carriage of goods (Art. 373 of Commerce Code of 1885) was even more generous, because it established a so-called "*hard* joint and several liability system" (*solidaridad perfecta*). In these cases, the lawsuit can be brought against any of the successive carriers by decision of the claimant, and also allowing to bring an action against several of them at the same time.

Although both systems – soft and hard joint and several liability – produce different effects *ad extra*, that is, with respect to the claimant (issuing carrier or consignee); they work in an identical way *ad intra*, respecting to the other non defendants.

Thus, a carrier who has paid compensation shall be entitled to recover such compensation, together with interest and all costs and expenses incurred by reason of the claim, from the other carriers who have taken part in the carriage. Evidently, the carrier responsible for the loss or damage shall be solely liable for the compensation whether paid by himself or by another carrier, as we mentioned before that liability arises from wrongful acts. However, when the loss or damage has been caused by the action of two or more carriers, each of them shall pay an amount proportionate to their share of liability; should it be impossible to apportion the liability, each carrier shall be liable in proportion to the share of the payment for the carriage which is due to him. Finally, if it cannot be ascertained to which carrier liability is attributable for the loss or damage, the amount of the compensation shall

be apportioned between all the carriers (Art. 37 Convention; Art. 66 Spanish Law 15/2009; and Art. 373 of abolished Spanish Commerce Code) [120].
We understand that this kind of system, whether it is a soft or a hard one, is appropriated to be applied in the DSC field. In this sense, a customer would have reinforced his position through the providers' joint and several liability; and this will increase trust in the DSC. And also, it is an efficient mechanism for providers whose reclamation ad intra shall be brought against the one or ones who committed the wrongful act, absolving the person or persons who paid compensation.

ii) Provisions relating to multimodal carriage.

Given that the multimodal carriage is an important economic phenomenon, it has not only been regulated among the different European Laws, but it also exists an International Convention - United Nations Convention on International Multimodal Transport of Goods (*Geneva, 2 May 1980*) which gives the notion about this particular kind of carriage and includes a wide regulation. However, this Convention has not had the great success expected, because new European Laws of carriage are not including such regulation. This is the case of the New Spanish Law which instead of copying the Convention regulation, opts for including the jurisprudence model extract from the German Supreme Court (*Bundesgerichtshof*) [121].
Before any regulation (convention or current laws), the Doctrine was divided into different opinions. One group understood that multimodal carriage could be subsumed in a joint and several liability system [122]. However, another group pointed out that the absence of specific regulation does not allow establishing joint and several liability, and therefore liability issues should be purged attending to each particular stretch of the route. The last opinion was established in the Spanish Law 15/2009.
The mentioned Law regards as a liable person the provider who is performing the portion of the carriage during which the event causing the loss, damage or delay occurs. In order to impute this responsibility, the Law will be applied corresponding to the transport (ad. ex. carriage by road, by air, by sea...) used for the route. In case that such portion could not be determined, the issuing carrier or consignee (claimant) would be able to opt for the more beneficial Law among the concurrent ones. Definitely, each provider shall be responsible exclusively for his own acts, according to the service that he executes.
This system of liability could be appropriated, too, in order to be considered for the DSC field, because the responsibility is assigned among the providers according to their own actions and the Law applicable to these actions. A dynamic composite service is a complex service, which joints several and different services, and whose fulfilment must be global in order to provide satisfaction to the customers. As we mentioned before, this complex business follows an economic purpose which is unitary, therefore services cannot be provided separately, breaching its global purpose that characterises the DSC. However, we should point out that the different services which compose that unity are provided by different co-offerors, and in this sense, if DSC application would allow monitoring the process in order to identify the provider who does not fulfil his portion or does not execute his part, it could be possible to apply this kind of system. In any case, it should be envisaged a "final

clause" (*cláusula de cierre*) which refers to the provider's joint and several liability when it is not possible to determine the portion (part) of the service during which the damage occurs or to identify the provider who caused this damage.

b) Liability model in the contract for package travel.

Different from the last system described before - liability arisen from wrongful acts - the liability model in the contract for package travel is based on a principle of liability without fault. It is regulated by the Council Directive 90/314/EEC of 13 June 1990 on package travel, package holidays and package tours, which establishes an objective liability system for providers in order to protect consumers. In these cases, the persons who are called by the Law to assume liability are the organizer of the package travel, the retailer party, or both [124].

An objective liability system does not require fault of the organizer or the retailer party in case of damage for the customer, and if more than one person is liable for the same damage, it is a joint and several liability. In other words, when this objective system is applied, it is not necessary to prove the wrongful act or the negligence committed by the provider, being enough to accredit the damage suffered by consumer. However, the Law establishes as well several exemption liability clauses. Only if the provider (organizer or retailer party) could prove that the damage occurred by a special risk contemplated on an exemption liability clause, he could be relieved from liability. These exemptions are enumerated in a closed list (*numerus clausus*) by Law and they are objective, therefore they must be applied strictly.

In Spain, the abolished Law 21/1995 in contract for package travels established an objective liability system as well as the Directive, but it opts for differencing between two types of responsibility against the consumer – *ad extra*. If the damage occurs under the supervision of the organizer, only he shall be responsible; in the same way, if the damage takes place under the monitoring of the retailer party, he shall assume the compensation. Only in case damage occurs when both – organizer and retailer party – are involved, it will be applied a joint and several liability [125].

In spite of the Law's literal meaning, the Supreme Spanish Court interpreted the abolish Spanish precept of liability for package travel (art. 11) as a joint and several liability in any case, because it was argued that both – organizer and retailer party – assume the monitoring and direction of the whole travel. Precisely, the direction and supervision were incentives for the customer to contract the package travel, therefore it must be applied the more beneficial option for him in order to bring his reclamation for damage. Also, it should be mentioned that the aim pursued by the package travel is to receive all the different services included on the contract, not being enough a partial fulfilment of it. The consumer only finds satisfaction if the whole service is executed [126].

As the interpretation of TS was considered as more appropriate, the Spanish legislator modified the Law and the current one establishes a joint and several liability system in any case [127]. Once more, we can realize that when a plurality of providers participate in the same contract, it is better to bet for a joint and several liability system.

4.3.1.3. Conclusions

After analysing the different models of liability in Law, we should opt for a system which allows us to establish joint and several liability for providers. The aim is to reinforce customer position in order to promote trust and participation in the environment. In this context, the joint and several liability would satisfy an utility similar than a guarantee.

From an economic point of view, a customer, whether he is a company or not, will take part in the process of DSC if the expected benefit for that participation is higher than the benefit obtained by composing himself the complex service under his own organization and direction. Then, the costumer would be interested on contracting through this if costs associated to electronic transactions in virtual environment were lower than in real world, or when expected revenues were higher. *Ad ex.*, an agent, whether he is provider or requester, would be interested on contracting through DSC when he could reduce his transaction costs; or when he expected to increase his sales (incomes).

According to COASE, we could differentiate between two types of transactions costs [128]. The first type is composed of information, evaluation and measurement costs (*ad ex.* time required to contract). The second type refers to the contract fulfilment or execution costs. In this sense, it must be regarded that if DSC helps to reduce the first group of costs, then it is more attractive to realise online exchanges. The explanation of the last point is easy: providers are determined *ex ante* by the system, therefore, it does not exist any risk when identifying possible liable persons. Moreover, the object of contract is plenty established in the offer – product, quantity, and price – so customers do not incur high costs looking for information about the service that they wish to contract. After the request, the system generates automatically the different existing offers about a particular complex service.

Despite considering some benefits, we must also point out one disadvantage for agents that will be certainly common in this kind of virtual environment: the lack of confidence and trust. The question of confidence and trust is essential to increase the utilization of theses virtual environments: It is acceptable to assume that. customers prefer secured transactions rather than efficient transactions, therefore, they will be motivated to contract with well-known providers or providers who offer a guarantee. To solve this problem we should consider effective reputation mechanism and also specific online complaint systems. Other intermediary solutions could be to establish online dispute resolution systems (ODR/ADR), or even, to consider a public "punishment" for non-fulfilment service providers.

With respect to the second group of transaction costs, they will be high. First of all, OPAALS, as a legal entity, does not satisfy the conditions to be regarded as an Intermediary Information Society Service Provider[129], therefore, it will not assume any liability for the utilization of the environment made by the users. In the second place, the existence of a plurality of providers generates more complexity in order to determine liability and, also, generates uncertainty in order to claim that liability among the different providers. Thus, it is important to provide efficient mechanisms to impute liability to providers who act under DSC, and this is the reason why we propose reinforcing the customer position by considering a joint and several liability for providers.

4.3.2. Clauses on Contractual Liability in DSC of ICT-Services. Definition for Demo.

Regarding the considerations exposed above, two clauses shall be *ex novo* designed for the purpose of solving the issue of contractual liability in DSC for ICT-Services: a) a clause which establishes a join and several liability standard that we could qualify as a "hard" ones, as far as there join and several liability applies in any case, and b) a clause which establishes a join and several liability standard that we could qualify as a "soft" one, which only turns to join and several liability if it is not possible for the user to easily identify who is the provider who is not fulfilling or wrongly fulfilling his performance. Therefore, if it were not easy for the user to identify the provider or providers against to whom promote action (*ad ex.*, because of the fact that the DSC interface does not provides information about the degree of execution of the service and the stage where delays or other troubles may happen), it shall be recommended to implement the hard clause, in order to increase users' trust in DSC.

4.3.2.1. Clause which establishes a hard joint and several liability in a system of responsibility for wrongful acts.

If a dynamic composite service governed by a single contract is performed by multiple providers simultaneously, each of them shall be responsible for the performance of the whole operation, under the terms of the contract and according to the bona fide.

Legal proceedings in respect of liability for loss, damage or delay, when executing the dynamic composite service contract, may be brought against any of those providers who become a party to the contract. An action may be brought at the same time against several of these providers.

A provider who has paid compensation in compliance with the provisions of this clause, shall be entitled to recover such compensation, together with the interest and all costs and expenses incurred by reason of the claim, from other providers who have taken part in the contract, subject to the following provisions:

(a) The provider responsible for the loss or damage shall be solely liable for the compensation whether paid by himself or by another provider.

(b) When the loss or damage has been caused by the action of two or more providers, each of them shall pay an amount proportionate to his share of liability; should it be impossible to apportion the liability, each carrier shall be liable in proportion to the share of the payment for the contract which is due to him.

(c) If it cannot be ascertained to which provider or providers liability is attributable for the loss or damage, the amount of the compensation shall be apportioned between all providers.

4.3.2.2. Clause which establishes a soft joint and several liability in a system of responsibility for wrongful acts.

If a dynamic composite service governed by a single contract is performed by multiple providers simultaneously, each of them shall be responsible for the performance of the whole operation, under the terms of the contract and according to the bona fide.

Legal proceedings in respect of liability for loss, damage or delay may only be brought against the provider who was performing that part of the service during which the event causing the loss, damage or delay occurred, an action may be brought at the same time against several of these providers.

The provider who has paid compensation in compliance with the provisions of this clause, shall be entitled to recover such compensation, together with interest and all costs and expenses incurred by reason of the claim, from the other providers who have taken part in the contract, when he could prove that the loss, damage or delay was not caused or not only caused by a wrongful act or neglect of him. The provider responsible for the loss or damage, caused by a wrongful act or negligence, shall be solely liable for the compensation.

If it cannot be ascertained to which provider or providers liability is attributable for the loss or damage, the amount of the compensation shall be apportioned between all the providers.

5. Tools for Document and Data Conversion.

5.1 Schema "Translator/Convertor" Tools and Algorithms.

5.1.1. Introduction.

In Phase II of the Opaals NoE in WP6:

- We looked for a way to publish and offer services, semantically discovering them and create the bases to both negotiate some aspects of the offered services and consume the services: wsdl, owl, ontologies and the different logical reasoners seem to be the best way to provide technological support for this.
- We tried to establish the legal background which should support the overall services processes. At first, we looked for a standard document format, and e-Contract Legal XML was selected.

  Further we tried to communicate worlds, the "real" world and the computing world. To do this we followed two different paths. First, we surveyed some ICT-SMEs looking for the services they offered. In order to create some examples of the problems to be solved, we selected the four more common offered services. We developed tools to create the contract definition, to transform it to its eContract XML Legal-XML representation and to look, in a very basic way, for the contract that fits best the requests of final users. Later we used Protégé to create an ontology to represent the contract and we tried to use different reasoners to provide the contract that fits best the requirements of a final user: that is select it from the contract repository.

A very simple view of the problem can reduce it to convert between two different (XML based) representations of the same information (contracts) which have to be transformed in order to allow both worlds (the human and the machine) coexistence and interaction. Although this simplified view may help up to base the problem, our intention is to consider the overall context for the problem: although e-Contract XML determine the schema of a contract, it does not say anything about its element tags name or, obviously, about its content; the previous considerations suppose that contracts can be defined in any language (spoken in Europe) and its services definition use any metric units. As a consequence the problem complexity is increased, moreover if we want to apply acquired knowledge to a general problem we take to look for standard translation tools and methods in order to reduce conversion efforts.

Data (document) conversion problems are not new in the computer world since they are able to exchange information and more recently to interact. As a consequence there is a lot of literature at the moment. From our point of view, Wustner [8] is a good starting point since this paper established the basic strategies and tasks to convert documents in a quite recent technological environment.

The next paragraphs expose the problem we have to solve.

5.1.2. A General Conversion Problem Formulation.

In order to coordinate different actors' interactions, it is be necessary to exchange information. Generally the information exchange uses different standards and

formats. To achieve compatibility, the peers basically have two possibilities to choose from: the usage of a neutral, standardized format or a conversion between varying formats.
In our case, we have to transform documents from "standardized" XML formats, the e-Contract representation and its corresponding OWL ontological representation, in a way that they won't communicate with different systems but in order to solve the problem between the human understandable format and the machine standard.
There are three basic terms which clearly define the problem to solve:

- **Format.** It is seen in this context as a means for the structured description of information subjects.
- **Standardization**: A common reference framework and definitions which is accepted by all the parties with in a business.
- **Conversion**: the proces which transforms information from the format from one party to the standard one or viceversa.

The next schema shows a typical scenario in which the actors have agreed to use the standards 1 and 2 on a lower level, but to use different formats on a higher level. In a possible scenario, standard 1 could be SOAP for sending the documents, standard 2 could be XML and on top of these standards, partners would use XML based vocabularies such as xCBL, OAGIS or cXML for defining their business documents. Furthermore, they would use XSLT stylesheets or Java classes to convert their formats.

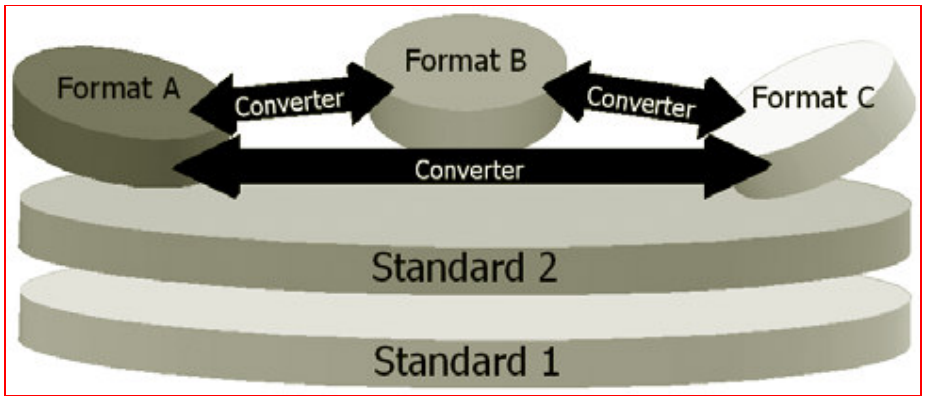


Fig. 2.- An Example of Conversion Possibilities

5.1.2.1. General Strategies to Solve Document Conversion Problems.

Conversions can be conducted using different strategies, which stem primarily from network topologies. It was assumed that bi-directional communication between the peers has to be supported. The following structures represent rather abstract, ideal solutions. In practice these structures are often combined or mixed up.

- **Ring Structure:** Conversion is conducted by passing the document in a single, pre-defined direction along the ring, until the document finally is converted into the requested target format.

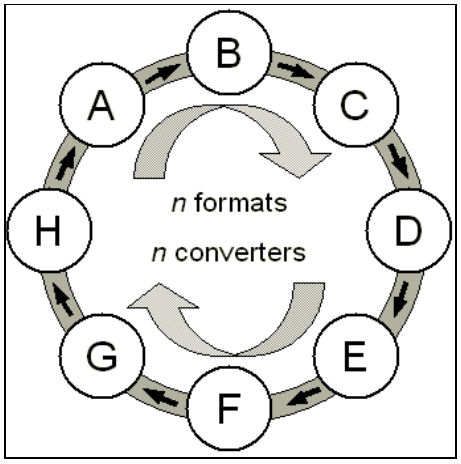


Fig. 3.- Document Conversion in a Ring Structure.

The problems this strategy involves are: if there are n nodes implied n converters should have to be developed, it could increase the quantity of lost information and the risk of errors propagation,

- **P2P structure**: in this case every format can be converted to any of the other formats.

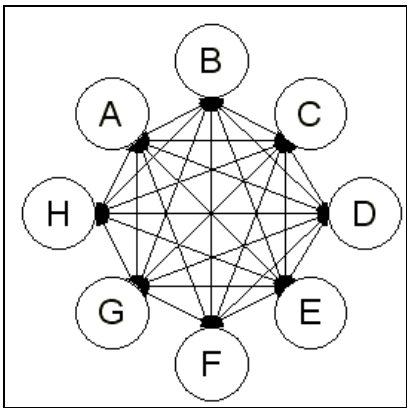


Fig. 4.- P2P Structure.

In this case if we considered n different formats, we should create n(n-1) converters. On the other side, the advantage is that the risk of information loss is reduced to the minimum while errors propagation will disappear as conversions are realized between peers.

- **Intermediary Formats:** this strategy is based on the definition of an intermediary format S which mediates in all document exchange processes within the network.

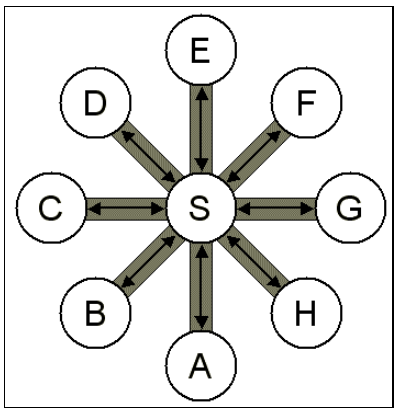


Fig. 5.- Intermediary Format Scenario.

In this scenario only 2 converters should have to be developed. However in order to decrease information loss, S has to be created to allow all the elements of the different formats. If it is created this way the risk of lost information will completely disappear and, as it happens with P2P structures, the error propagation problem will disappear.

5.1.2.2. A General Standard for Document (Information) Exchange: XML.

The Extensible Markup Language (XML)[1] is a standard to describe and exchange business documents. XML is a rather fundamental specification, which is just the basis for developing concrete standards, which then –on a higher layer– define means for storing and exchanging data in a content-dependent manner. In order to share information, people agree on the basic standard XML, but they are able to adopt the XML vocabulary that fits best to their specific needs. Examples of this are the specifications of the e-Legal XML Contract and the OWL documents.
Technologies such as Extensible Stylesheet Language Transformations (XSLT) [13], [14] can be used to convert XML documents [11], [12]. We base this hypothesis on the improved conversion possibilities when using XML as a means for electronic data interchange: usually a stylesheet is applied to an XML document by an XSLT processor such as Xalan of Apache[15]. The decreased efforts for generating the converter in comparison to traditional converting result from the following aspects:

- XML grammar is defined in DTDs or XML Schemas [11]. In terms of conversion, schemas provide additional functionality since they have more data types and can be parsed for further processing. This results in more implicit knowledge about the data that shall be converted.
- XML is human readable, so conversion information can often be extracted directly from the document.
- XML teams up perfectly with common programming languages such as C++ and Java.
- The existence of a large developer community around XML, with much of the software available being open source.

5.1.2.3. Using XML to Convert Documents Common Problems to Solve.

The authors examined the translation and conversion of XML-defined documents via XSLT at the syntactical and lexical level. They did not focus on the semantic level and so creating the style-sheet requires human assistance for building conversion rules. They categorized problems involved in the process of XML transformation and provide solutions and XSLT snippets where applicable to solve them. They focused both on the conversion of tree structures together with the conformity of the data problem which cannot be solved by simply transforming DTDs or XML-Schemas.

In order to clarify the exposition some terms are highlighted:

- **Structure of an XML document** is the extent to which information is drilled down into logical subunits**.**
- **Core Data** is the content of the character data encapsulated in tags and attributes
- **Core Data Format** is used to describe the format of the character data encapsulated in tags.
- **Grouping** means a set of tags, which form a group by belonging to the same context or sub context.
- **Overall Content Information** refers to the information provided by the combination of core data, markup and structure.

Permutations of these elements raise a number of problems, which are presented in the following classification:

- **Markup is different, overall content information is equal:** that means for the tag or attribute to be converted, that there is a corresponding tag or attribute in the target document, containing the same core data. In this case, transformation is simply done by inserting the core data into the matching target markup, which often results just in "renaming the tags".

a)

```
<Person>
  <FirstName>John</FirstName>
  <LastName>Doe</LastName>
</Person>
```

b)

```
<PersonalData>
  <Firstname>John</Firstname>
  <FamilyName>Doe</FamilyName>
</PersonalData>
```

Fig. 6.- Example of Documents with Different Markup.

- **Core data format differs, whereas core data itself is the same**: Converting information with XSLT in this case is comparably easy, since the core data is equal, only their format differs. The XSLT statement needed to transform the data has to be based on a known and fixed algorithm. This algorithm contains information on character order, separating characters, calculation rules, etc.

a)

```
<Date>11/15/2001</Date>
```

b)

```
<Date>15.11.2001</Date>
```

Fig. 7.- Example of Different Core Data Formats.

From this simple case more complex ones can be derived and have to be solved in order to correctly process the document transformation.

- The first case to consider is, at the authors suggested, the conversion between a finite number of values, such as country codes (Spain,

SPA, 34, E, España…). Lookup tables can be used to determine the appropriate identifier. A lookup table contains replacements and equivalences of certain values.

- A more complex problem to solve, which is not considered by the author but which we want to solve is the common unit conversion. This sometimes may be solved by a lookup table but usually would suppose to convert the measured value (for example from M/h to m/s). In consequence it would mean to first select the reference unit and then to define all considered units to the reference one.
- Finally the most complex problem to solve is the currency exchange rate one: it is exactly equal to the previous one but it needs to consider different conversion factors as the times go by. Moreover it has an important legal component which can be solved by the technical staff of the project.

- **Different structure depths, while the structure itself contains meta information:** The next picture illustrates two different formats, which contain strictly speaking only two names. The format a) explicitly states the information, that the first name is "John" and the last name is "Doe", while a reader needs semantic background knowledge in order to retrieve the same information out of format b), because the <Name>-Tag of format b) gives not enough meta information on its core data. The problem is, that the structure of b)'s core data has to be known. This knowledge can be defined in an algorithm for generating the appropriate structure.

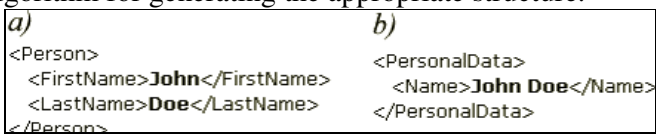


Fig. 8.- Example of Different structure depth.

- **Core data is the same, overall content information differs:** the two examples below show two document chunks, where the overall content information is different whereas the core data is the same. Format a) just stores two plain phone numbers, whereas format b) provides additional information about the network type of the phone number. This causes a problem if a conversion from a) to b) is needed, since there is simply no knowledge available about the network type in format a). In a few cases, lookup tables might be helpful. Otherwise, one could only try to guess. Hence, without additional information a correct and reliable conversion is rather unlikely in this case. Converting from b) to a) is no immediate difficulty, but nevertheless implies the loss of information about the network type of the phone numbers. This issue is a typical example, that there are situations, where information loss cannot be avoided.

Fig. 9.- An Example with Content Information Differences and Inevitable Information Loss in Conversion.

- **Different Grouping:** in a XML document the groups of tags form the semantic units through their nesting. In the real world there are a variety of possibilities to logically structure information.It is illustrated in next example. While in a) information about the VAT rate logically belongs to the item, context, the document in b) presumes that information on the VAT rate is part of the ‚Invoice Summary'. In such cases, exact and reliable conversion is strongly questionable, because it is by all means unsafe to infer information between varying contexts.

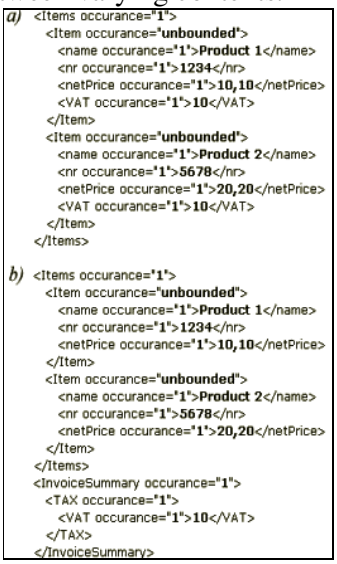


Fig. 10.- Different Information Grouping.

- **Redundant Information:** Some XML business vocabularies contain additional summaries or calculated results, which may be useful for the further processing of the document. This type of information is redundant since it can be obtained by processing, e.g. through calculating, summarizing, filtering or formatting other data given in the document. The following figure shows two formats. The tags <NetTotal>, <VATTotal> and <GrossTotal> of b) are redundant, since their core data is the result of processing

a)

```
<InvoiceDocument>
  <ItemList>
    <Item>
      <ItemNr>001</ItemNr>
      <Name>Product 1</Name>
      <NetPrice>100.00</NetPrice>
      <VATRate>10.00</VATRate>
    </Item>
    <Item>
      <ItemNr>002</ItemNr>
      <Name>Product 2</Name>
      <NetPrice>200.00</NetPrice>
      <VATRate>10.00</VATRate>
    </Item>
  </ItemList>
</InvoiceDocument>
```

b)

```
<Invoice>
  <Items>
    <Item>
      <ItemCode>001</ItemCode>
      <ItemName>Product 1</ItemName>
      <ItemNetPrice>100.00</ItemNetPrice>
      <VAT>10.00</VAT>
    </Item>
    <Item>
      <ItemCode>002</ItemCode>
      <ItemName>Product 2</ItemName>
      <ItemNetPrice>200.00</ItemNetPrice>
      <VAT>10.00</VAT>
    </Item>
  </Items>
  <NetTotal>300</NetTotal>
  <VATTotal>30</VATTotal>
  <GrossTotal>330</GrossTotal>
</Invoice>
```

Fig. 11.- Example of Redundant Information.

- **Missing Information:** It is obvious, that documents, which shall be converted into each other, have to contain the same overall information. Correct and precise conversion between formats with asymmetric overall information is by definition not possible. Anyhow, in practice this presumption is too strict. A common example is information about time in business documents. A number of XML business vocabularies require this kind of information, while many others do not. If such information is not necessary for the recipient of an XML document, the insertion of a "null" value might be a pragmatic alternative compared to no conversion and hence no data interchange at all.

5.1.2.4. How May XML Conversion Help to Develop DE?

In previous paragraphs we presented **the interoperability of systems** as the ability of ICT systems and of the business processes they support to **exchange data** and to enable the sharing of information and knowledge [21]. From our point of view, this does not only mean to provide the systems with the capacity to integrate (convert) data, it means to provide the systems with the capacity to correctly interpret different process definition languages too: in essence it can be considered a schema transformation process, which is usually represented in XML documents and thereby may be considered data transformation from an abstract point of view.

To follow this idea, a good starting point could be [10] where they made a comparison of the interchange formats for BPM. These standards use XML to define the "infrastructure" of its documents and are studied both to data interchange and process modeling formats. A more technical point of view of the same idea has been proposed in [22]. They suggested to define transforming processes using

BPEL as their EPC Markup Language (EPML) as a way to represent them to be used in EPC (Even-Process Driven Chains) modeling tools.
Both ideas, the necessity to transform data and processes representation –their xml representation-, were included in the definition of the architecture of WSMX [23]. They called the transformers "mediators" and used them to convert data and for processes representation.
Finally, there exists an important gap to save between the WSDL representation of a Web service and, if the deductive capabilities of OWL and the DL wants to be exploded, their ontological representation: SAWSDL (an extension of WSDL 2.0) has to be transformed to OWL if we want to use reasoning technology to intelligently search services. This process can be conceptually considered another schema transformation.

5.1.3 XML Tools for Document Transformation: Choosing the Tool.

In the previous paragraphs we introduced the general problem we want to solve: transform documents from one XML structured (or semi-structured) format to other. In Annex V we perform a comparison of different OSS tools which allows the transformation between XML Schema based documents. Interested reader can take a look to Annex V. Next paragraphs briefly summarize our selection to implement the DocumentTransformerService which is described in point 7.1 of this deliverable.

From Annex V, it is not clear to decide what XSLT engine is the better one. It is necessary to weigh the pros and cons of both tools.

Xalan integrates slightly better with Ant, while Saxon has much more features since the latest Saxon version implements not only XSLT 1.0 but also much of the working draft for XSLT 2.0, while Xalan basically only supports XSLT 1.0 and some extensions. But most of the extra functionality of Saxon is available only in the commercial versions and Xalan functionality is enough for OPAALS goals. According to some XSLT benchmark performance tests, Saxon is faster than Xalan [34], [35], but the performance is not a key requirement for the OPAALs' XSLT transformations.

Since both Xalan and Saxon implement the interface Java API for XML Processing (JAXP) 1.3, it is not necessary to take a decision and choose an XSLT tool because it is possible to write code using that API that works for both Xalan and Saxon. Xalan seems to be more standard compliant than Saxon, because Saxon's main API is not an standard API, so we will make the development using Xalan and we will be able to run the generated code using both Xalan and Saxon.

5.2. Case of Studies for Document Conversion.

These paragraphs are dedicated to introduce de examples we considered in order to converse a document from e-Contract Legal XML to its ontological representation.

5.2.1. Document Structure of the e-Contract Legal XML.

We briefly introduce the structure of the e-Contract document. First we briefly describe the general structure of the documents under study and then we detail for each of these parts the detailed parts under consideration for each kind of document:

- Housing.
- Hosting.
- Web Development.
- Network Maintenance.

These are the services that are more frequently offered to the customers.

5.2.2. The General Document Parts.

e-Contract Legal XML allows people to define very different kinds of contract. One of the conclusions of our work in our previous deliverable is that in order to avoid the complexity we have to reduce the complexity of the document under study. In consequence all the documents we are going to consider are going to be composed of:

- Contract Front.
- Body.

We introduce in the next paragraphs each of the elements we are going to consider for each of the part of the documents. First we will introduce the elements which are common to all parts of the documents and then, where it is necessary, we will detail the elements which are proper of each kind of document.

5.2.2.1. Elements of the Contract Front.

The Contract Front has obviously the purpose of identifying the contract together with the contract parties and the relation between them. The next table summarizes the elements we are considering:

TITLE	
AUTHOR	
Provider	
	Company Registered Name
	Company ID Number
	Company Registered Address
	Company Legal Representative
	PIN
Client	
	Company Registered Name
	Company ID Number
	Company Registered Address
	Company Legal Representative
	PIN
Acknowledgements	In witness whereof, the parties agree to the terms and conditions of this contract on behalf of his or her organization as of the date indicated below
Date	

Table 2.-Elements of the e-Contract Legal XML under Consideration in our Examples.

In order to allow finding offers it seems logical to consider that the only elements/fields to be considered are the Title and the ones under the Provider group: Company Registered Name, Company iD number, Company Registered Address, Company Legal Representative and PIN. The Date field has to be considered, too, in order to determine the validity of the offer.

The rest of the elements are not going to be present when a user (either a human or a computer) look for a service availability as they are, in the case of service contract, their own (enterprise) data and the date from which the contract is effective.

5.2.2.2 Elements of the Contract Body.

As it can be assumed, the contract body defines the contract features. This means that the e-Contract documents can be composed of several pieces. Our examples take into consideration five items:

- **Contract Object:** this is in our examples the place where users define the contract purpose. It only has an element "Type of Service" which can take one of the next values (or synonyms about the contract):
  - "Web hosting", Hosting from now on.
  - "Co-located service", we will refer to this as Housing from here.
  - "Networks Management & Maintenance" (NMM).
  - "Web Development" (WB).
- **Specifications:** these elements describe the contents and specific features included in each kind of contract. For our example they can be defined under two different elements: "SPECIFICATIONS" and "CONTRACT CONTENT". The later is used by the Web Development contracts whereas the former one is used by the other kind of contracts. For the purpose of our study both have the same semantic and they can be considered synonymous.

In consequence they should have to be translated to a common representation in order to be correctly processed.
The difference with the Contract Object is that each kind of contract has different sub-elements. They will be introduced in the next paragraphs.

- **Contract Duration:** these elements will reflect the duration of the contract. They can be classified in two different groups: "Duration" and "Delivery Time"**.** The first one is used by the Hosting, Housing and NMM kinds of contract with the meaning of time when the service is provided. On the other hand the "Delivery Time" is used by the WD contracts with the meaning of time to create the product.
- **Contract Price:** the price the claimant of the service should have to pay to receive the service or product. This will be interpreted in a different way by each of the contract types:
  - Hosting and Housing will be defined as the quantity per month to be paid in Euros in order to receive the service.
  - WD: the quantity to pay for the compromised development.
  - NMM: will interpret this in a similar way to the Hosting and Housing but, in some cases, with reference to the Annexes of the contracts.
- **General Terms:** These paragraphs of the contracts will increment the accuracy of the contract Specifications. They are completely dependent on the kind of contract. We refer the readers to Annex III in case they are interested in more details.

5.2.2.3. Other Elements to Be Considered in a Real Scenario.

Although we do not consider them, the parties' signatures, have to be considered in a real scenario. We do not consider them because the scope of our work is to offer and find contracts.
Another important part which is not included in our study but what is referenced in some elements of semantic description, are the Annexes.

5.3. The DTD of the e-Contract Legal XML.

The DTD of the e-Contract Legal XML can be consulted at:
http://docs.oasis-open.org/legalxml-econtracts/CS01/DTD/eContracts-Reference.dtd

5.4. The Ontological Representation of the e-Contracts.

One of the conclusions of the D6.9 was that, in order to create some tools to handle the e-Contract Ontology, it was necessary to simplify the ontology definition. In accordance with this we have created a new ontology from the one we created during Phase II. From this ontology (its OWL definition can be found in **Annex III**) we have derived the ontology which is used to "translate" the concepts representations from one language (Spanish, English,…) to the one we considered to define the ontology, that is English. The next paragraphs briefly introduce both ontologies.

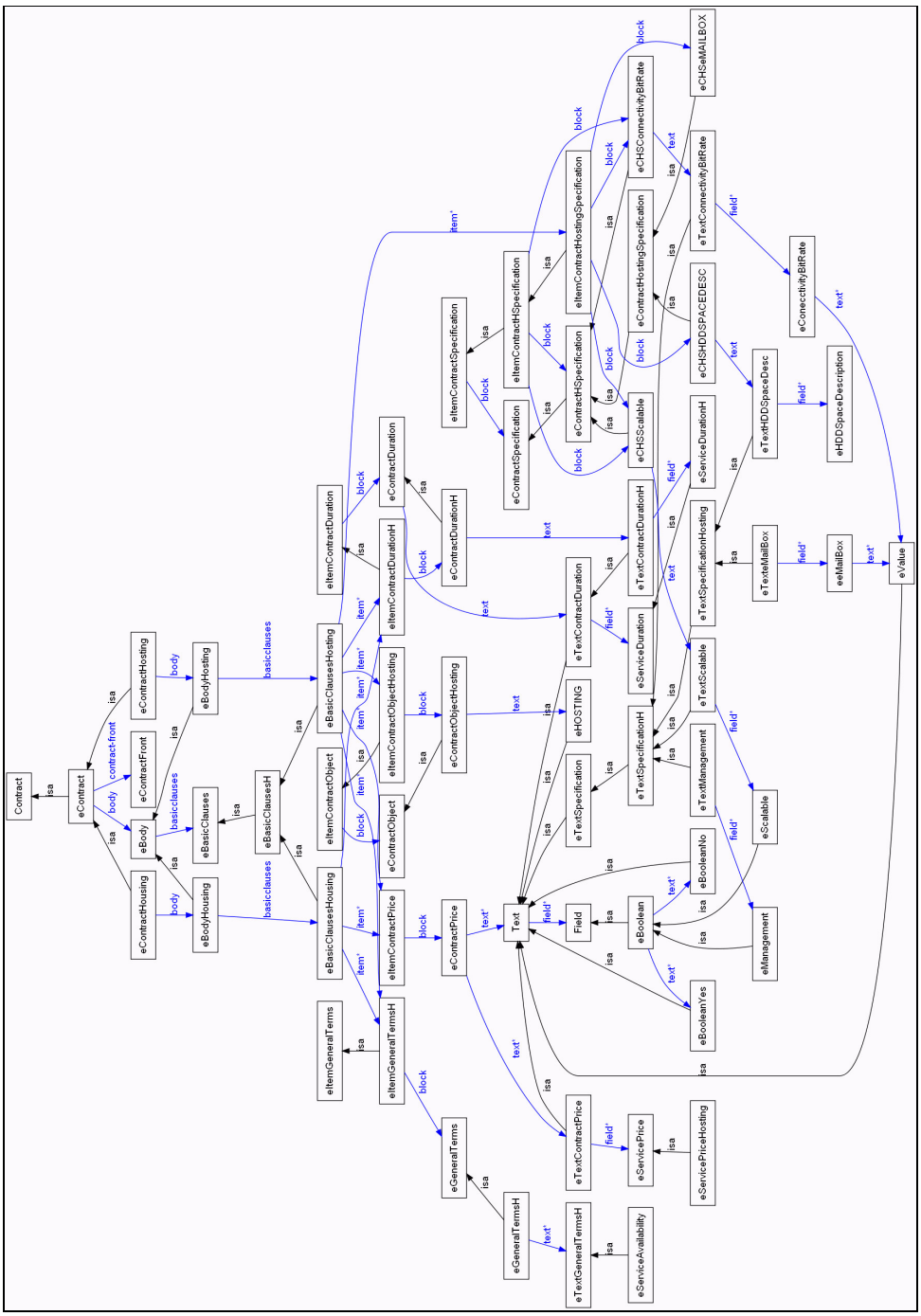


Fig. 12.- Hosting e-Contract Ontology.

5.4.1. The e-Contract Ontology 2.0.

One of the problems we found when we defined the previous version of the ontology was that e-Contract LegalXML is a generalist contract structure definition language, admitting recursive definitions of document elements and allowing them to have many relations between each other. For example, there is an element call

item which can be “anything” and which can be used within/related to many other elements.

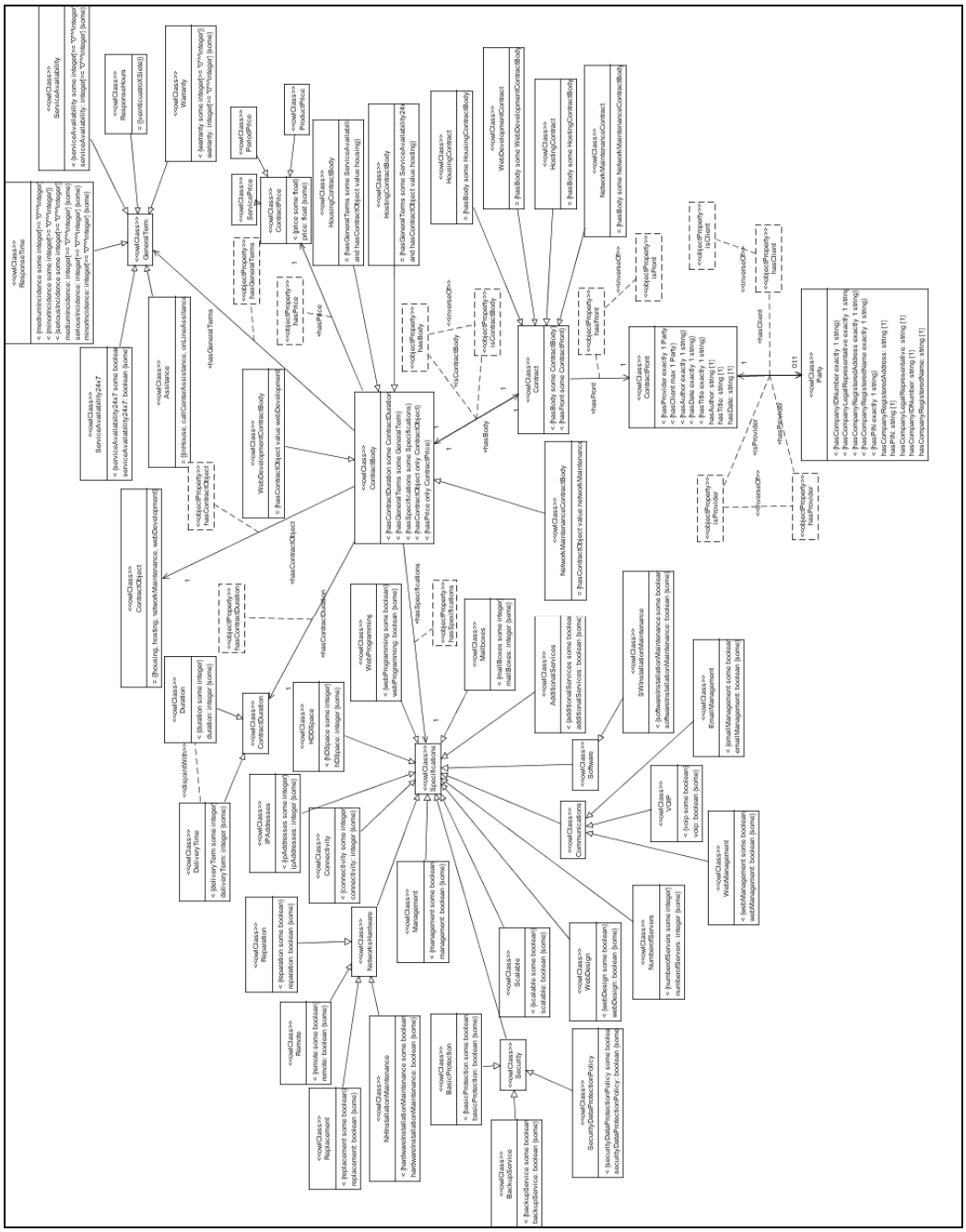


Fig. 13.- eContracts Ontology v2.0

As a consequence of this we obtained a big, complex and heavy ontology which is difficult to maintain, understand and increase. For example, the picture on the previous page shows the Ontology representation for a Hosting e-Contract Ontology. If we think that it was one of the contracts under consideration, even

considering that they had common elements, we can imagine the complexity of the complete ontology.
That is why we decided to reduce the ontology scope to the "sentences" we consider in our prototyping process. The result of the ontology, is shown as an UML representation in the picture 24.
A first look comparison of the v2.0 ontology with the picture 23 on previous page let know that the number of classes has considerably reduced. This helped in the prototype building and it will support the contract transformations from the Legal XML representation to OWL representation. Moreover the reasoning speed would benefit from it.
If the readers are interested in taking a more detailed view of the ontology and its owl definition they may take a look to the Annex III of this document.

5.4.2. The e-Contract Ontology 2.0 Version to Concepts Translation.

Another possibility for the use of ontologies is the "concepts translation". The underlying principle is very simple. An ontology's basic structure is a taxonomy of concepts that is a hierarchy, a tree of concepts. If synonyms are associated to concepts we can use this mapping to "translate" concepts between different languages. The next picture summarizes this idea:

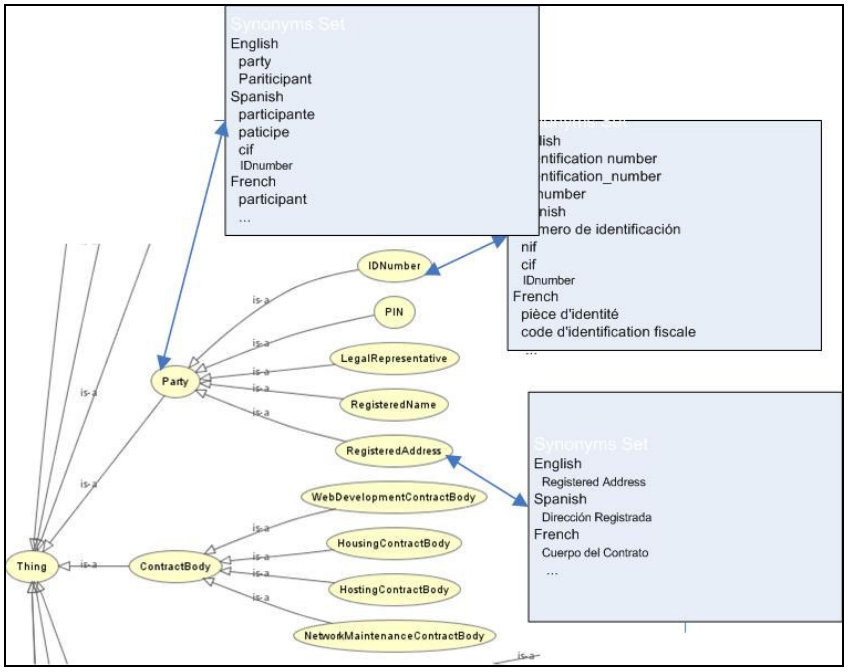


Fig. 14.- Summary of Concepts and Synonyms

There were many approaches in the past like the ones shown in [90],[91],[92],[93]. The process can be simplified since we are interested in the concepts (which are a commitment of the community's knowledge) but they can be expressed by different

terms in different languages. We only have to take a look for the terms and find the associated concept.
We used a "reviewed" ontology to do it which is visualized in the next picture.

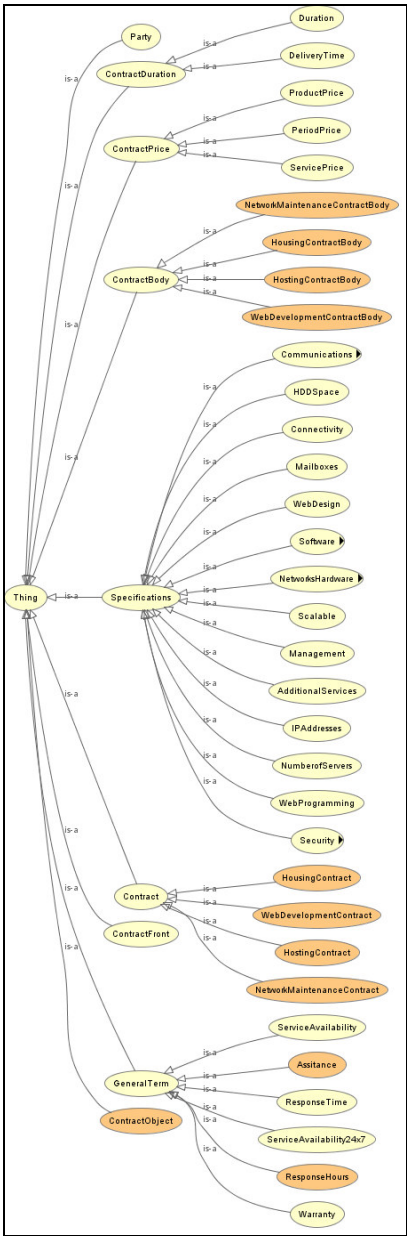


Fig. 15.- e-Contract Ontology 2.0 for Concepts Translation.

This ontology maps exactly the e-Contract ontology without considering "entities" relations. A detailed list of the synonyms can be found in Annex A.III.6.

6. Using Ontologies as Unit Converters.

6.1. Introduction.

Many people are convinced that a formal [97], digital definition of units and quantities is required. Automated support for parameter and unit selection, dimension and unit consistency checking, conversion of units, etc., would benefit from such a standard and eliminate many errors and misconceptions in science and engineering. Already for many decennia, more or less formalized versions of different unit standards exist on paper.

The reasons to arrive at a "standardized" ontological definition of the unit and measurements representations are:

- First scientists are moving from free text documents to digitized, structured information that can be processed by automated systems, that is taking advantage of the capabilities of the Semantic Web. Mathematical models are already a common way to express scientific knowledge (ontologies). In fact, emphasis in automating scientific practice has been on numerical processing and visualization. However, mathematics are not rich enough to support meaningful communication. The next challenge is to formalize the physical concepts and their relations underlying these models to enable proper interpretation.
- The second reason is that the interaction between scientists has become much more intensive, crossing disciplinary boundaries. Rather than exchanging finalized work through publications, worldwide electronic communication allows exchanging early model proposals or raw data. This will significantly influence the dynamics of scientific research. It also implies the need for standards for exchanging (the semantics of) models and data.

We use the next paragraphs to introduce the basic concepts in the background of a common ontology for unit conversion and interpretation and some ontological approaches.

6.2. Features Required of an Units/Measurements Ontology.

We first introduce what an Ontology Engineer should consider (general objectives) to construct a good ontology. Raskins [167] identified five design principles in constructing scientific ontologies, they are:

- **Scalability:** An ontology should be easily extendable to enable specialized domains to build upon more general ontologies already generated.
- **Application-independence:** The structure and contents of an ontology should be based upon the inherent knowledge of the discipline, rather than on how the domain knowledge is used.
- **Natural language-independence:** The structure should provide a representation of *concepts*, rather than of terms. The concepts remain the same regardless of the inclusion of slang, technical jargon, foreign languages, etc. Synonymous terms (e.g., marine, ocean, sea, oceanography, and ocean science) can be mapped separately to an ontology element.

- **Orthogonality:** Compound concepts should be decomposed into their component parts, to make it easy to recombine concepts in new ways.
- **Community involvement:** Community input should guide the development of any ontology.

Scalability, Application-Independence and Orthogonality are design principles which are common to other Software Engineering "Fields" in order to keep the developments alive, that is, to be sustainable in the future. The Community Involvement is obviously necessary: if you want your software being recognized and used you should listen to the world you want to be a part of. Finally and related to the previous objective, the Ontology should be Language-independent: one of the purposes of an Unit Ontology is to be used to convert between different measurements Systems which are language (regional dependent).
These design principles are complemented by Rijgesberg in the introduction of its paper [97]. It introduces other important items that should be considered in order to get a complete and standardized representation of a Unit and Measurement Conceptualization, that is an ontology:

- The Engineering Team should have a profound knowledge of the fundaments of physical unit systems as defined by the major standardization bodies.
- It also should have to hold knowledge and skills to apply modern ontology languages.
- They should have to acquire practical experience with the application of units in research: ontology of units serves a number of practical goals in science and engineering.

Another important thing that is introduced on this paper are some interesting "testing" questions what should be answered for evaluating the practical purpose of the ontology:

- For a given quantity, provide useful units.
- For a given unit, provide quantities that can be expressed using that unit.
- Perform dimension checking on a set of equations.
- Unit consistency checking.
- Convert from one unit to another.
- Provide meaningful quantities and units for a specific problem context.
- Give explanation and context of the quantities and units used.

6.3. Concepts to Be Included by The Ontology.

There exist different approximations to define an Unit Ontology which share some basic concepts. We introduce these concepts here and, if it exists, we show the different shade of meaning we found. In order to guide our discourse we follow a top-down approximation from the more general approximations [96] to the more concrete ones although they are subjective appreciations.
Raskins [167] worked on the definition of the SWEET ontology which is concerned with the definition of an Earth system science. It is shown in the next picture and it can provide the necessary context for the rest of our concept definitions.

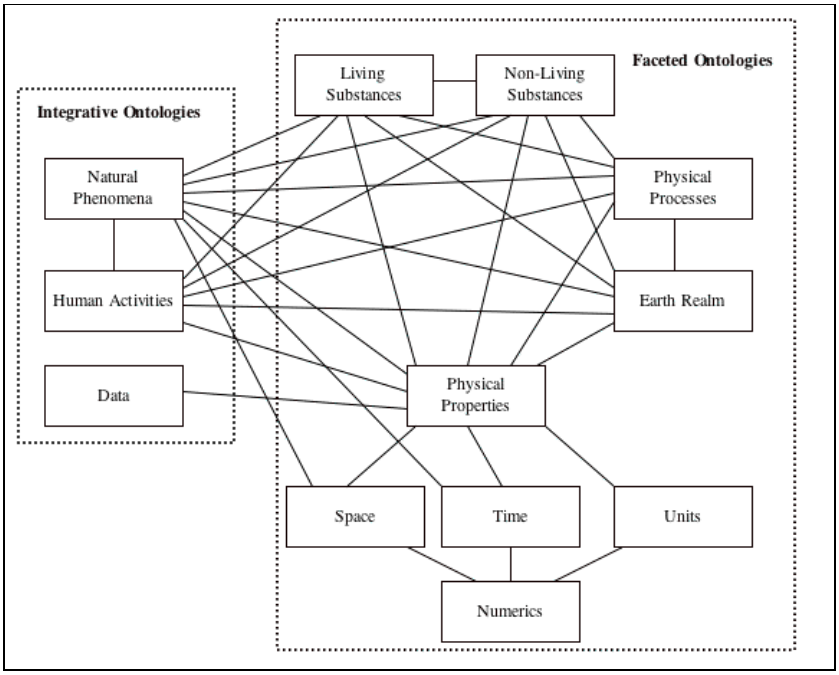


Fig. 16.- Major Properties used to Define Concepts Across Ontology Spaces.

We summarize here the concepts he defined which are related with our interests and some of their definitions will be detailed in the next paragraphs:

- An *EarthRealm* ontology, based upon the physical properties of the planet. This ontology can be considered a "state" of the planet that is extendable to past or future time periods, (as well as to other planets).
- Non-LivingSubstances: particles, electromagnetic radiation and chemical compounds. These substances constitute an ontology of physics and chemistry.
- LivingSubstances: plants and animals species.
- PhysicalProcesses: Physical processes include processes that affect living and non-living substances, such as diffusion, evaporation, etc.
- PhysicalProperties: "temperature", "pressure", "height", "composition", etc. This ontology could be applied to NonLivingSubstances, LivingSubstances, PhysicalProcesses, etc. These properties typically are measured physical quantities (or qualities) with units.
- Units: under this subontology the conversion factors between various units are gathered. Prefixed units such as km are defined as a special case of m with appropriate conversion factor.
- Time is essentially a numerical scale with terminology specific to the temporal domain. Temporal extents include: duration, season, century, 1996, etc.
- Space is essentially a multidimensional numerical scale with terminology specific to the spatial domain. We developed a space ontology in which the spatial extents and relations are special cases of numeric extents and relations, respectively. Spatial extents include: country, Antarctica, equator, inlet, etc. Spatial relations include: above, northOf, etc.

- Numerics: Numerical extents include: interval, point, 0, R2, etc. Numerical relations include: greaterThan, max. etc. We defined multidimensional concepts, as these are not native to the OWL and XML environments.
- PhysicalPhenomena: A phenomena ontology that is used to define transient events. A phenomenon crosses bounds of other ontology elements. Examples include: hurricane, earthquake, El Niño, volcano, terrorist attack and each has associated Time, Space, EarthRealms, NonLivingElements, LivingElements, etc. We also include specific instances of phenomena, spanning approximately 50 events over the past two decades.
- HumanActivities: This ontology is included for representing activities that humans engage in, such as commerce, fisheries, etc. This ontology is included because scientific processes and phenomena have human impacts and there is a need for representing such activities.
- Data. The data ontology provides support for dataset concepts, including representation, storage, modeling, format, resources, services and distribution.

As we said before this paper defines the global (universal) scenario where and Unit and Measurement Ontology should be "assembled" and it introduced some concepts which must be defined by the Ontology. In order to simplify the Ontology definition Rijgesberg[97] represents the "Real World" under the concept of the "Application Domain".

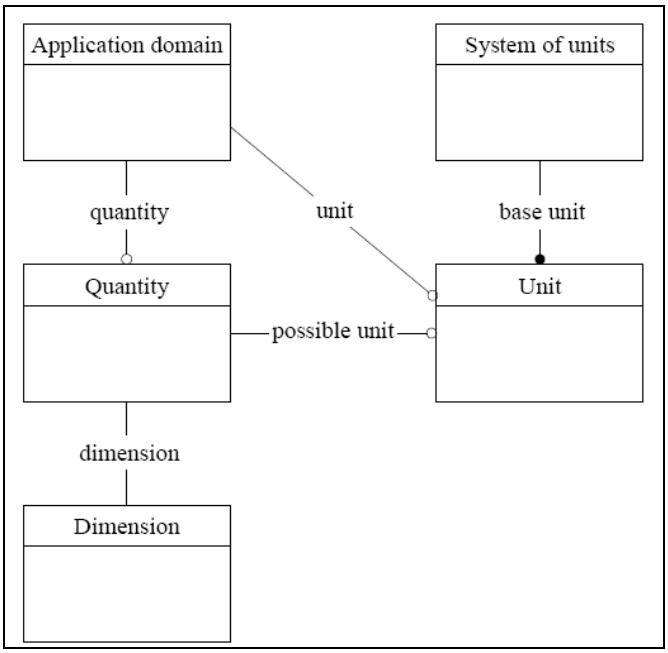


Fig. 17.- Class diagram of the central concepts quantity, unit, system of units, dimension and application domain in UnitDim.

That is the Application Domain is the link between the Ontology and the Universe model. In order to obtain an ontology of applicable units, they claimed for the formalization of quantities as a precondition. Several definitions of **Quantity** are used in different papers:

- [97] Defines quantities as the central elements in terms of which knowledge is formulated, in consequence, knowledge can be postulated in the form of statements involving quantities and their magnitudes.
- A **physical quantity** for [92] is a measure of some quantifiable aspect of the modeled world. Quantities "admit of degrees" in contrast to qualities, which are all-or-none. What makes quantities "quantifiable" is the ability to combine them with algebraic operations. The types of quantities determine the conditions under which operations are allowed and the types of the results. The central difference between a physical quantity and a purely numeric entity like a real number is that a quantity is characterized by a physical dimension: the physical dimension of a quantity distinguishes it from other types of quantities. This paper also extends this definition by defining Quantity from the point of view of the Artificial Intelligence convention of defining concepts as classes and so they define the class Physical Quantity and define species of quantities as subclasess of physical quantity.
- Another definition of Quantity which is relative to its classification in *Quantity Kinds* is given by [95]. A Quantity is the magnitude of a quantity kind. Quantities of the same kind can be compared to one another.

We can summarize the definition of Quantity [98] as a *unit* paired with a *magnitude*, a definition which is complemented by [92] by saying the set of values should have and order: When we express the magnitude of a quantity, a unit is needed as a common reference. Just one thing should be noted: Although all the definitions seem to be related to the term "physical quantity" for this generalized notion of quantitative measure, the definition allows for nonphysical quantities such as amounts of money or rates of inflation.

From the previous definitions it follows that three new concepts should be de defined: *Quantity Kind*, *Unit* and *Magnitude*. In order to close the generalization of the *Quantity* definition, we first study the approaches to the *Quantity Kind* concept. The relation between all these concepts is summarized on pictures 26 and 27 from [97] and [95] respectively.

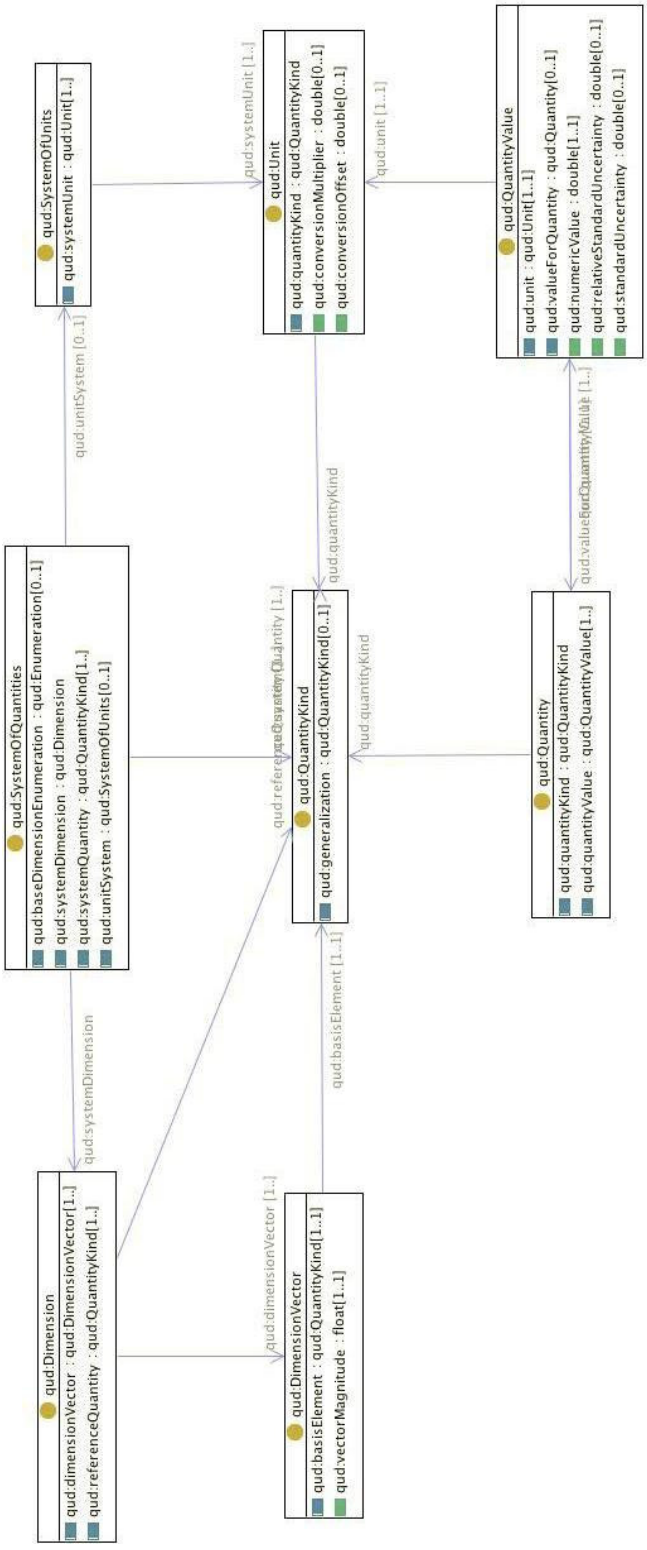


Fig. 18.- QUDT relation Ontology Class Structure.

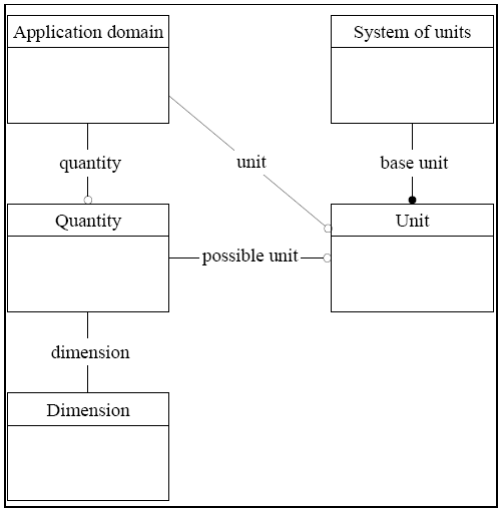


Fig. 19.- UnitDim: Class diagram of the central concepts quantity, unit, system of units, dimension and application domain.

Although definitions of Quantity Kind can be provided from different points of view such as *application domain* (mechanics, electromagnetic, economics…), we think a good definition is the one provided by [95]. In this paper a **Quantity Kind** is any observable property of an object, event or system that can be measured and quantified numerically. This definition links to the *Physical Properties* concept introduced by [96] and the *Application Domain* definition introduced by [97] as is shown in the next picture. An alternative way [98] for Typing Quantities is to use *Dimensions*.

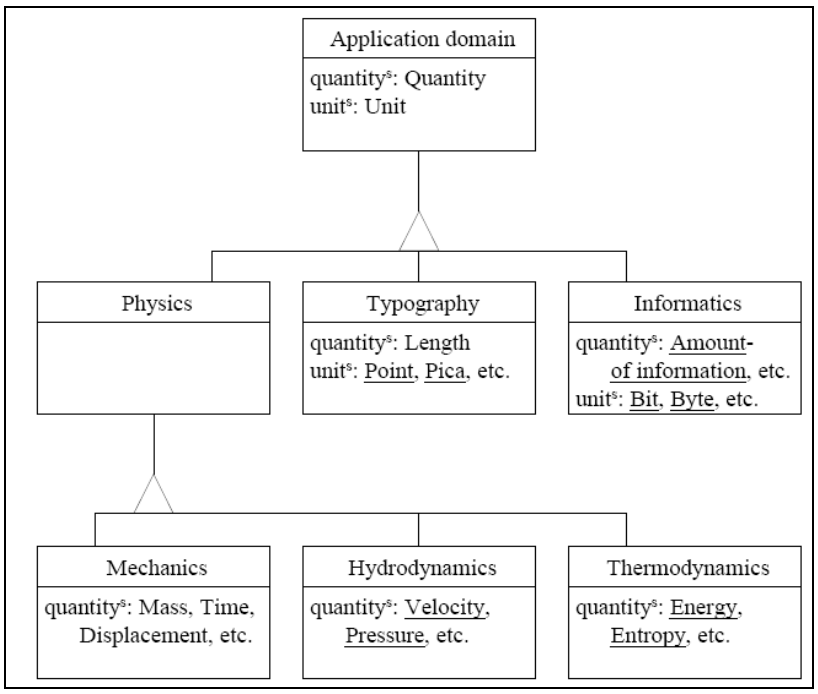


Fig. 20.- Class diagram of the concept application domain.

After putting the Units/Measurement Ontology within its context we proceed to define the rest of the concepts in a more detailed way. The next concept we try to define is **Unit.** For [95] a **Unit of Measurement** is a particular quantity of a given kind that has been chosen as a scale for measuring other quantities of the same kind. In order to clarify this definition we take the definition of measurement introduced by Finkelstein [91] which states that a measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. The previous definition is comparable with the definition given by:

- [98] Defines a Unit as a specially designated quantity, as such associated with a particular dimension, which is defined specifically for use in the description of other quantities.
- [92] Units of measurement are quantities themselves (positive, scalar, constant quantities). A unit of measurement is an absolute amount of something that can be used as a standard reference quantity. Like all the quantities, units have dimensions and units can be defined as any other scalar quantity: to provide unit conversion over all physical dimensions, every product and real-valued exponentiation of unit is also a unit of measure.
- And [97] for which Units define reference standards that express the (quantified) extension along a quantity's dimension.

This last approximation differs from the other two in the sense that it defines Unit from a more Mathematical, Engineering and Scientific point of view. While [95] and [98] do not refine the definition of Unit [97] extends it by introducing the concepts of Simple and Compound Unit as the next picture shows.

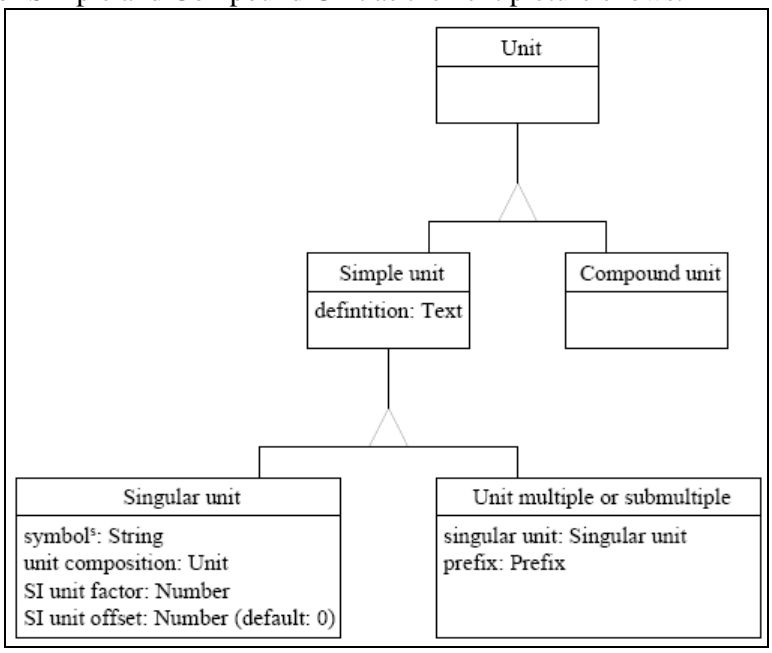


Fig. 21.- Class diagram of the concept unit.

A **Simple Unit** may have a definition. **Compound Units** are (mathematical) composites of simple units or other compound units. Examples of compound units are cubic meter, Pascal second and Newton per square meter. It also specified two concepts which are subclasses of the Simple Unit one:

- **Singular Units** are units that have a special name and (often) a symbol. Meter, gram and Kelvin are examples of singular units. The ontological representation of a Singular Unit includes attributes to identify its symbol; the conversion factor of a unit to SI units is given by the attributes SI unit factor and SI unit offset, combined with the dimension of the associated quantity. The SI unit offset is important in case absolute temperatures and times (years) is converted from the one scale to the other. Sometimes it is meaningful to give the composition of a unit in terms of other units, by means of the attribute unit compositions. An example is the *weber*, which is equivalent to volt second, a unit that is sometimes preferred when expressing magnetic flux. Some singular units remain implicit, i.e., they do not show. This typically occurs when the associated *quantities are dimensionless.*
- **Multiple and Submultiple Units** are introduced to avoid very small or very large numerical values: in practice prefixes are used to form decimal and binary multiples and submultiples of units. SI prefixes, representing powers of ten, are widely known. Although being called SI prefixes, these prefixes are also used outside the SI system of units (e.g. the decibel employs the prefix deci, but the bel is not an SI unit).
  In addition to decimal prefixes, binary prefixes were introduced by the International Electrotechnical Commission (IEC), to offer a format preventing erroneous usage of the SI prefixes in computer science. For example the prefix kilo is commonly used to indicate 1024, instead of 1000 since $2^{10} = 1024$ binary prefix kibi is introduced, representing exactly this factor 1024. The next picture summarizes both kinds of Prefixes.

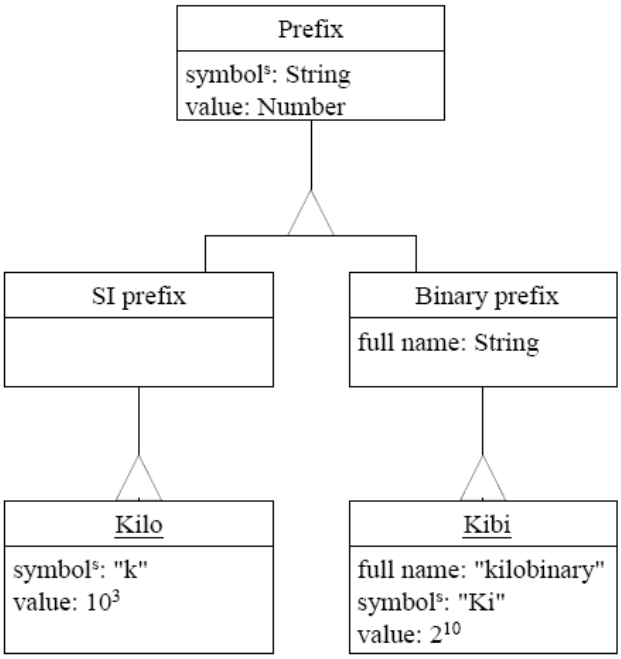


Fig. 22.- Class diagram of the concept prefix.

Although the concept of *Dimensions* will be introduced in next paragraphs we use them to give the relation between the *Single and Compound Unit* concepts used by [97] to the concepts of *Based Unit and Derived Units* used by [95] and of *Based and Derived Dimensions* used by [98].

To prove it let us take the definition of **Compound Units** which is given in [97]: Compound units are formed by multiplying or dividing one or more simple units (singular or multiple). Basic unit operations are division, exponentiation and multiplication. More complex operations are division-exponentiation (a division with an exponentiation in its denominator) and division-multiplication-exponentiation (a division with a multiplication in its denominator, where one of the terms is an exponentiation). Explicit definition of this set of complex operations is needed to prevent for meaningless combinations of units in more complex combinations from appearing as separate compound units in the ontology. An example is second squared, which occurs in many more complex unit expressions, but makes no sense in a physical way. The next picture exemplifies our previous definition:

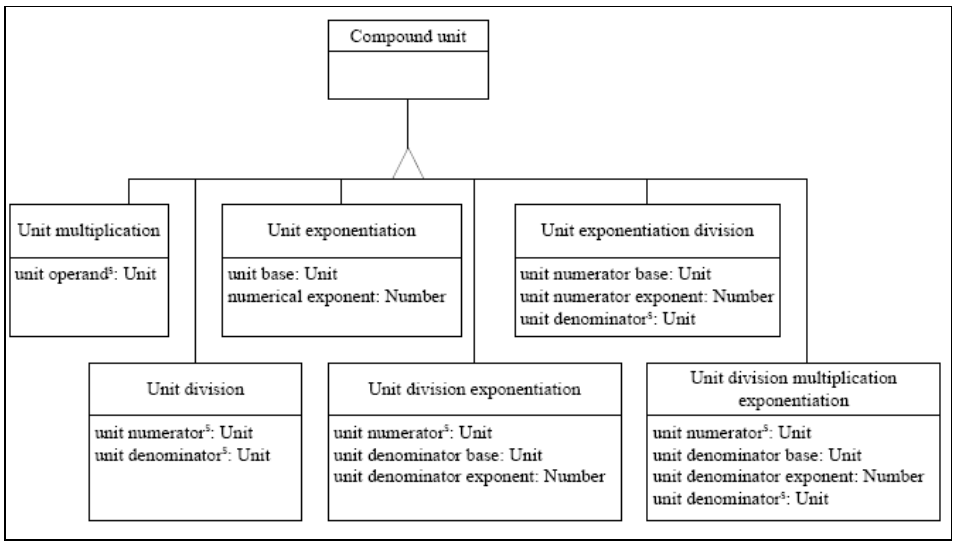


Fig. 23.- Class diagram of the concept compound unit.

As happens to the concepts we already introduce in previous paragraphs the definition of **Dimension** has different hint for the different authors. Gruber [92] defines a physical dimension as a class of scalars with some important algebraic properties. Physical Dimension also should provide necessary conditions for comparing quantities; two quantities are comparable only if they are of the same physical dimension.
The next picture summarizes the definition of QUDT [95]:

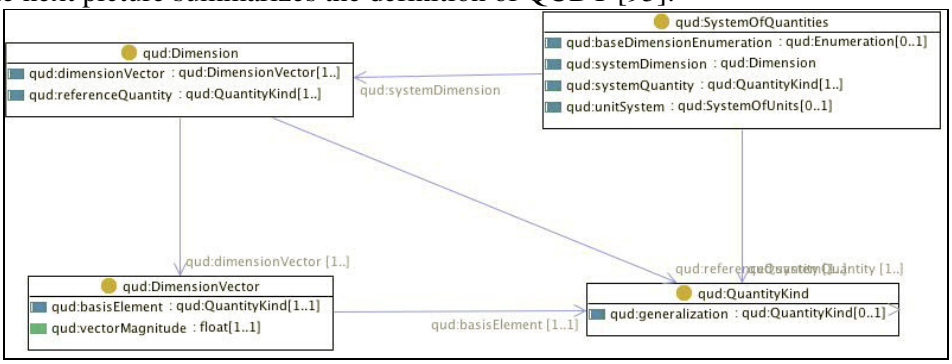


Fig. 24.- The QUDT OWL model of Dimensions.

The authors of the paper define Dimension from a "practical" point of view: Quantity is a kind of systems that defined base and derived sets have certain mathematical properties that permit quantity kinds to be manipulated symbolically. The construction goes as follows: Assign a distinct dimension symbol to each base quantity kind. For each derived quantity kind, take the formula that expresses it in terms of the base quantity kinds and replace every occurrence of a base quantity with its symbol. This is the dimension symbol for the derived quantity kind. In this way, every quantity kind maps to a dimension symbol of the form:

$$\dim Q = (B_1)^{d1}(B_2)^{d2}\ldots(B_n)^{dn}$$

Here $\{B_1,\ldots,B_n\}$ are the dimension symbols for the base quantities and $\{d_1,\ldots,d_n\}$ are rational numbers. Typically, the values of the d_i are between -3 and 3, however magnitudes as high as 7 are required to cover the range of quantity kinds currently defined. Using the multiplication identity for exponents $A^nA^m = A^{n+m}$ one can show that the set of dimension symbols is homomorphic to an n-dimensional vector space over the rational numbers. Multiplication of quantity kinds corresponds to vector addition, division corresponds to vector subtraction and inverting a quantity kind corresponds to computing the additive inverse of its dimension vector.
In some cases, distinct quantity kinds may have the same dimension symbol. This often occurs in cases where physical laws are discovered and formalized independently from each other, but are reduced to the same base quantity kinds. Perhaps the most commonly quoted example is the dimensional equivalence of mechanical torque and energy. Both have the same dimensions (L^2MT^{-2}) but are defined very differently. One consequence of the equivalence is that the same units of measure are applicable to both. One salient difference between the two in this example is that torque is a pseudo-vector while energy is a scalar. However, this distinction (value type) is not accounted for in the quantity kind system formalism.
This definition is the same as the one provided by [97] which defines that the Dimension of a quantity points out in terms of which base quantities, for a given system of units, that quantity can be expressed, using exponents to express the mathematical relation. For example, in SI, the dimension of force has length exponent 1, mass exponent 1, time exponent –2, electric current exponent 0, temperature exponent 0, amount of substance exponent 0 and luminous intensity exponent 0. Rijgersberg's definition differs from the previous one in the sense that it relates dimensions to *System of Units*. SI uses mass in its dimensions, whereas the British system has force as an elementary dimension. Although strictly spoken a dimension is an abstraction of a (compound) quantity, in practice there is no real difference between the expression of a quantity in terms of base quantities or in terms of its dimension. It is ontologically represented in the following picture:

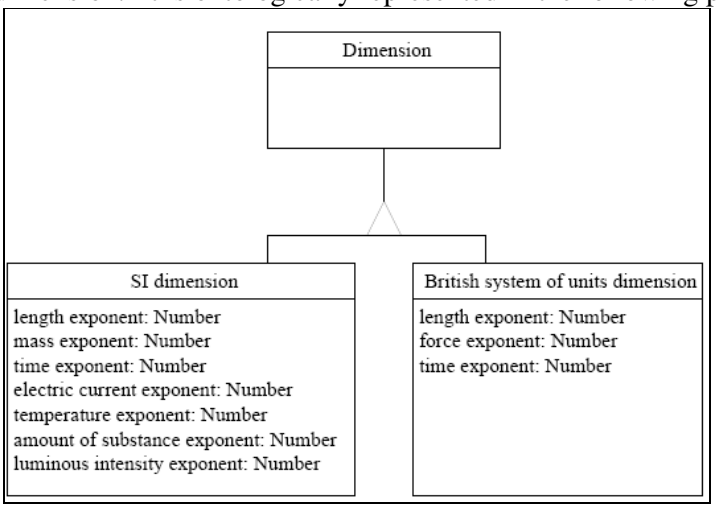


Fig. 25.- Class diagram of the concept dimension.

As we previously said, the Physical Dimension of a quantity distinguishes it from other types of quantities [92, 98] and Non Physical Dimensions are also possible, such as the amount of money. Dimensions tell us something about the quantity that is invariant over models and measurement.
One of the advantages of using Dimensions is that they can be used to verify the consistency of mathematical expressions by "checking the units". Inconsistent use of units points to sloppy or even erroneous modeling [97]. This idea is based on the one introduced by [92]: **Dimensional homogeneity** is a prerequisite to unit conversion and other algebraic operations on quantities.
A special kind of Quantities and Units, already mentioned in previous paragraphs, are **Dimensionless Quantities and Units or quantities of dimension one** [95]**:** they are those for which all the exponents of the factors corresponding to the base quantities in its quantity dimension are zero: counts, ratios and plane angles are examples of dimensionless quantities. Gruber [93] refers to this as **Constant** quantities whose physical dimension is the identity dimension are called, paradoxically, **dimensionless quantities**.
As picture 37 shows **Magnitude** is related to the Dimension concept: it is an attribute of the DimensionVector. Gruber [92] says the **Magnitude** of a physical quantity is not a property of the quantity, it is given by a binary function that maps a quantity and unit of measure to a numeric value (a dimensionless quantity).
The final component of an ontological definition of a Unit of Measurement is the System of Quantities and Units, as it was recommended by Rijgesberg [97]: a profound knowledge of the fundaments of physical unit systems as defined by the major standardization bodies will help to create an Ontology fitting the requeriments introduced in point 3.4.3. The more concise definition of a **System of Units** is the one that Gruber introduce in [92]: A **System of Units** is a class of units defined by composition from a base set of units, such that every instance of the class is "standard" unit for a physical dimension and every physical dimension has an associated unit. This is an interesting representation problem, because both the set of units and the space of physical dimensions are conventions and both are constrained (but not determined) by the background domain theory assumed in a model. The set of dimensions and their mutual relationships are determined by a physical theory, while the choice of units for each dimension is a measurement convention. The concept of system of units is defined so that commitments to physical theories, sets of fundamental dimensions and standard units are independent. To define a system of units, the model builder chooses a set of fundamental dimensions that are orthogonal (i.e., not compassable from each other). According to this physical theory, mass and time are orthogonal, but force and mass are not. The set of fundamental dimensions determines the space of possible quantities that can be described in this system: those whose physical dimensions are some algebraic combination of the fundamental dimensions. For each of the fundamental dimensions, the model builder chooses a standard unit of that dimension; these are called the **base-units** of the system. Then every other standard unit in the system is a composition (using * and expt) of units from the base set.
A similar definition and explanation is given by Masters [95]. This paper divided the definition in two kinds of system:

- The **System of Quantity Kinds**: is a set of one or more quantity kinds together with a set of zero or more algebraic equations that define relationships between quantity kinds in the set. Here they introduce a different quantity kind with the former definition: they extend it by saying that, in the physical sciences, the equations relating quantity kinds are typically physical laws and definitional relations and constants of proportionality. They put as examples the Newton's First Law of Motion, Coulomb's Law and the definition of velocity as the instantaneous change in position.
- The **System of Units**: as a set of units which are chosen as the reference scales for some set of quantity kinds together with the definitions of each unit. Units may be defined by experimental observation or by proportion to another unit not included in the system. If the unit system is explicitly associated with a quantity kind system, then the unit system must define at least one unit for each quantity kind.

This paper also reflects the relation between the Units and their Reference Systems as shown in picture 37.

This relation is also shown in the paper of Rijgesberg, as shown in figure 38. This paper does not give a "formal" definition of a System of units but it introduces the problems their introduction and creation try to solve in order to make the reader understand their capabilities and objectives: over centuries, an enormous number of units have been proposed. Many countries and regions had and still have their own units or versions of units. This has caused severe problems in science, but also in economy, trade and everyday life. People have and had problems understanding each other due to a lack of standardization of units. In an effort to organize units in a proper way, they have been grouped in terms of **Systems of Units**. As with previous definitions, they say within a system of units the relations between the units it defines are postulated, in the form of mathematical expressions, often involving conversion factors. All units are either base units or combinations of *base units*, so-called *derived units*. Derived units can either be singular (e.g., newton in SI) or compound (e.g., newton meter).

They introduce the concept of **Coherent Systems**, meaning that derived units are related by a factor of 1 to the base units and all scaling factors are expressed in terms of prefixes. For example SI and CGS are Coherent Systems while other systems are not as for example the British system of units.

Previous paragraph introduce some of the most used Systems all around the world:

- the **International System of Units [94]** (abbreviated **SI** from the French le Système international d'unités) is the modern form of the metric system and is generally a system of units of measurement devised around seven base units and the convenience of the number ten. It is the world's most widely used system of measurement, both in everyday commerce and in science. Interested reader can take a look to the [100] and [101] references.
- The **centimeter-gram-second** system (abbreviated **CGS** or **cgs**) [102] is a metric system of physical units based on centimeter as the unit of length, gram as a unit of mass and second as a unit of time. All CGS mechanical units are unambiguously derived from these three base units, but there are several different ways of extending the CGS system to cover

electromagnetism. The CGS system has been largely supplanted by the MKS system, based on meter, kilogram and second. MKS was in turn extended and replaced by the International System of Units (SI). The latter adopts the three base units of MKS, plus the ampere, mole, candela and Kelvin. In many fields of science and engineering, SI is the only system of units in use. However, there remain certain subfields where CGS is prevalent.

- **Imperial units or the imperial system** [101] is a system of units, first defined in the British Weights and Measures Act of 1824, later refined (until 1959) and reduced. The system came into official use across the British Empire. By the late 20th century most nations of the former empire had officially adopted the metric system as their main system of measurement. A more detailed description can be found in [105].

7. Description of the Prototype System.

We now briefly describe the architecture of the Prototype System we implemented in order to allow the semantic request of contract from a hypothetical user. First we introduce the architectural definition of the modules and their iterations and then we introduce some of the tests we have performed with them.

7.1. Required Functionalities.

The high level functionalities that are required of the system can be summarized as:

- The system should be able to automatically transform documents from eContract Legal XML to OWL allowing the use of a semantic reasoner to answer user questions.
- Some basic concept "translation" should be necessary to support multilingual contract definition.
- A reasoning engine should have to be developed in order to allow users to look for a contract solving his requests.
- A basic user interface should be developed to allow users access to the modules functionalities.

The diagram on the next page summarizes the different modules' interaction. First we describe the meaning of the colors and forms:

- Green Files represent the eContract in LegalXML format, which are provided by the user to create a new offer or to replace and old one.
- Brown Files are the translated LegalXML eContracts as they are stored by the Concept Translation Service. This color is used to represent the Ontology used by the translator and the synonym files two.
- Red Files Represents the original ontology (that is, the backbone of concepts and relations), the eContract LegalXML DTD used to validate the user offer files and the offers ontology in OWL format.
- Red arrows represent input and output connections between processes and files.
- Black dot arrows represent calls between processes to perform its operations.
- Green arrows are used to show the input connection from between a file and a process.
- eContract (offers) publication: During Opaals' Phase II we developed a program which transforms Contract from MS Excel format (it is not Open Source software but it is widely used within the computing world) to eContract Legal XML. This means to put the contract digital document in a standard format.

  Although some basic searches are possible using XPATH capabilities a real semantic search is not feasible.

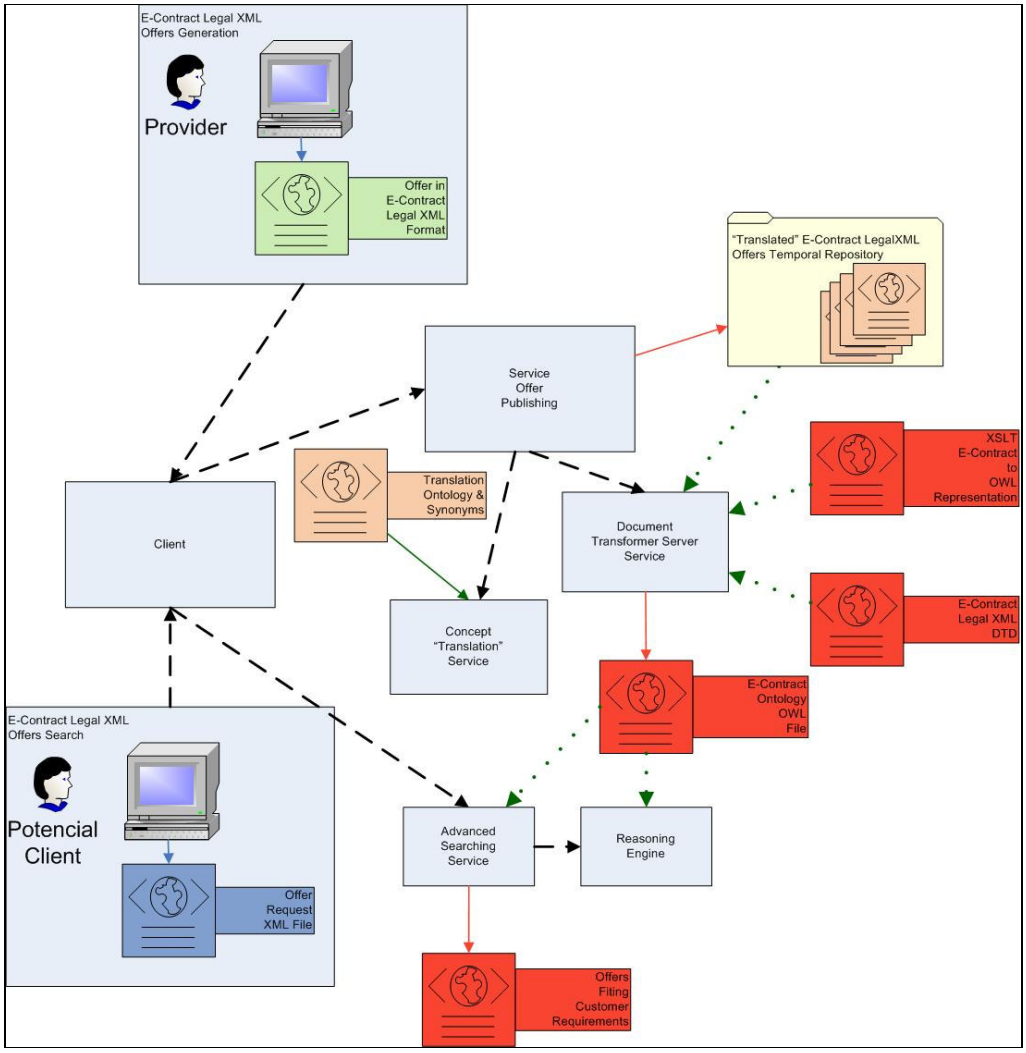


Fig. 26.- SFinder V2.0. High Level Definition.

The implemented processes, which are detailed in the next paragraphs are:

- OfferPublishingService: this is the user interface and, from the previous diagram, it includes the Advanced Searching Services Functionalities as it can be observed in the following picture.

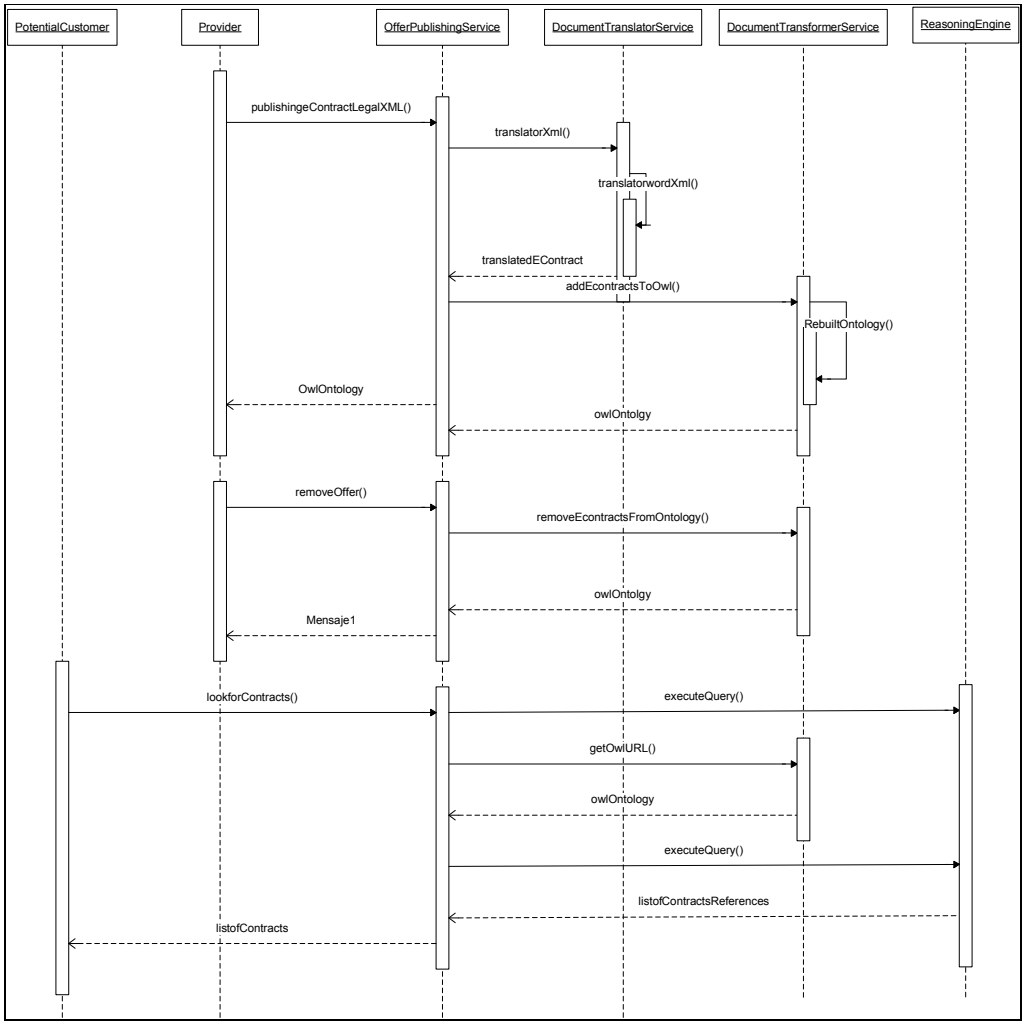


Fig. 27.- sFinder v2.0 Sequence Diagram.

- DocumentTranslatorService: giving an eContract LegalXML document, it is translated from its original language to the "normalized" representations in English: that is, the DocumentTranslatorService substitutes the appereance of a synonym by the concept it is associated to.
- DocumentTransformerService: this service manages the instaces of contracts within an instance of the ontology. For example, giving an eContract LegalXML document "translated" to English, it add to an ontology. This component is responsible of maintaining the ontology coherence: each time and operation is performed against the ontology it is completely rebuilt. To perform it each time a provider requests the addition or the deletion of a contract from a given ontology intance the DocumentTransformerService follows the lists of contracts which is associated to the ontology and , one by one, confirm they are still available before adding them to the new release of the ontology. If the module find a contract which is nos accessible just ignored it.

- ReasoningEngine: giving the IRI of an ontology and a query expressed on DL Query Language it returns the set of instances making the axioms true.

7.2. Equivalences and Differences with Software Composition Services Processes.

Many relations can be established between the processes described in the previous paragraphs and the processes that references architectures and different DCS initiatives and projects purposes, like NexOf RA [94] and WSMO [95].
Saving the distances between our high business level approximation and the business level and high infrastructure level of the SWS DCS initiatives, we list relations and differences that can be established:

- The main difference between the two transformation processes is that the objective of the eContract is to obtain a common repository (that does not mean centralized) of eContracts to answer user requests and the service owner location. This objective is common to one of the objectives of the WSDL transformation process but this process has another and likewise important objective: to enable/realise the execution of the described service, as we show below, that means to define the way in which the data that the client and the provider shared should have to be modified.
- Both processes are based on standards: eContract LegalXML is a representational standard for Contract supported by OASIS members while the SWS DCS has WSDL as the starting point. These digital representations of documents are converted to OWL representations in order to support automatic classification and searching processes. In some way these representational conversion processes are normalization ones as they move from a flexible representational framework to a rigid one which is imposed by an ontological representation of the shared knowledge.
- Both processes, for different reasons and through different ways, suppose the semantic annotation of the original documents. For our processes this means to “translate”, that is the substitution of the synonyms of a concept by the shared name of it, and by modifying the document structure from the Legal-XML representation to the OWL one. In the cases of the WS, the process to follow is that the WSDL representation is annotated against the Domain ontology and the ontological representation of a WS (either OWL-S or WSMO) and a SAWSDL file is obtained and added to the ontology.
- When our process to add eContracts to an ontology is basically a schema transformation (similar to the grounding processes as we show below) the WSDL to SAWDSL transformation is an annotation one: new attributes are added to the WSDL tags to map these tags to the desired concepts in the reference Ontology (domain + WS representation). As a consequence our process is simpler and can be completely automated while the WSDL to SAWSDL transformation process, at least for the moment, usually need some manually annotation processes and revision.

Another point of similarity of the eContract and the WSDL transformation processes, for the second to the service consumption is that the eContract Legal XML to OWL representation transformation is equivalent to the grounding processes: both take documents which are defined according to a known data

schema to another schema and they specify the way in which data should be modified to fix the destination schema.

7.3. Some Snapshots of the Developed Work.

This section summarizes the work and results of the developments we have performed in order to turn a "human understandable" eContract LegalXML digital document into a machine-readable one.
We developed 3 modules offered as (web) services which "implement" (what we did was to adapt OS modules to our necessities):

- The eContract translator.
- The eContract conversion, from LegalXML to OWL, the ontology maintenance (deletion, upgrading and addition of eContrac instances).
- The eContract reasoning module.

These modules are developed in Java and we deployed them on Open Bip [163], an Open Source Experimental P2P Web Services Integration Platform. Open Bip follows the OSGI philosophy (everything is a service) and it is following a Bottom-Up building approximation from the core services (execution framework, hardware access, communication services …) to the high level ones (Semantic Service Annotation, Service Publishing, …[Dinamyc] Service Composition). Althought Open Bip is in a very early stage of the development, when we started the development of the modules described in previous paragraphs, it was more advanced than Flypeer and we decided to used it on a basis that would not going to stop our development.
An additional "global" client has been developed to allow users to include and manage offers (eContract Documents) on the searching platform. This client has been developed as a text console one in the base of centered the interest and the other three modules which could be the heart and brain of futures developments.
We show here some snapshots of the deployment of the modules in order to clarify their interactions and the user interactions with the "complete" system.

7.3.1. Main Menu.

When the complete system is working, a user can interoperate with it thought a text console. The next picture shows the user options:

```
Select an option:
        1 - Add eContract to ontology
        2 - Remove eContract to ontology
        3 - Search eContract in ontology
        4 - Exit
```

Fig. 28.- User Options.

7.3.2. Adding a New Contract.

If a user chooses option 1 he will be able to add a new Contract (offer) definition to the existing ontology:

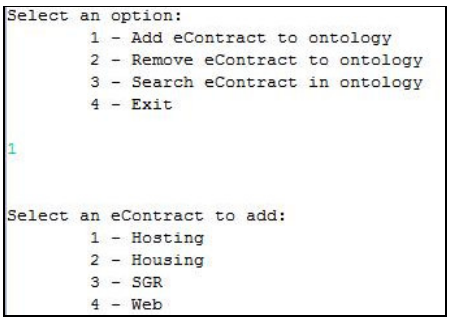


Fig. 29.- Adding a new Offer Options.

After accepting this option, the information regarding the kind of contract to be added is demanded. We have simplified client implementation by considering that each provider only can have a contract of each type and that they are stored in a predefined file (that does not mean that the contract content can be changed). According to this and after the user has chosen an option, the corresponding offer definition will be added (or upgraded) to the ontology. Once the kind of contract has been chosen the addition process starts.

The first task performed by the server is the contract "translation", that is the labels are expressed in the selected common language. The next picture shows the "translator" module:

Fig. 30.- Translator Message.

When the translator process finishes, the next step is to add the translated version of the eContract to the ontology. Although it can be performed by adding each of the final entities we find its implementation easier by transforming the eContranct format to an owl representation of it. This is done by the transformer module and uses an XSLT document as guide to perform the desired document format conversion.

Fig. 31.- Ontology manager Messages.

7.3.3. Removing an Offer.

Users are provided of a removing functionality. It is accessed by chosing 2 in the main menu.

The supposition to simplify the client implementation is the same as in our previous case and then the user is asked for the kind of contract to remove. After a while a message such as shown below is received by the user:

Fig. 32.- Returned Message After Removing an Offer.

7.3.4. Looking for the Desired Offers.

Publishing contracts is performed by the reasoning module which is accessed by selecting the option number 3 in the main menu.

Although the Reasoning module is able to execute any correct DL query we have restricted it in the examples to find those contracts that fit each of the possible kinds. After receiving the user query the reasoner returns the available contracts.

Fig. 33.- Searching Answer.

8. Conclusions and Future Work.

During the last 2 years and in particular in Phase III of Opaals we have worked at reducing the gap which exists between the "real world" business offers and contracts and their equivalents in the world of computing services, which in the Web environment are WSDL files.

There are many initiatives, projects, and tools which are trying to solve or which already solved the problem to create Dynamic Composed Web Services. The idea which has guided our work is to take advantage of the existing knowledge to try to solve the problems we established in point 2.4 of this deliverable.

The conclusions of our Work are:

- The tools for creating and using ontologies are in very early stages, standard interfaces and formats change and there is not always compatibility with previous versions. For example when we introduced the reasoning module we suffered a compatibility problem between OWL API 2 and 3 which delayed our work.
- Reducing the gap between human readable documents and machine readable documents is highly dependent in the possibility of structuring the human document. In our case the offers/contract clauses are expressed in a standard format and are created within a concrete domain. It reduces the complexity and difficulty of the problem to solve: we solve it by translating contract to "conceptual" contract and converting this to owl. Even the obtained results are good they are only two steps to a complete integrated framework. We think that a full solution should consider more dimensions than the one we introduced here: document scheme conversion, language translation, unit conversion, coins conversions (which extends the unit conversions with other problems), or intelligent querying. For example natural language processing or "business" (non executable) level service/product composition are dimension to explore what we did not consider.
- From the legal point of view the realization of the Digital Ecosystem for real world business composition has to solve several legal problems as they are not solved in the real world, because they do not exist in the real word. This shows the existence of a regulatory issue, whose treatment is faced from a more global perspective in D12.12.

Appendix I. References.

A.II.1. "Computing" References.

[1] W3C. Extensible Markup Language (XML). http://www.w3.org/XML/ , November 26th of 2008

[2] OWL Web Ontology Language. Overview. http://www.w3.org/TR/owl-features/ .

[3] OWL 2 Web Ontology Language. Document Overview. http://www.w3.org/TR/2009/PR-owl2-overview-20090922/ .

[4] OntoWeb. http://www.ontoweb.org/ .

[5] Web-Ontology (WebOnt) Working Group. *http://www.w3.org/2001/sw/WebOnt/*

[6] OWL 2 Web Ontology Language. Primer. http://www.w3.org/TR/2009/PR-owl2-primer-20090922/all.pdf .

[7] Description Logics. http://www.dl.kr.org/ .

[8] Converting business documents:a classification of problems and solutionsusingXML/XSLT. Wustner, E. Hotzel, T. Buxmann, P. Freiberg University of Technology; Advanced Issues of E-Commerce and Web-Based Information Systems, 2002. (WECWIS 2002). Proceedings. Fourth IEEE International Workshopon Publication, 2002, pages 54-61 ISSN: 530-1354 ISBN:0-7695-1567-3.

[9] D. Nardi, R. J. Brachman. An Introduction to Description Logics. In the Description Logic Handbook, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 5-44.

[10] Mendling, J., Neumann, G. and Nüttgens, M. A Comparison of XML Interchange Formats for Business Process Modelling. In Proceedings of the EMISA 2004 Workshop "Information Systems in E-Business and E-Government", Vol. 56 of Lecture Notes in Informatics (LNI), 2004.

[11] T. Hornung, A. Koschmider and J.Mendling, Integration of heterogeneous BPM Schemas: The Case of XPDL and BPEL,in *CAiSE,* 2006.

[12] G. Tremblay and J. Chae. Towards specifying contracts and protocols for web services. In H. Mili and F. Khendek, editors, Proceedings of the MCeTech Montreal Conference on eTechnologies, pages 73{85, Montreal, Canada, January 2005.

[13] W3C: XSLT, http://www.w3.org/TR/xslt, May 1999.

[14] W3C: XSLT 2.0, http://www.w3.org/TR/xslt20/, January 2007.

[15] Xalan for java home page. http://xml.apache.org/xalan-j/

[16] XSLT Processor Benchmarks. http://www.xml.com/pub/a/2001/03/28/xsltmark/index.html, March 23th of 2001.

[17] Saxon: Anatomy of an XSLT processor http://www.ibm.com/developerworks/xml/library/x-xslt2/

[18] Volker Hoyer, Modeling Collaborative e-Business Processes in SME Environments. JIST 5(2), 2008.

[19] What kind of language is XSLT?, Michael Kay, http://www.ibm.com/developerworks/xml/library/x-xslt/?dwzone=x, April 25th of 2005.

[20] Saxon The XSLT and XQuery Processor, home page.

http://saxon.sourceforge.net/
Java XML transformer for processing XML
[21] IDABC Home Page. http://ec.europa.eu/idabc/en/home
[22] WSMO Architecture. http://www.wsmo.org/TR/d13/d13.4/v0.1/#Scope. June 24th, 2004.
[23] Schmitz, V. et al. (2003). "XML data modeling concepts in B2B catalog standards". In: Proceedings of IADIS International Conference e-Society 2003 (ES 2003), June 3-6, 2003, Lisbon, Portugal, pp. 227-234.
[24] Michael Kay. What kind of language is XSLT? An analysis and overview. Available from http://www.ibm.com/developerworks/library/x-xslt/ , April 2004.
[25] David Mirra. An Overview of XSLT. August 1, 2007.
[26] XSLT, Perl, Haskell, & a word on language design. Tom Moertel. http://www.kuro5hin.org/story/2002/1/15/1562/95011. January 15th of 2002.
[27] Sax home page. http://www.saxproject.org/
[28] Sax on wikipedia. http://en.wikipedia.org/wiki/Simple_API_for_XML
[30] The W3C DOM homepage. http://www.w3.org/DOM/
[31] The Robin Cover's DOM Pages. http://xml.coverpages.org/dom.html, August 23rd of 2003.
[32] DOM Primer Part I. http://www.w3c.rl.ac.uk/primers/dom/domprimer-p1.htm. Oxford Brookes University 2002
[33] DOM Primer Part II. http://www.w3c.rl.ac.uk/primers/dom/domprimer-p2.htm . Oxford Brookes University 2002
[34] Benchmarking XSLT processors. http://www.davidpashley.com/articles/xslt-benchmarks.html.
[35] XSLT Benchmark. http://www.dpawson.co.uk/xsl/sect4/N9883.html.
[36] NeOn Project Home Page http://www.neon-toolkit.org/
[37] Kaon2 http://kaon2.semanticweb.org/
[38] U. Hustadt, B. Motik, U. Sattler. **Reducing SHIQ- Descrption Logic to Disjunctive Datalog Programs**. *Proc. of the 9th International Conference on Knowledge Representation and Reasoning (KR2004)*, June 2004, Whistler, Canada, pp. 152-162.
[39] B. Motik. **Reasoning in Description Logics using Resolution and Deductive Databases**. *PhD Thesis, University of Karlsruhe, Karlsruhe, Germany, January 2006.*

[40] B. Motik and U. Sattler. **A Comparison of Reasoning Techniques for Querying Large Description Logic A-Boxes**. *Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, Phnom Penh, Cambodia, November, 2006.

[41] Open Knowledge Base Connectivity Home Page. http://www.ai.sri.com/~okbc/
[42] Protégé Home Page. http://protege.stanford.edu/
[43] The Protégé Reasoner API http://protegewiki.stanford.edu/index.php/ProtegeReasonerAPI
[44] Pellet Reasoner Home Page. http://clarkparsia.com/pellet
[45] Jürgen Bock, Tuvshintur Tserendorj, Yongchun Xu, Jens Wissmann and Stephan Grimm. A Reasoning Broker Framework for Protégé. http://protege.stanford.edu/conference/2009/abstracts/S9P1Bock.pdf

[46] DIG 2.0: The DIG Description Logic Interface. http://dig.cs.manchester.ac.uk/
[47] Using the protégé-owl reasoner api. http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html
[48] Uli Sattler. Description Logic Reasoners List. http://www.cs.man.ac.uk/~sattler/reasoners.html , the University of Manchester.
[49] RACER Home Page. http://www.sts.tu-harburg.de/~r.f.moeller/racer/
[50] P.F. Patel-Schneider and B. Swartout. Description logic knowledge representation system specification from the krss group of the arpa knowledge sharing effort. Technical report, Bell Labs, 1993.
[51] FACT Home Page. http://www.cs.man.ac.uk/~horrocks/FaCT/
[52] Camelot Project Home Page. http://www.cs.man.ac.uk/~horrocks/Camelot/
[53] FACT++ Home Page http://owl.man.ac.uk/factplusplus/
[54] The OWL API Home Page. http://owlapi.sourceforge.net/index.html
[55] Pellet Home Page. http://clarkparsia.com/pellet
[56] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz. Pellet A practical OWL-DL reasoner Journal of Web Semantics, 5(2), 2007.
[57] Sean Bechhofer. The DIG description logic interface: DIG/1.1. In Proceedings of the 2003 Description Logic Workshop (DL 2003), 2003.
[58] Uli Sattler. A list of Description Logic Reasoners. http://www.cs.man.ac.uk/~sattler/reasoners.html
[59] Legal XML Home Page. http://www.legalxml.org/
[60] Dublin Core Home Page. http://dublincore.org/metadata-basics/
[61] Gruber, T. 1994. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. IJHCS, 43(5/6): 907-928.
[62] Feature Sinopsis of Owl Lite and OWL. http://www.ksl.stanford.edu/people/dlm/webont/OWLFeatureSynopsisJuly29.htm
[63] OWL DL Semantics. http://www.obitko.com/tutorials/ontologies-semantic-web/owl-dl-semantics.html
[64] OASIS Home Page http://www.oasis-open.org/home/index.php
[65] Juergen Bock, Tuvshintur Tserendorj, Yongchun Xu, Jens Wissmann and Stephan Grimm. A Reasoning Broker Framework for OWL. Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009.
[66] Golbreich, C., Wallace, E.K.: OWL 2 Web Ontology Language: New Features and Rationale. W3C working draft, W3C (June 2009)..
[67] Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language:Profiles. W3C candidate recommendation, W3C (June 2009).
[68] Bock, J., Haase, P., Ji, Q., Volz, R.: Benchmarking OWL Reasoners. In: Proc. of the ARea2008 Workshop, Tenerife, Spain (June 2008).
[69] Gardiner, T., Tsarkov, D., Horrocks, I.: Framework For an Automated Comparison of Description Logic Reasoners. In: Proc. of the 5th Int. Semantic Web Conf. (ISWC). Volume 4273 of LNCS., Berlin, Springer (2006) 654–667
[70] Tserendorj, T., Rudolph, S., Kr¨otzsch, M., Hitzler, P.: Approximate OWLReasoning with Screech. In: Proc. of the 2nd Int. Conf. on Web Reasoning and Rule Systems. Volume 5341 of LNCS., Berlin, Springer (October 2008) 165–

180
[71] VICODI Home Page. http://www.vicodi.org/about.htm
[72] SWRC Home Page. http://ontoware.org/swrc/
[73] LUMB Home Page. Y. Guo, Z. Pan and J. Heflin. LUBM: A benchmark for owl knowledge base systems. J. Web Sem., 3(2-3):158–182, 2005.
[74] The Wine Ontology Definition. http://www.w3.org/2001/sw/WebOnt/guide-src/wine.rdf
[75] J. Broekstra, A. Kampman, F. van Harmelen, Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema , First International Semantic Web Conference (ISWC'02), Sardinia, ITAly, June 9-12, 2002.
[76] Onto-Knowledge Project. http://www.ist-world.org/ProjectDetails.aspx?ProjectId=2ef2a30e9cce4ee784938bc80d232001
[77] OWLIM Home Page. http://www.ontotext.com/owlim/
[78] OWLIM. Pragmatic OWL Semantic Reposity.
[79] TRREE Home Page. http://www.ontotext.com/trree/
[80] Ontotext Lab. RDF(S), Rules and OWL Dialects. Introductory web page.
[81] Rob Shearer, Boris Motik and Ian Horrocks. Hermit: A highly-ecient owl reasoner. In Catherine Dolbear, Alan Ruttenberg and Ulrike Sattler, editors, OWLED 2008, volume 432 of CEUR Workshop Proceedings. CEUR-WS.org, 2008.
[82] Matthew Horridge, Sean Bechhofer. The OWL API: A Java API for Working with OWL 2 Ontologies. OWLED 2009, 6th OWL Experienced and Directions Workshop, Chantilly, Virginia, October 2009.
[83] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax.
[84] The Jena API Home Page.
[85] HP Labs Semantic Web Research.
[86] K. Wilkinson, C. Sayers, H. Kuno and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In SWDB, 2003.
[87] R.J. Brachman and J. Schmolze, "An Overview of the KL-ONE Knowledge Representation System", Cognitive Sci 9(2), 1985.
[88] Semantic Reasoner Definition in Wikipedia.
[89] OWL Working Group. Reasoners Implementations Page.
[90] C´assia Trojahn, Paulo Quaresma, and Renata Vieira. 2008. A framework for multilingual ontology mapping. In Proceedings of the 6th International Language Resources.
[91] D. Bonino, F. Corno, L. Farinetti, A. Ferrato, Multilingual Semantic Elaboration in the DOSE platform, SAC 2004, ACM Symposium on Applied Computing, March 14-17, 2004, Nicosia, Cyprus
[92] Vossen P. (1998) EuroWordNet: Building a Multilingual Database with Wordnets for European Languages. In: K. Choukri, D. Fry, M. Nilsson (eds), The ELRA Newsletter, 3, ISSN: 1026-8200.
[93] T. Declerck, A.G. Perez, O. Vela, Z. Ganter, and D. Manzano-Macho: Multilingual
Lexical Semantic Resources for Ontology Translation, In Proceedings of the International Conference on Language Resources and Evaluation (LREC), pp. 1492- 1495.

[94] Nexof-RA Home Page. http://www.nexof-ra.eu/
[95] WSMO Home Page. http://www.wsmo.org/
[96] Wikipedia page. http://en.wikipedia.org/wiki/Xslt
[164] Swoop Home Page. http://www.mindswap.org/2004/SWOOP/
[165] Tserendorj, T., Rudolph, S., Kr¨otzsch, M., Hitzler, P.: Approximate OWLReasoning
with Screech. In: Proc. of the 2nd Int. Conf. on Web Reasoning and
Rule Systems. Volume 5341 of LNCS., Berlin, Springer (October 2008) 165–180
[166] Tserendorj, T., Grimm, S., Hitzler, P.: Approximate Instance Retrieval. Technical report, FZI Research Center for Information Technology, Karlsruhe, Germany (December 2008) availavle at http://www.aifb.uni-karlsruhe.de/WBS/
[167] Raskin, R. 2006. Guide to SWEET ontologies. Available at: http://sweet.jpl.nasa.gov/. Accessed 18 October 2006.

A.I.2."Legal" References.

[97] See Deliverable 6.9. OPAALS Project. 7.1 Digital Ecosystems. "A DE can be defined as a combination of a specific technological infrastructure, the so-called "digital environment", and those entities or "digital components" (software, services, business processes or models, contractual frameworks, law, knowledge, etc.) which have been formalized, digitalised and transported within the ecosystem and which can be further processed by humans or by computers."
[98] Extract from "Dynamic Service Composition and Its Applicability to E-Business Software Systems – The ICARIS Experience (2000)". Vladimir Tosic, David Mennie, and Bernard Pagurek. Department of Systems and Computer Engineering, Carleton University.
[99] See FERNÁNDEZ. La Cooperación Empresarial. Información Comercial Española, no. 693, 1991, p. 27. Also FERNÁNDEZ, Innovación, Tecnología y Alianzas Estratégicas: factores clave de la competencia. Madrid. Civitas, 1996, p. 463.
[100] Sections 1. 4. 1. and 1. 4. 2.
[101] See ALFARO AGUILA-REAL, Las condiciones generales de la contratación. Estudio de las disposiciones generales, Col. Monografías, Civitas, Madrid, 1991, p. 275 ff.
[102] GUISADO MORENO, Formación y perfección del contrato en Internet, 1st ed., Marcial Pons, Madrid, 2004, p. 178
[103] http://www.uncitral.org/pdf/english/texts/sales/cisg/CISG.pdf
[104] http://www.unidroit.org/english/principles/contracts/main.htm
[105] http://www.jus.uio.no/lm/eu.contract.principles.1998/doc.html#105
[106] SANJUÁN Y MUÑOZ, Incorporación de las nuevas tecnologías en el comercio: aspectos legales, Consejo General del Poder Judicial (Coeditada en colaboración con la Conselleria de Justicia y Administraciones Publicas de la Generalitat Valenciana), 1st ed., Madrid, 2009, p. 240.
[107] Article 13 of the Legislative ITAlian Decree 70/2003 solves expressly this question, which completes Article 1335 of the Codice civile (J. C. MENÉNDEZ MATO, El contrato vía Internet, 1st ed., J.M. Bosch Editor, Barcelona, 2005, p. 54). In the same way can be understood Article 1262 of the Spanish Civil Code and Article 54 of the Spanish Commercial Code.
[108] FLORES DOÑA, Impacto del comercio electrónico en el derecho de la

contratación, 1st ed., EDERSA, Madrid, 2002, p. 160.
[109] DIEZ PICAZO, L.: Los sujetos de la relación obligatoria, en Fundamentos de Derecho Civil Patrimonial II. Las Relaciones Obligatorias. Madrid, 1996, p. 162 and ff.
[110] SÁNCHEZ ANDRÉS, A. El transporte combinado de mercancías. RDM, no. 135-136, 1975, p. 50 and ff.
[111] DIEZ PICAZO, cit., p. 176.
[112] De la contratación común o general - Contratos de prestación de servicios y realización de obras, en Tratado de contratos. Bercovitz Rodríguez-Cano, R. (Dir.) / Moralejo Imbernón, N., Quicios Molina, S. (Coord.). Tirant lo Blanch, 2009.
[113] MARTINEZ DE AGUIRRE - El contrato, en Curso de Derecho Civil II. Derecho de obligaciones. Coord. Martínez de Aguirre. COLEX, 2000, p. 114.
[114] LÓPEZ RUEDA.: El régimen jurídico del Transporte multimodal, colección Biblioteca de Derecho de los Negocios, Madrid, 2000.
[115] PILOÑETA ALONSO, El subtransporte: análisis crítico y teórico de un concepto práctico, en: MARTINEZ SANZ, F./PETIT LAVALL M. V (Dirs). Aspectos jurídicos y económicos del transporte. Hacia un transporte más seguro, sostenible y eficiente, Vol I, Publicacions de la Universitat Jaume I, Castellón, 2007, p. 763; GABALDÓN GARCÍA: Intermediarios de transporte y portadores contractuales, RDM., no. 238, 2000, p. 1723 ff.
[116] ALONSO SOTO, R.- El contrato de transporte-, in Uría/Menéndez (Dirs.), Curso de Derecho Mercantil, Madrid, 2001, Tomo II, p. 309 ff.
[117] DIEZ PICAZO - La pluralidad de deudores, ob. cit, p. 207
[118] Geneva, 19 May 1956, United Nations.
[119] In this sense, SÁNCHEZ ANDRÉS, cit., p. 83. Cfr. Report of the General Secretary to the Working Group of UNCITRAL 3rd Period of Sesions, Yearbook, vol III, 1972, p. 331 ff.
[120] ORTUÑO BAEZA, M. T. Contrato de transporte. Transporte de cosas. Transporte de personas. Ley de viajes combinados y la contratación turística, Contratación Mercantil, Vol. III, Tirant Lo Blanch, Valencia, 2003, p. 1413 ff.
[121] ZERPA ALEMÁN, P. La responsabilidad en el transporte multimodal, en Vector Plus no. 13, Ed. Fundación Universitaria de las Palmas, 1999, p. 68 ff.
[122] Ad ex. GÓMEZ SEGADE: El transporte combinado de mercancías a la luz del artículo 373 del Código de Comercio", en Estudios Jurídicos en Homenaje al Profesor Alfonso Otero, Santiago de Compostela, 1981, p. 105 ff.
[123] QUINTANA CARLO: "El contrato de transporte terrestre de mercancías" en Contratos Mercantiles, Dir. A. Bercovitz, Pamplona, 2005, p. 931 ff.
[124] VICENT CHULÍA. Introducción al Derecho Mercantil. Tirant lo Blanch, Valencia, 2005. p. 888.
[125] In this sense, PÉREZ DE LA CRUZ, A.: Los contratos turísticos, en Curso de Derecho Mercantil II, Dirs Rodrigo Uría/Aurelio Menéndez, Madrid, 2001, p. 294 f.
[126] See SANCHEZ CALERO, Instituciones de Derecho Mercantil, I y II, 28ª ed., Barcelona, 2005. Also, TUR FAÚNDEZ - El contrato de viaje combinado: notas sobre la Ley 21/1995, de 6 de julio, de regulación de los viajes combinados", Aranzadi Civil, 1996, p.. 209 and ff.

[127] MARTÍN OSANTE, J. M. – Contrato de viaje combinado: resolución y responsabilidades. RDM, num 273, Madrid, 2009, p. 1053 ff.
[128] See also ALFARO J. - Los Costes de Transacción, en Estudios jurídicos en homenaje al profesor Aurelio Menéndez, coord.. Juan Luis Iglesias Prada, Vol 1, 1996, p. 131 ff.
[129] See D12.12
[130] Type of change fixed by the European Central Bank http://www.ecb.int/stats/exchange/eurofxref/html/index.en.html
[131] ALONSO SOTO, R.: El contrato de transporte, en Uría/Menéndez (Dirs.) *Curso de Derecho Mercantil,* Madrid, 2001, Tomo II, pág. 309-334.
[132] ALFARO ÁGUILA-REAL, J.: Los Costes de Transacción, en: *Estudios jurídicos en homenaje al profesor Aurelio Menéndez*, coord.. Juan Luis Iglesias Prada, Vol 1, 1996, pág. 131-162.
[133] ALFARO AGUILA-REAL, J, *Las condiciones generales de la contratación. Estudio de las disposiciones generales*, Col. Monografías, Civitas, Madrid, 1991.
[134] CLEMENTE MEORO, M: La responsabilidad civil de los prestadores de servicios de la sociedad de la información, en *Responsabilidad civil y contratos en Internet. Su regulación en LSSICE*. Granada, 2003, págs 1-116.
[135] DIEZ PICAZO, L.: Los sujetos de la relación obligatoria, en *Fundamentos de Derecho Civil Patrimonial II. Las Relaciones Obligatorias*. Madrid, 1996, págs. 162 y ss.
[136] FERNÁNDEZ, E.: La Cooperación Empresarial, Información Comercial Española, nº 693, 1991, pag 27.
[137] FERNÁNDEZ, E.: *Innovación, Tecnología y Alianzas Estratégicas: factores clave de la competencia*. Madrid . Civitas, 1996.
[138] FLORES DOÑA, M. S. *Impacto del comercio electrónico en el derecho de la contratación*, 1st ed., EDERSA, Madrid, 2002
[139] GABALDÓN GARCÍA: Intermediarios de transporte y portadores contractuales. RDM., núm 238, 2000, pág 1723-1748.
[140] GARROTE FERNÁNDEZ: Acciones civiles contra los prestadores de servicios de intermediación en relación con la actividad de las plataformas P2P. Su regulación en la Ley 34/2002 y en la Ley de Propiedad Intelectual, en Revista de Propiedad Intelectual, num. 16, 2004, pág 81.
[141] GÓMEZ SEGADE: El transporte combinado de mercancías a la luz del artículo 373 del Código de Comercio, en *Estudios Jurídicos en Homenaje al Profesor Alfonso Otero*, Santiago de Compostela, 1981, pág 105 y ss.
[142] GUISADO MORENO, A., *Formación y perfección del contrato en Internet*, 1st ed., Marcial Pons, Madrid, 2004.
[143] HERNÁN CASTILLO, NOGUEROL CARMENA y ROSÓN OLMEDO: Obras en acceso abierto y responsabilidad por hecho ajeno en Internet, en *El Derecho de autor y las nuevas tecnologías, reflexiones sobre la reciente reforma de la Ley de Propiedad Intelectual.* La Ley, Madrid, 2008, págs 42 y ss.
[144] LACRUZ MANTECÓN: Copias privadas y calamidades públicas, en Anuario de Propiedad Intelectual, 2005, págs 393 y ss.
[145] LLANEZA GONZÁLEZ: Aplicación práctica de la LSSI-CE: Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y comercio electrónico, en NJ Bosch, núm 1, 2002.

[146] LÓPEZ RUEDA.: *El régimen jurídico del Transporte multimodal*, colección Biblioteca de Derecho de los Negocios, Madrid, 2000.
[147] MENÉNDEZ MATO, J. C., *El contrato vía Internet*, 1st ed., J.M. Bosch Editor, Barcelona, 2005.
[148] MARTÍN OSANTE, J. M.: Contrato de viaje combinado: resolución y responsabilidades. RDM, num 273, Madrid, 2009. Pags 1053 y ss.
[149] MARTINEZ DE AGUIRRE: El contrato, en *Curso de Derecho Civil II. Derecho de obligaciones*. Coord. Martínez de Aguirre. COLEX, 2000, pag 114.
[150] MENCONI: Obbligazioni di "rísu/toío" e obbligazioni di mezzi, en R. D. C., 1954, parí, I. Págs 188 y ss.
[151] PEGUERA POCH, M: Sólo se que no se nada (efectivamente): la apreciación del conocimiento efectivo y otros problemas de aplicación judicial de la LSSI, en Internet, Derecho y Política, 2009, págs 199-226.
[152] PÉREZ DE LA CRUZ, A.: Los contratos turísticos, en *Curso de Derecho Mercantil*, Dirs Rodrigo Uría/Aurelio Menéndez, Madrid, 2001, Tomo II pág 294-295.
[153] PILOÑETA ALONSO.: El subtransporte: análisis crítico y teórico de un concepto práctico, en MARTINEZ SANZ, F./PETIT LAVALL M. V (Dirs). *Aspectos jurídicos y económicos del transporte. Hacia un transporte más seguro, sostenible y eficiente,* Vol I, Publicacions de la Universitat Jaume I, Castellón, 2007, pag 763.
[154] PLAZA PENADÉS: Breve comentario a la Ley de Servicios de la Sociedad de la Información y Comercio Electrónico, en Revista de Derecho Informático, num 49, 2002.
[155] QUINTANA CARLO*: El contrato de transporte terrestre de mercancías*, en *Contratos Mercantiles*, Dir. A. Bercovitz, Pamplona, 2005, pág 931 y ss.
[156] ROJO, A: El empresario, en *Curso de Derecho Mercantil*, Dirs Rodrigo Uría/Aurelio Menéndez, Madrid, 2006, Tomo I, págs 70-71.
[157] SÁNCHEZ ANDRÉS, A.: El transporte combinado de mercancías. RDM, núm 135-136, 1975, pág 50 y ss.
[158] SANCHEZ CALERO: La comisión de transporte, en *Instituciones de Derecho Mercantil, I y II*, 28ª ed., Barcelona, 2005.
[159] SANJUÁN Y MUÑOZ, E., *Incorporación de las nuevas tecnologías en el comercio: aspectos legales*, Consejo General del Poder Judicial (Coeditada en colaboración con la Conselleria de Justicia y Administraciones Publicas de la Generalitat Valenciana), 1st ed., Madrid, 2009.
[160] TUR FAUNDEZ*: El contrato de viaje combinado: notas sobre la Ley 21/1995, de 6 de julio, de regulación de los viajes combinados*, Aranzadi Civil, 1996.
[161] VICENT CHULÍA.: *Introducción al Derecho Mercantil*. Tirant lo Blanch, Valencia, 2005. Pág. 888.
[162] ZERPA ALEMÁN, P.: La responsabilidad en el transporte multimodal, en Vector Plus num. 13, Ed. Fundación Universitaria de las Palmas, 1999, pág 68-73.
[163] Open Bip Home Page. http://sigma.ita.es/openBIP/
[164] OWL 2 Web Ontology Language New Features and Rationale.

Appendix II. Language Description of OWL.

This section provides an introduction to OWL. First paragraphs introduce We first introduce the OWL-Lite features and then we extend it to OWL-DL and OWL-Full.

A.II.1 OWL: Web Ontology Language.

OWL is an ontology language (or rather a family of three languages) developed by the World Wide Web Consortium (W3C) [2],[3],[5],[6][69]. Although initially developed in order to satisfy requirements deriving from Semantic Web research [13], OWL has rapidly become a de facto standard for ontology development in general and OWL ontologies are now under development and/or in use in areas as diverse as e-Science, medicine, biology, geography, astronomy, defense and the automotive and aerospace industries.

OWL[2],[3],[5],[6] is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called ontology. It is part of the growing stack of W3C recommendations related to the Semantic Web:

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a datamodel for objects ("resources") and relations between them, it provides a simple semantics for this datamodel and these datamodels can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry) and enumerated classes.

OWL[6] provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- **OWL Lite**[62] supports those users primarily needing a classification hierarchy and simple constraints. OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL.
- **OWL DL**[63] supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. OWL DL is so named due to its correspondence with *description logics*, a field of research that has studied the logics that form the formal foundation of OWL.

- **OWL Full** is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

The figure below [79] represents a simplified map of the complexity of a number of OWL dialects and related languages, as well as of their disposition towards DL and LP-based semantics. The languages that still lack standardization are given using rectangles with dashed borders. The dot-bordered rectangle, labeled OWLIM, presents the most complex OWL dialect supported by the OWLIM semantic repository.

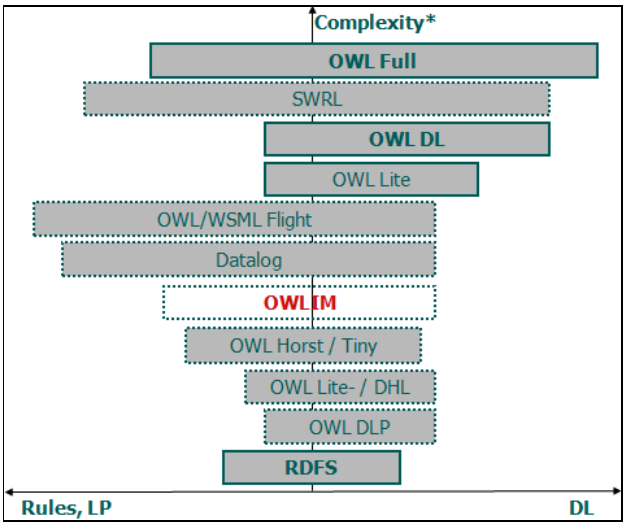


Fig. 34.- OWL Dialects and Related Languages.

An interested reader can review the main features of all OWL dialects on Appendix II.

A.II.2 OWL 2 Overview.

Now we briefly introduce the OWL 2. It was a Proposed Recommendation of the W3C which was published on September 22nd of 2009. This paragraph gives a short description of its main features. [6]

Figure 1 gives an overview of the OWL 2 language, showing its main building blocks and how they relate to each other. The ellipse in the center represents the abstract notion of an ontology, which can be thought of either as an abstract

structure or as an RDF graph. At the top are various concrete syntaxes that can be used to serialize and exchange ontologies. At the bottom are the two semantic specifications that define the meaning of OWL 2 ontologies.

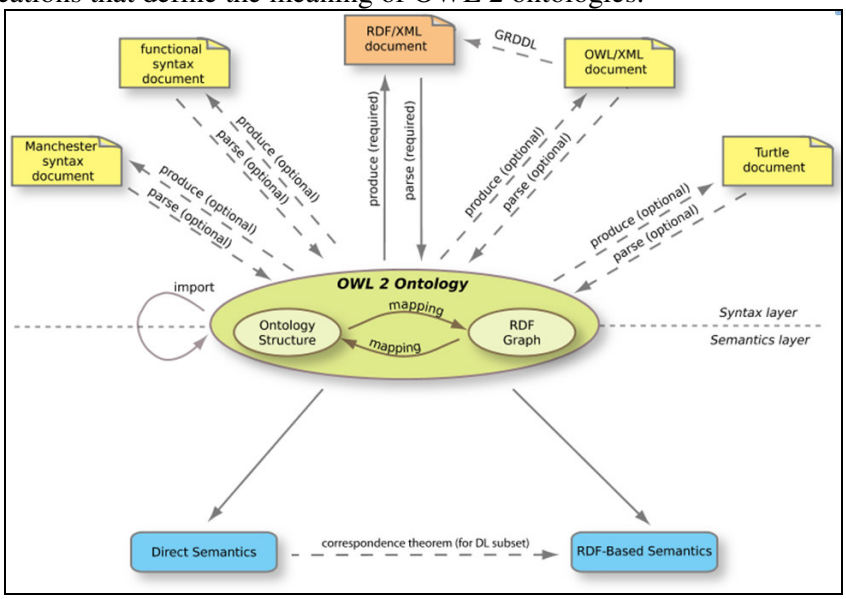


Fig. 35.- The Structure of OWL 2 .

Most users of OWL 2 will need only one syntax and one semantics layer; for them, this diagram would be much simpler, with only their one syntax at the top, their one semantics at the bottom and rarely a need to see what's inside the ellipse in the center. The primary OWL syntaxes are summarized in next table:

Syntax	Specification	Status	Purpose
RDF/XML	Mapping to RDF Graphs, RDF/XML	Mandatory	Interchange (can be written and read by all conformant OWL 2 software)
OWL/XML	XML Serialization	Optional	Easier to process using XML tools
Functional Syntax	Structural Specification	Optional	Easier to see the formal structure of ontologies
Manchester Syntax	Manchester Syntax	Optional	Easier to read/write DL Ontologies
Turtle	Mapping to RDF Graphs, Turtle	Optional, Not from OWL-WG	Easier to read/write RDF triples

Table 3.- Primary OWL Sintax specifications.

The OWL 2 Structural Specification document defines the abstract structure of OWL 2 ontologies (i.e its syntax), but it does not define their meaning. The **Direct Semantics** and the **RDF-Based Semantics** provide two alternative ways of assigning meaning to OWL 2 ontologies, with a correspondence theorem providing a link between the two. These two semantics are used by reasoners and other tools, e.g., to answer class consistency, subsumption and instance retrieval queries.

The **Direct Semantics** assigns meaning directly to ontology structures, resulting in a semantics compatible with the model theoretic semantics of the **SROIQ description logic**—a fragment of first order logic with useful computational

properties. The advantage of this close connection is that the extensive description logic literature and implementation experience can be directly exploited by OWL 2 tools. However, some conditions must be placed on ontology structures in order to ensure that they can be translated into a SROIQ knowledge base. Ontologies that satisfy these syntactic conditions are called **OWL 2 DL ontologies** (OWL 2 DL).

The **RDF-Based Semantics** assigns meaning directly to RDF graphs and so indirectly to ontology structures via the Mapping to RDF graphs. The RDF-Based Semantics is fully compatible with the **RDF Semantics** and extends the semantic conditions defined for RDF. The RDF-Based Semantics can be applied to any OWL 2 Ontology, without restrictions, as any OWL 2 Ontology can be mapped to RDF. "**OWL 2 Full**" is used informally to refer to RDF graphs considered as OWL 2 ontologies and interpreted using the RDF-Based Semantics.

The correspondence theorem of the RDF-Based Semantics Document defines a precise, close relationship between the Direct and RDF-Based Semantics. This theorem states, in essence, that given an OWL 2 DL ontology, inferences drawn using the Direct Semantics will still be valid if the ontology is mapped into an RDF graph and interpreted using the RDF-Based Semantics.

OWL 2 introduces the concept of **OWL 2 Profiles**. They are sub-languages (syntactic subsets) of OWL 2 that offer important advantages in particular application scenarios. Each profile is defined as a syntactic restriction of the OWL 2 Structural Specification. Each of the profiles trades off different aspects of OWL's expressive power in return for different computational and/or implementational benefits. Three different profiles are defined:

- **OWL 2 EL** enables polynomial time algorithms for the entire standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed and where expressive power can be traded for performance guarantees.
- **OWL 2 QL** enables conjunctive queries to be answered in LogSpace (more precisely, AC^0) using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL).
- **OWL 2 RL** enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

To conclude this brief introduction to OWL2 we summarize here its main differences in respect to OWL. OWL 2 has a very similar overall structure to OWL 1: all the building blocks of OWL 2 were present in OWL 1, albeit possibly under different names.

The central role of RDF/XML, the role of other syntaxes and the relationships between the Direct and RDF-Based semantics (i.e., the correspondence theorem) has not changed. More importantly, backward compatibility with OWL 1 is, to all intents and purposes, complete: all OWL 1 Ontologies remain valid OWL 2 Ontologies, with identical inferences in all practical.

OWL 2 adds new functionality with respect to OWL 1. Some of the new features are syntactic sugar (e.g., disjoint union of classes) while others offer new expressivity, including:

- keys;
- property chains;
- richer datatypes, data ranges;
- qualified cardinality restrictions;
- asymmetric, reflexive and disjoint properties; and
- enhanced annotation capabilities

As it was shown before, OWL 2 also defines three new profiles and a new syntax. In addition, some of the restrictions applicable to OWL DL have been relaxed; as a result, the set of RDF Graphs that can be handled by Description Logics reasoners is slightly larger in OWL 2.

All of the above is documented in detail in the OWL 2 New Features and Rationale document [164].

A.II.3. OWL-Lite Features.

OWL Lite RDF Schema Features	
Class	A class defines a group of individuals that belong together because they share some properties. Classes can be organized in a specialization hierarchy using subClassOf. There is a built-in most general class named **Thing** that is the class of all individuals and is a superclass of all OWL classes. There is also a built-in most specific class named **Nothing** that is the class that has no instances and a subclass of all OWL classes.
rdfs:subClassOf:	Class hierarchies may be created by making one or more statements that a class is a subclass of another class.
rdf:Property:	Properties can be used to state relationships between individuals or from individuals to data values. Both owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property.
rdfs:subPropertyOf:	Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties.
rdfs:domain:	A domain of a property limits the individuals to which the property can be applied. If a property relates an individual to another individual and the property has a class as one of its domains, then the individual must belong to the class. Note that rdfs:domain is called a global restriction since the restriction is stated on the property and not just on the property when it is associated with a particular class.
rdfs:range:	The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual and the property has a class as its range, then the other individual must belong to the range class. Range is also a global restriction as is domain above.
Individual:	Individuals are instances of classes and properties may be used to relate one individual to another.
OWL Lite Equality and Inequality	
Class	Equivalent classes have the same instances. Equality can be used to create

	synonymous classes.
equivalentProperty	Equivalent properties relate one individual to the same set of other individuals. Equality may be used to create synonymous properties.
sameAs	Two individuals may be stated to be the same. These constructs may be used to create a number of different names that refer to the same individual.
differentFrom	An individual may be stated to be different from other individuals. Explicitly stating that individuals are different can be important in when using languages such as OWL (and RDF) that do not assume that individuals have one and only one name.
AllDifferent	A number of individuals may be stated to be mutually distinct in one AllDifferent statement. The AllDifferent construct is particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects. It is used in conjunction with distinctMembers to state that all members of a list are distinct and pairwise disjoint.
OWL Lite Property Characteristics	
inverseOf:	If the property P1 is stated to be the inverse of the property P2, then if X is related to Y by the P2 property, then Y is related to X by the P1 property.
TransitiveProperty:	If a property is transitive, then if the pair (x,y) is an instance of the transitive property P and the pair (y,z) is an instance of P, then the pair (x,z) is also an instance of P. OWL Lite (and OWL DL) impose the side condition that transitive properties (and their superproperties) cannot have a maxCardinality 1 restriction. Without this side-condition, OWL Lite and OWL DL would become undecidable languages. See the property axiom section of the OWL Semantics and Abstract Syntax document for more information.
SymmetricProperty:	If a property is symmetric, then if the pair (x,y) is an instance of the symmetric property P, then the pair (y,x) is also an instance of P.
FunctionalProperty:	If a property is a FunctionalProperty, then it has no more than one value for each individual (it may have no values for an individual). This characteristic has been referred to as having a unique property. FunctionalProperty is shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1.
InverseFunctionalProperty:	Thus the inverse of the property has at most one value for each individual. This characteristic has also been referred to as an unambiguous property.
OWL Lite Property Restrictions	
allValuesFrom:	The restriction allValuesFrom is stated on a property with respect to a class. It means that this property on this particular class has a local range restriction associated with it. Thus if an instance of the class is related by the property to a second individual, then the second individual can be inferred to be an instance of the local range restriction class. Note that a reasoner cannot deduce from an allValuesFrom restriction alone that there actually is at least one value for the property.
someValuesFrom:	The restriction someValuesFrom is stated on a property with respect to a class. A particular class may have a restriction on a property that at least one value for that property is of a certain type. Unlike allValuesFrom, someValuesFrom does not restrict all the values of the property to be instances of the same class. Note that a reasoner cannot deduce (as it could with allValuesFrom restrictions) that all values of hasKeyword are instances of the SemanticWebTopic class.
OWL Lite Restricted Cardinality	
minCardinality:	Cardinality is stated on a property with respect to a particular class. If a *minCardinality* of 1 is stated on a property with respect to a class, then any instance of that class will be related to at least one individual by that property. This restriction is another way of saying that the property is required to have a value for all instances of the class. For example, the class Person would not have any minimum cardinality restrictions stated on a hasOffspring property since not all persons have offspring. A minimum cardinality of zero on a property just states (in the absence of any more specific information) that the property is optional with respect to a class.
maxCardinallity:	Cardinality is stated on a property with respect to a particular class. If a *maxCardinality* of 1 is stated on a property with respect to a class, then any instance of that class will be related to at most one individual by that property. A maxCardinality 1 restriction is sometimes called a functional or unique property. For example, the property hasRegisteredVotingState on the class. From a maximum cardinality one restriction alone, a reasoner can not deduce a minimum cardinality of 1. It may be useful to state that certain classes have no values for a particular property.
cardinality:	Cardinality is provided as a convenience when it is useful to state that a property on a class has both *minCardinality* 0 and *maxCardinality* 0 or both *minCardinality* 1 and *maxCardinality* 1.

OWL Lite Class Intersection	
intersectionOf:	OWL Lite allows intersections of named classes and restrictions.
OWL Lite Datatypes	
	OWL uses the RDF mechanisms for data values: xsd:string xsd:normalizedString xsd:boolean xsd:string xsd:decimal xsd:float xsd:double xsd:decimal xsd:integer xsd:nonNegativeInteger xsd:positiveInteger xsd:integer xsd:nonPositiveInteger xsd:negativeInteger xsd:nonPositiveInteger xsd:long xsd:int xsd:short xsd:long xsd:unsignedLong xsd:unsignedInt xsd:unsignedShort xsd:unsignedLong xsd:hexBinary xsd:base64Binary xsd:hexBinary xsd:dateTime xsd:time xsd:date xsd:dateTime xsd:gYear xsd:gMonthDay xsd:gDay xsd:gYear xsd:anyURI xsd:token xsd:language xsd:anyURI xsd:NMTOKEN xsd:Name xsd:NCName xsd:NMTOKEN

Table 4.- OWL-Lite Features.

A.II.4. OWL-DL and OWL Full Features.

Incremental Language Description of OWL DL and OWL Full	
oneOf:	Enumerated classes. Classes can be described by enumeration of the individuals that make up the class. The members of the class are exactly the set of enumerated individuals; no more, no less.
hasValue:	A property can be required to have a certain individual as a value (also sometimes referred to as property values).
disjointWith:	Classes may be stated to be disjoint from each other. From this disjointWith statement, a reasoner can deduce an inconsistency when an individual is stated to be an instance of both and similarly a reasoner can deduce that if A is an instance of Man, then A is *not* an instance of Woman.
unionOf, complementOf, intersectionOf	Boolean combinations: OWL DL and OWL Full allow arbitrary Boolean combinations of classes and restrictions: unionOf, complementOf and intersectionOf.
minCardinality, maxCardinality, cardinality	Full cardinality. While in OWL Lite, cardinalities are restricted to at least, at most or exactly 1 or 0, full OWL allows cardinality statements for arbitrary non-negative integers.
complex classes	In many constructs, OWL Lite restricts the syntax to single class names (e.g. in subClassOf or equivalentClass statements). OWL Full extends this restriction to allow arbitrarily complex class descriptions, consisting of enumerated classes, property restrictions and Boolean combinations. Also, OWL Full allows classes to be used as instances (and OWL DL and OWL Lite do not).

Table 5.- OWL-DL & OWL Full Features.

Appendix III. OWL eContract OWL Ontology v2.0.

As it was indicated on point 3.3.3 of the D6.12 we give here a more detailed description of the Ontology we designed and used to build the semantic contract search prototype.

A.III.1 Ontology v2.0 Complete View.

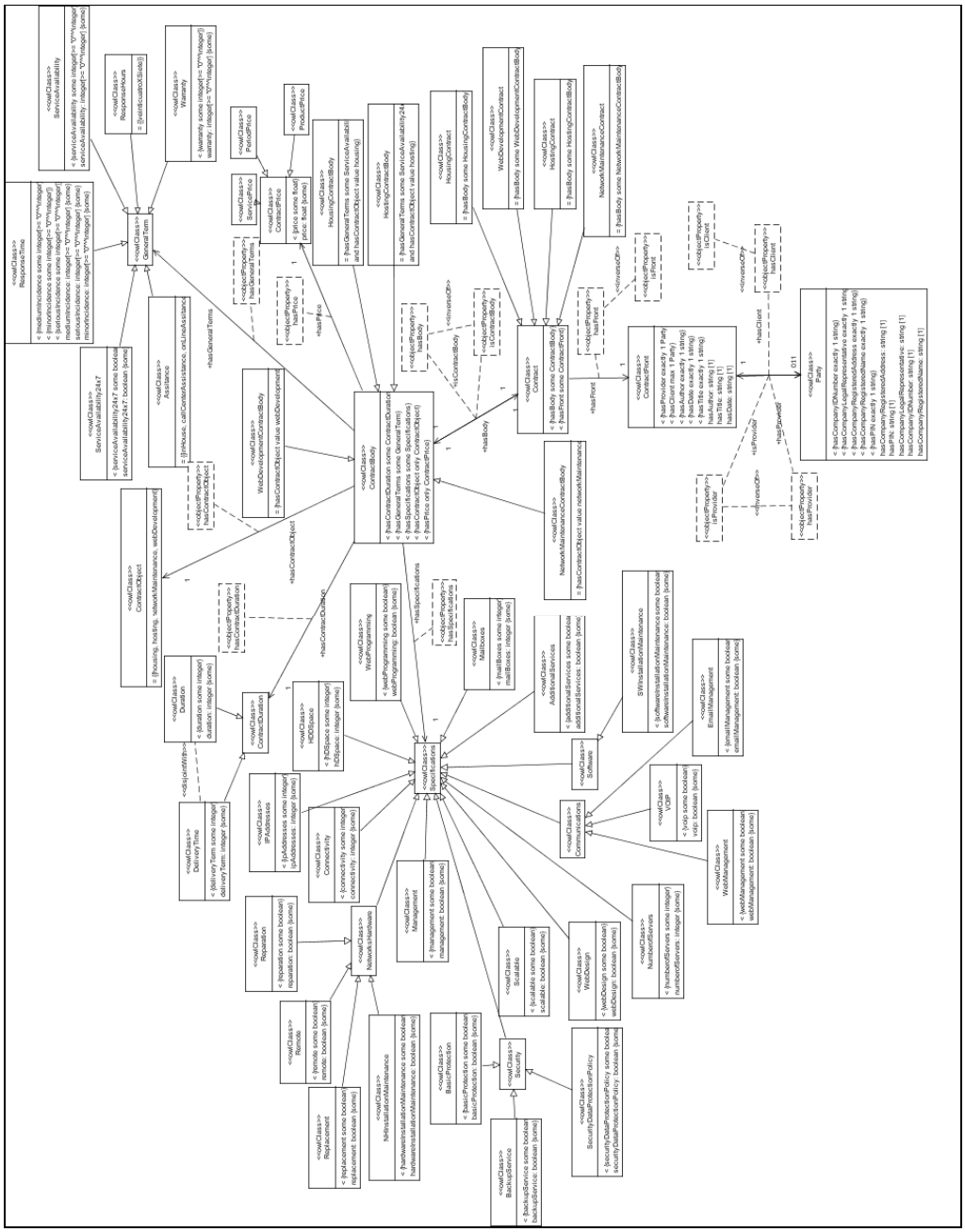


Fig. 36.- UML representation of the eContract OWL Ontology v2.0.

A.III.2 Ontology v2.0 Contract and Contract Body Detail

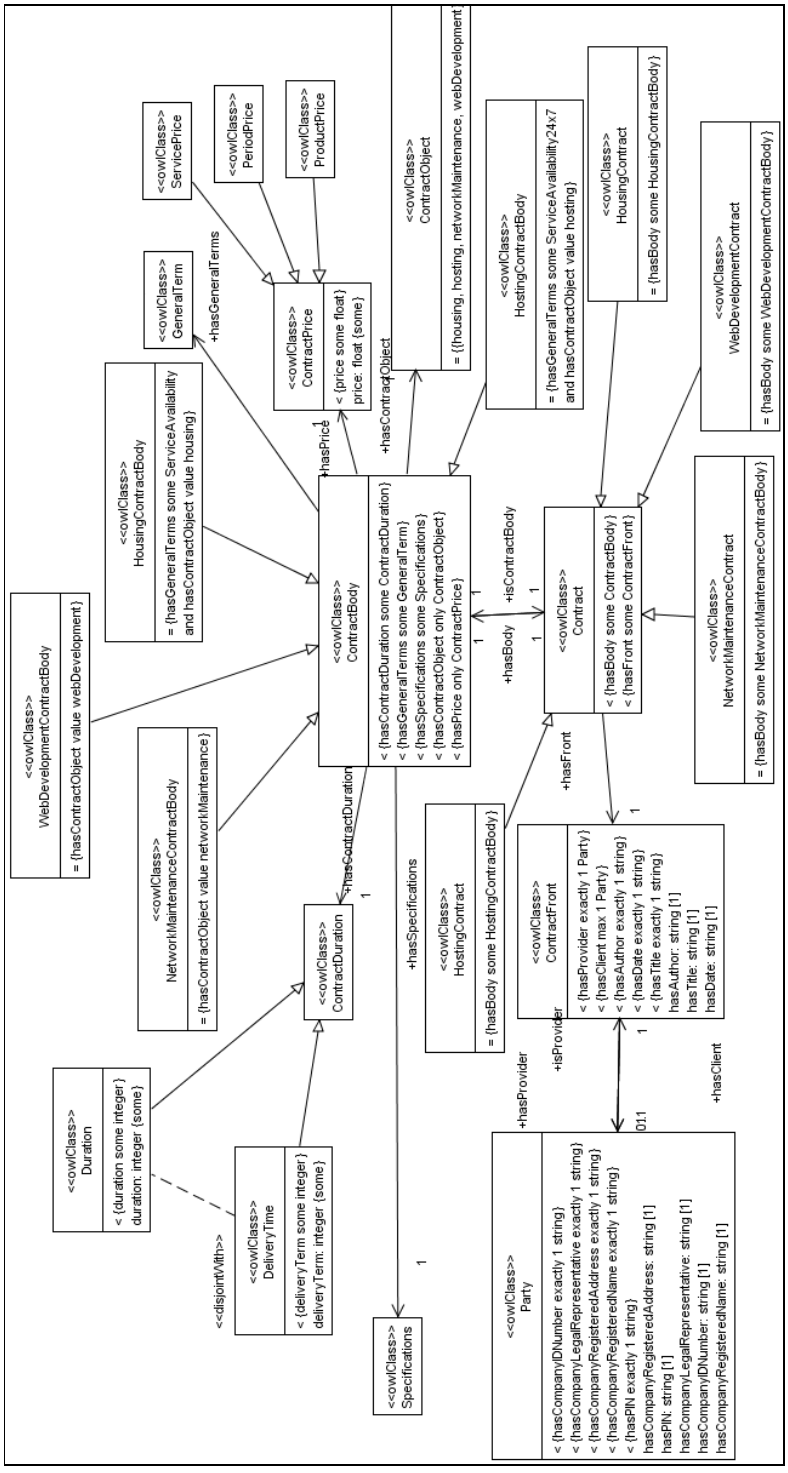


Fig. 37.- Contract and Contract Body Detail.

A.III.3 Ontology v2.0 Specifications Detail

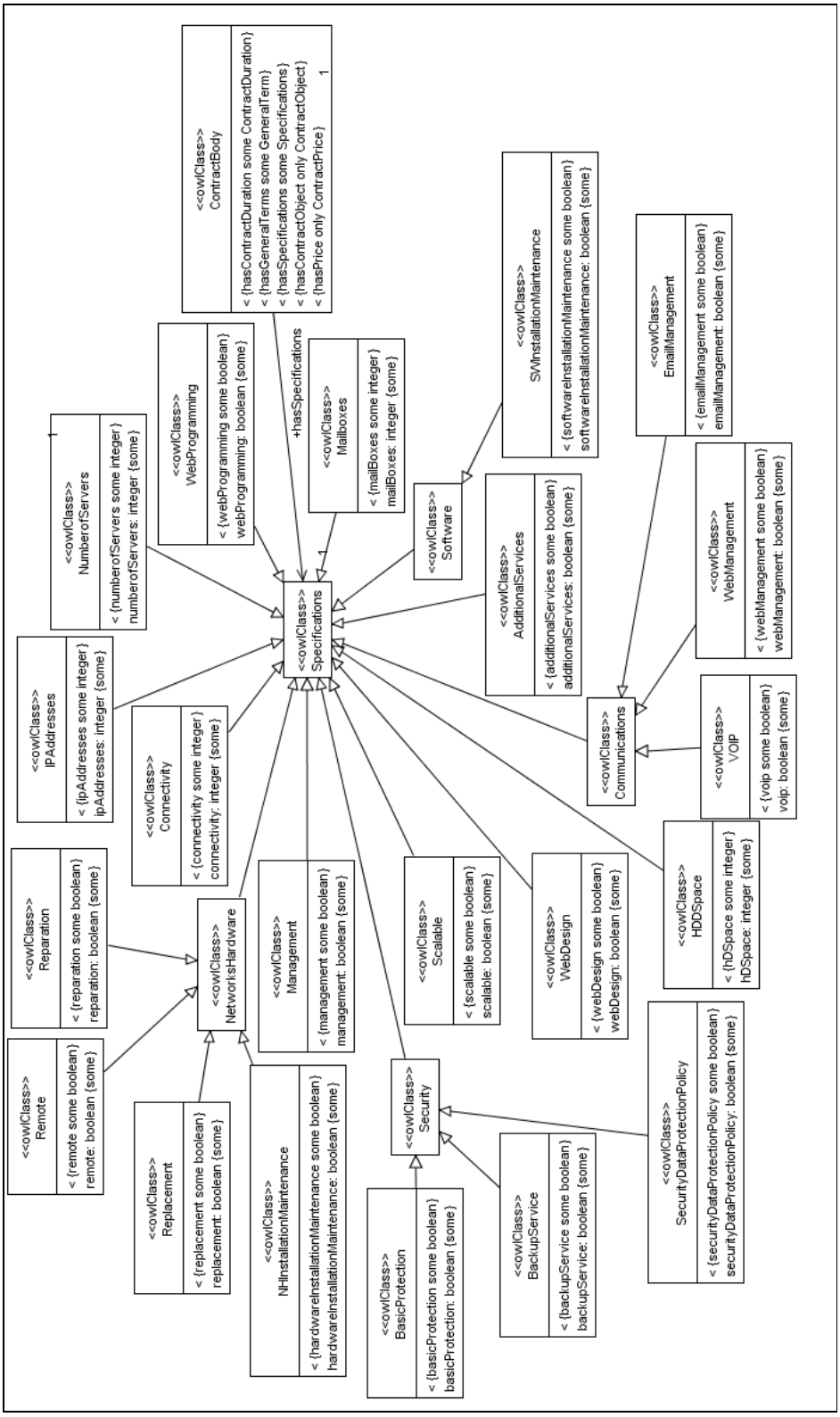


Fig. 38.- eContract Ontology Specifications Detail.

A.III.4 Ontology v2.0 General Terms Detail.

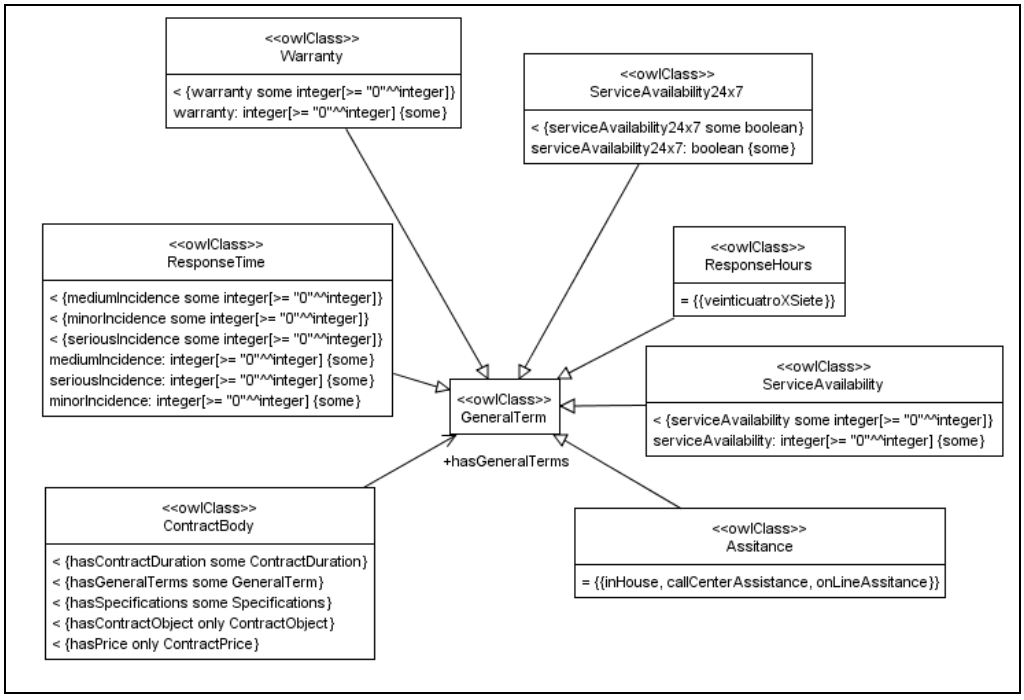


Fig. 39.- eContract Ontology General Terms Detail.

A.III.5. OWL Definition.

This version is compatible with Protégé 4.1.0 Alpha, OWLAPI 3.0 and the Pellet reasoner 2.1.1. If some reader wants to use it with Protégé **4.0.x** and OWLAPI 2.0, the only thing to change is " IRI=" to " URI=" a little problem it took us to solve about half a day.

A.III.6. XML Synset Definition.

The table exemplified on next page includes some synonyms definition:

- The file is a set of syssentries which are between the <synsets> </synsets> labels. Each synsets is mapped to one ontology by the onto attribute value in our case to the *eContract_Traduccion.owl*.
- Each synsets is composed of at least one *synset*. Each of them is mapped to an ontology concept. That is made with the value of the "concept" attribute: "*concept=*http://www.ita.es/ontologies/Opaals/eContract_Traduccion#Contract_Price" is an example of.
- Each synset has a *synsetentry* per language which is going to be "interpreted". The language is defined by the *lang* attribute, for example <*SYNSETENTRY lang="es"*>.

- At least one *MWORD* must be established for each of the *synsetentry*. In languages like Spanish or French some vocals should have to be printed with an accent over them. It is not necessary to use it, or to write the word in capitals and/or smalls as the items are "stemed" before the word is used to discover the associated synonym, that is, all them are normalized.
  In the case of have the necessity to weight the connection of a word to a concept we can use the attribute *weight* of the *MWORD* label to represent it.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<synsets onto="eContract_Traduccion.owl">
        <SYNSET concept="http://www.ita.es/ontologies/Opaals/eContract_Traduccion#Contract_Price">
      <SYNSETENTRY lang="es">
       <MWORD weight="1.0">precio del contrato</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="en">
       <MWORD weight="1.0">contract_price</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="fr">
       <MWORD weight="1.0">TIN</MWORD>
      </SYNSETENTRY>
        </SYNSET>
        <SYNSET concept="http://www.ita.es/ontologies/Opaals/eContract_Traduccion#General_Terms">
      <SYNSETENTRY lang="es">
       <MWORD weight="1.0">terminos generales</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="en">
       <MWORD weight="1.0">general_terms</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="fr">
       <MWORD weight="1.0">TIN</MWORD>
      </SYNSETENTRY>
        </SYNSET>
        <SYNSET concept="http://www.ita.es/ontologies/Opaals/eContract_Traduccion#CompanySignature">
      <SYNSETENTRY lang="es">
       <MWORD weight="1.0">firma_compañia</MWORD>
       <MWORD weight="1.0">Firma Empresa</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="en">
       <MWORD weight="1.0">companySignature</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="fr">
       <MWORD weight="1.0">TIN</MWORD>
      </SYNSETENTRY>
        </SYNSET>
        <SYNSET concept="http://www.ita.es/ontologies/Opaals/eContract_Traduccion#ClientSignature">
      <SYNSETENTRY lang="es">
       <MWORD weight="1.0">firma_cliente</MWORD>
       <MWORD weight="1.0">FirmaCliente</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="en">
       <MWORD weight="1.0">clientSignature</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="fr">
       <MWORD weight="1.0">TIN</MWORD>
      </SYNSETENTRY>
        </SYNSET>
        <SYNSET concept="http://www.ita.es/ontologies/Opaals/eContract_Traduccion#cif">
      <SYNSETENTRY lang="es">
       <MWORD weight="1.0">Código de identificación fiscal</MWORD>
                        <MWORD weight="1.0">CIF</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="en">
       <MWORD weight="1.0">Tax Identification Number</MWORD>
                        <MWORD weight="1.0">TIN</MWORD>
      </SYNSETENTRY>
      <SYNSETENTRY lang="fr">
       <MWORD weight="1.0">TIN</MWORD>
```

```
        </SYNSETENTRY>
            </SYNSET>
</synsets>
```

Fig. 40.- Example of Synsets File.

The interested reader can find the full XML Synset File and the translating ontology at: http://opaalstools.sourceforge.net.

A.IV. Tools for Designing and Using Ontologies.

As we already mentioned in 3.1 the **standardization** of OWL[65] has led to the development and adaption of a wide range of tools and services, including reasoners such as FaCT++ [53], Racer-Pro [49], Pellet [44][56] and KAON2[37]. Reasoners are often used with editing tools, e.g., Protégé [42] and Swoop [164], in order to compute the class hierarchy and alert users to problems such as inconsistent classes.
One of the key benefits that a formal language standard provides to users of web-based technologies is *interoperability*. In the case of OWL, this should mean that users can load ontologies from the internet and use them in applications, possibly answering queries against them using one of the available OWL reasoners. One of the goals of **standardization is that the result of this process is independent of the chosen reasoner**.
The following paragraphs briefly introduce tools which are related with the creation and use (managing and querying) of OWL based ontologies. A good starting point to obtain information about ontologies is [89].
An important issue in order to achieve standardization and interoperability with different reasoning systems is the definition of standardized APIs to access to its functionalities. We introduce three attempts (DIS, Jena and OWL-API) to do it. The implementation of any of these interfaces by a reasoner would be really evaluated in our chose as it will allow future changes without affection in other improvements of the system.
Next we introduce a couple of tools for Ontology development and test (Neon Toolkit and Protégé). They are modularized and they can use different reasoners as they implement the client side of different APIs.
After that we proceed to introduce the features of the more popular reasoners in order to be able to evaluate, compare and chose the one who best fit the proposal of our development but considering the independence we obtain of the API implementation. In order to get an evaluation of the reasoners we show an Automatic Framework for comparing DL Reasoners [65] and a Benchmarking of OWL Reasoners [68] together with and we chose one of the introduced reasoners and justify the selection. Then we evaluate the reasoners and chose the one we use within the project. We contrast our evaluation with the one provided by [88].
As a final point we mention HERAKLES a reasoner brokering initiative which seems to be really interesting for incoming research.

A.IV.1. The DIG 2.0 Interface.

The DIG Interface [46] provides an implementation-neutral mechanism for accessing Description Logic reasoner functionality. At a high level the interface consists of XML messages sent to the reasoner over HTTP connections, with the reasoner responding as appropriate.
DIG provides a basic API to a DL system. It is intended to be a lightweight mechanism providing access to reasoning functionality. There are many things that a reasoning service may be expected to provide that the DIG interface does not provide — it is expected that a DIG reasoner will be one component within a larger architecture. That is, DIG does not intend to provide what we might truly call a

reasoning service, but rather helps to insulate applications from the location and implementation language of a DL reasoner.
DIG uses HTTP as its underlying protocol for communicating with a reasoner. The reasoner accepts HTTP POST requests and responds as appropriate. The contents of the requests are elements defined by an XML schema.
The core DIG specification is an XML Schema for a DL concept language, ask/tell functionality and a description of a protocol used to communicate these operations (using HTTP). Along with the definition of the interface, however, there is also a commitment from implementers of leading DL reasoners to provide implementations conforming to the specification.
The DIG interface makes a number of assumptions.

- The specification is agnostic as to multiple client connections. Multi-threaded implementations of a reasoner may be provided, but no guarantees are made as to the semantics when clients attempt to simultaneously update and query.
- The connection to the reasoner is effectively stateless. Clients are not identified to the reasoner, thus the reasoner will not distinguish between clients and maintain any kind of consistency checking or record of which client is adding information or making requests. Conversely, a client has no way of ensuring that the reasoner has not been given additional information (such as additional axioms) since its last communication.
- There is no explicit classification request. The reasoner will decide when, for example, it is appropriate to build a classification hierarchy of concepts. This may happen after each TELL request, alternatively the reasoner may choose to defer the classification until absolutely necessary, or even when there is a lull in traffic.

The specification is not intended as a”database system" for knowledge bases. It is simply a protocol that exposes the reasoning services provided by a DL reasoner. A DIG reasoner may be used to implement a service that provides concurrent access, transactions etc, such functionality is not inherently supported by DIG.
The advantage of DIG is that applications can communicate with any DIG compliant reasoner, without needing to know specific reasoner details or reasoner interaction protocols. This means that it is possible to "plug in" any DIG compliant reasoner into any DIG aware application. For example, RACER, FaCT++ or any other DIG compliant reasoner may be used with Protégé-OWL.

A.IV.2 The Jena/Jena2 Framework

Jena is a leading Semantic Web programmers’ toolkit [84][86]. It is an open-source project, implemented in Java and available for download from SourceForge. It grown out of work with the HP Labs Semantic Web Programme[85] distributed under the liberal BSD-style license. HPLabs management have recently decided not to continue with an active programme of Semantic Web research at HPL

The Jena Framework includes:

- A RDF API
- Reading and writing RDF in RDF/XML, N3 and N-Triples
- An OWL API

- In-memory and persistent storage
- SPARQL query engine

Jena offers a simple abstraction of the RDF graph as its central internal interface. This is used uniformly for graph implementations, including in-memory, database-backed and inferred graphs. The main contribution of Jena is a rich API for manipulating RDF graphs. Around this, Jena provides various tools, e.g., an RDF/XML parser, a query language, I/O modules for N3, N-triple and RDF/XML output. Underneath the API the user can choose to store RDF graphs in memory or in databases. Jena provides additional functionality to support RDFS and OWL. The two key architectural goals of Jena2 are:

- Multiple, flexible presentations of RDF graphs to the application programmer. This allows easy access to and manipulation of, data in graphs enabling the application programmer to navigate the triple structure.
- A simple minimalist view of the RDF graph to the system programmer wishing to expose data as triples.

The first is layered on top of the second, so that essentially any triple source can back any presentation API. Triple sources may be materialized, for example database or in-memory triple stores, or virtual, for example resulting from inference processes applied to other triple sources.

An simplified overview of the Jena 2 architecture is shown in next Figure. Applications typically interact with an abstract Model which translates higher-level operations into low-level operations on triples stored in an RDF Graph.

At an abstract level, the Jena2 storage subsystem need only implement three operations: add statement, to store an RDF statement in a database; delete statement, to remove an RDF statement from the database; and the find operation. The find operation retrieves all statements that match a pattern of the form <S,P,O> where each S, P, O is either a constant or a don't-care. Jena's query language, RDQL, is converted to a set of find operations in which variables are permitted in the find patterns. The variables enable joins across the patterns.

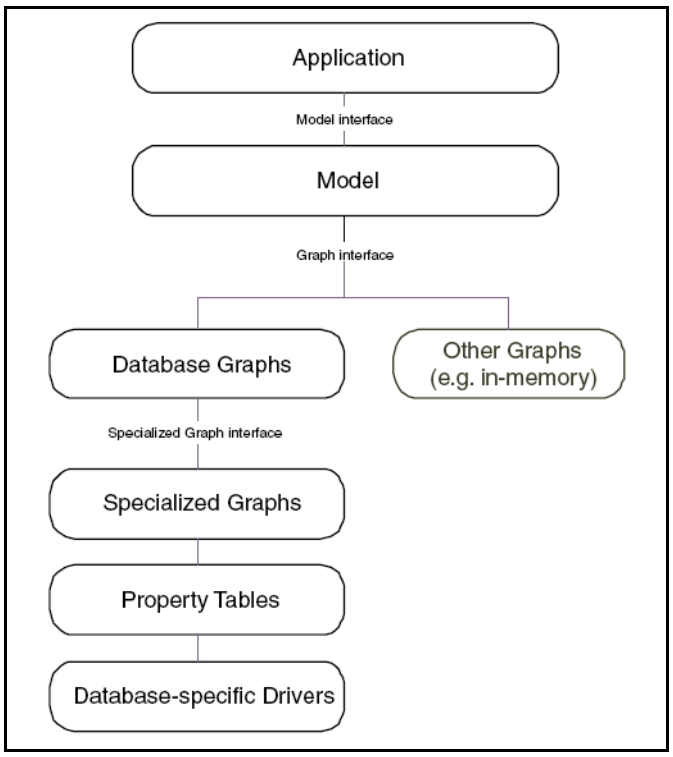


Fig. 41.- Jena2 Architectural Overview.

A.IV.3. The OWL API.

The OWL API [54][82] is a high level API for working with OWL 2 ontologies. Aside from the fact the OWL is a W3C Recommendation [2][3], one of the probable reasons for its success and relatively broad adoption in academia and industry is that there have been a wide range of OWL tools available. These tools have supported the creation and editing of OWL ontologies, reasoning over ontologies and using ontologies in applications. Generally speaking, all of these tools require some kind of underlying API that allows ontologies to be loaded, manipulated and queried. In the case of ontology editors, such as Protégé [42], the NeOn Toolkit [36], **Swoop**, **TopBraid Composer1** and OWLSight2, it is usually a necessity to have some mechanism to load ontologies in various concrete formats, along with clean mechanisms to manipulate and modify ontologies. With regard to reasoning, client applications usually require general purpose reasoner interfaces so that they can easily swap in and out different reasoner implementations. On the other side of the coin, reasoner developers can benefit from being able to provide a well known interface to their reasoners.
The API is closely aligned with the OWL 2 structural specification. It supports parsing and rendering in the syntaxes defined in the W3C specification, namely, the Functional Syntax, RDF/XML, OWL/XML and the Manchester OWL Syntax. Finally, the reference implementation of the API, which is written in Java, includes validators for the various OWL 2 profiles - OWL 2 QL, OWL 2 EL and OWL 2 RL.

The features (interfaced) provided by the API are:

- **Ontology Management**. The *OWLOntology* interface provides a point for efficiently accessing the axioms contained in an ontology. The *OWLManager* provides a central point for creating, loading, changing and saving ontologies, which are instances of the OWLOntology interface. Each instance of an ontology is created or loaded by an ontology manager. Each instance of an ontology is unique to a particular manager and all changes to an ontology are applied via its manager. This centralized management desing allows client applications to have one access point to ontologies, to provide redirection mechanisms and other customizations for loading ontologies and allows client applications to monitor all changes that are applied to any loaded ontologies.

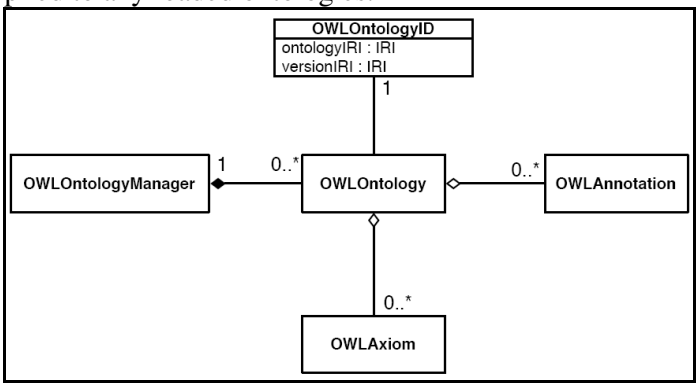


Fig. 42.- A UML diagram showing the management of ontologies in the OWL API.

- **Ontology Change**. Changes within an ontology include adding and removing axioms, setting the id of an ontology, adding and removing annotations and adding and removing imports. They are performed by objects which implements the *OWLOntologyChange*. In essence, change support is implemented using the *Command* design pattern which makes it easy to record and serialize ontology changes, implement undo and redo functionality and support transactional behavior. The changer are applied by the ontology manager.
- **Parsing and Rendering OWL Ontologies**. A major benefit of aligning the OWL API with the OWL 2 structural specification is that there is no commitment to a particular concrete syntax. The OWL API therefore includes out of the box support for reading and writing ontologies in several syntaxes, including RDF/XML, Turtle, OWL/XML, OWL Functional Syntax, The Manchester OWL Syntax, KRSS Syntax and the OBO flat file format. Due to the underlying design of the API, it is possible for the imports closure of an ontology to contain ontologies that were parsed from ontology documents written in different syntaxes. The reference implementation of the OWL API uses a registry of parsers and renderers, which makes it easy to add support for custom syntaxes. The appropriate parser is automatically selected at runtime when an ontology is loaded. By default, ontologies are saved back into the format from which they were parsed, but it is possible to override this in order to perform syntax conversion tasks and "save as" operations in editors for example.

- **Data Structure Storage**. The reference implementation provides data structures for efficient in-memory representations of ontologies. For many purposes this is sufficient. However, the API has been designed so that it is possible to provide other storage mechanism for ontologies. For example, there are use cases for storing ontologies in relational databases or triple stores. The OWL API has been designed with these use cases in mind and it is possible to "mix and match" storage implementations, so that an ontology imports closure could contain in-memory representations of ontologies, ontologies persisted in secondary storage in the form of a custom database and ontologies stored in triple stores.
- **Reasoner Interfaces.** The OWL API has various interfaces to support interacting with OWL reasoners. The main interface is the *OWLReasoner* interface. The reasoner interfaces have been designed so as to make it easier for reasoners to expose functionality that provides incremental reasoning support. The API allows a reasoner to be set up so that it listens for ontology changes and either immediately processes the changes or queues them in a buffer which can later be processed. At the time of this writing, the CEL [1], FaCT++ [17], HermiT [14], Pellet [16] and Racer Pro [5] reasoners provide OWL API wrappers. This means that they are also available as reasoner plugins to Protégé-4.

Comparing OWL API other APIs such as Jena [3], or the Protégé 3.X API, the representation of class expressions and axioms is not at the level of RDF triples. Indeed, the design of the OWL API is directly based on the OWL 2 Structural Specification [83]. This means that an ontology is simply viewed as a set of axioms and annotations as depicted in previous figure. The names and hierarchies of interfaces for entities, class expressions and axioms in the OWL API correspond closely to the structural specification. In fact, there is almost a one-to-one translation between the OWL 2 Structural Specification and core OWL API model interfaces, meaning that it is easy to relate the high level OWL 2 specification directly to the design of the API.

A.IV.4. NeOn Toolkit.

The NeOn Toolkit (NOT) is one of the results of the NeOn [36] project, an FP6 project which finished during 2010. The main goal of the NeOn project is to provide methodological and tool support for developing and managing a new generation of semantic applications. The NeOn Toolkit is a state-of-the-art, open source multi-platform ontology engineering environment, which provides comprehensive support for the ontology engineering life-cycle. The toolkit is based on the Eclipse platform and provides an extensive set of plug-ins covering a variety of ontology engineering activities, including Annotation and Documentation, Development, Human-Ontology Interaction, Knowledge Acquisition, Management, Modularization and Customization, Ontology Dynamics, Ontology Evaluation, Ontology Matching, Reasoning and Inference and Reuse.

The key outcomes from NeOn project are:

- Open, scalable and service-centered reference architecture for supporting the lifecycle of semantic applications.

- **The NeOn Toolkit** – a resource for engineering contextualized networked ontologies and semantic applications. It is a reference implementation of the NeOn Architecture. It supports ontology engineering and management, that is, its functionalities cover the complete ontology lifecycle. The ontologies it understands can be developed in different languages (OWL, F-Logic) and it has support for networked ontologies (modules, mappings).
- Industry-strength documentation and reference material.
- Three case studies in two sectors: pharmaceuticals and agriculture/fisheries.

Our interest in the project is to evaluate the capabilities the NOT has for Ontologies lifecycle management and which is more important for us which are the features of reasoning and inferring the available plug-ins have.

The NOT is distributed under two different configurations which differs in the kind of license agreement the user have to commit on. The Commercial Version of NOT is the Extended configuration. It contains a set of sophisticated plugins for developing F-logic ontologies with rules, for integrating external data sources, for mapping between ontologies and much more. It can be used for free by academicals.

As the name suggests the former version extends the Basic configuration which is Open Source and completely free. The included features in the Basic configuration can be increased by adding proprietary plugins using temporary license of for free if they are used under academic activities: that is from the Basic configuration and user is able to obtain the Extended configuration by adding new plugins.

This flexibility is based in the NOT architecture which is Open reference architecture for: Ontology engineering environments and Building semantic applications. The architecture is modular and in consequence the NOT platform is extensible: "Everything is a plugin". To get it, it is based on open standards: existing standards are used and the APIs are open and it is Service oriented and has support for using distributed components. To get all these features NOT take advantage of the Eclipse capabilities: it is used as base platform for engineering plugins, the OSGI runtime kernel as base for "non-GUI components", the OSGI runtime + web service infrastructure bundles for Ws-based plugin deployment, the SWT and JFace for GUI components, the Modeling Framework for implementation of NeOn Metamodel and Editor support and Finally Eclipse Communication Framework as base for collaboration functionalities.

One of the advantages of the NOT Architecture definition and its reference implementation, as it is already mentioned in previous paragraphs, is that the complete Ontology lifecycle can be managed thought the Neon Toolkit API. It follows the KAON2 API definition. In consequence it supports OWL and SWRL and its main features are listed below the picture.

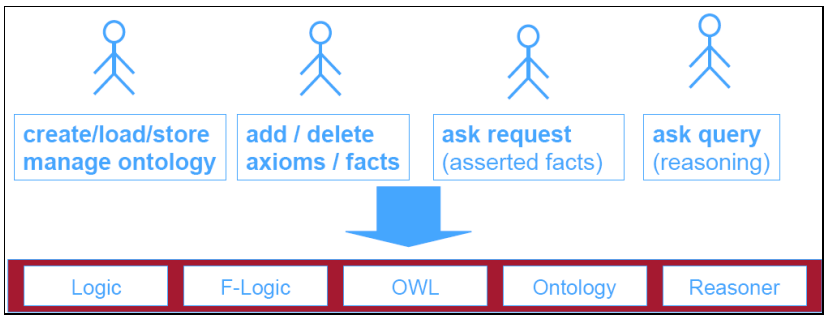


Fig. 43.- Neon Toolkit API.

- It has an OWL parser and serializer for OWL RDF, OWL XML and Abstract Syntax.
- It provides interfaces for manipulation of OWL-DL, SWRL and F-Logic ontologies.
- It is a stand-alone server providing access to ontologies in a distributed manner using RMI: An ontology may be local (to be implemented in a file) or remote placed. It is identified by Logical URIS while the physical URI identified their location and it provides a Resolver which manages the mapping between Logical and Physical URIS.
- It gives the opportunity of retrieve data from the Ontology in two different ways. The faster one is the Request method. This method provides purely syntactic access to the data model without reasoning. The second way is using the Query feature. It is performed by the Reasoner which compiles and evaluates the user queries and uses them to answer him over an ontology. The Query represents a conjunctive DL-safe query over an Ontology that can be constructed manually or via SPARQL.
- It provides efficient access to instances via relational databases.

A.IV.5. Protégé.

Protégé is a free, open-source platform that provides a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application API for building knowledge-based tools and applications. It allows ontologies to be exported into a variety of formats including RDF(S), OWL and XML Schema.

The Protégé platform supports two main ways of modeling ontologies. The **Protégé-Frames** editor provides a full-fledged user interface and knowledge server to support users in constructing and storing frame-based domain ontologies, customizing data entry forms and entering instance data. Protégé-Frames implements a knowledge model which is compatible with the Open Knowledge Base Connectivity protocol (OKBC) [41]. In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and

relationships and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

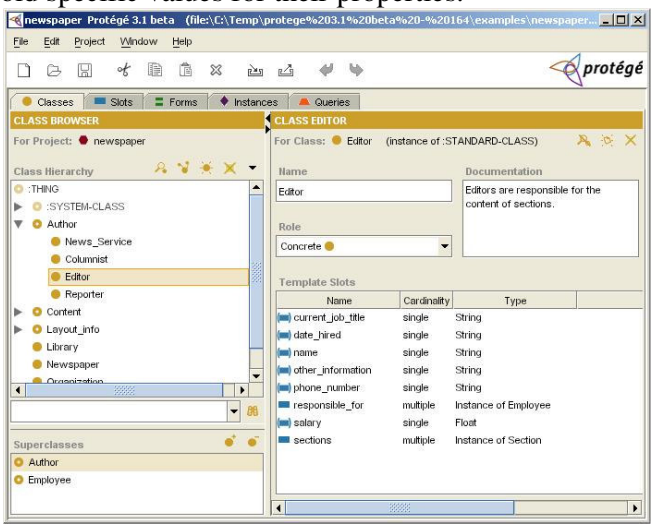


Fig. 44.- Protége-Frames Screenshot.

Features of Protégé-Frames include:

- A wide set of user interface elements that can be customized to enable users to model knowledge and enter data in domain-friendly forms.
- A plug-in architecture that can be extended with custom-designed elements, such as graphical components (e.g., graphs and tables), media (e.g., sound, images and video), various storage formats (e.g., RDF, XML, HTML and database back-ends) and additional support tools (e.g., for ontology management, ontology visualization, inference and reasoning, etc.).
- A Java-based API that makes it possible for plug-ins and other applications to access, use and display ontologies created with Protégé-Frames.

The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the Open Knowledge Base Connectivity protocol (OKBC). In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated to classes to describe their properties and relationships and a set of instances of those classes - individual exemplars of the concepts that hold specific values for their properties.

The **Protégé-OWL** editor is an extension of Protégé that supports the Web Ontology Language (OWL). This editor enables users to:

- Load and save OWL and RDF ontologies.
- Edit and visualize classes, properties and SWRL rules.
- Define logical class characteristics as OWL expressions.
- Execute reasoners such as description logic classifiers.
- Edit OWL individuals for Semantic Web markup.

Protégé-OWL's flexible architecture makes it easy to configure and extend the tool. It provides the Protégé Programming Development Kit (PDK). It is a set of documentation and examples that describes and illustrates how to develop and

install plug-in extensions for Protégé and how to work directly with the Protégé APIs. Plug-ins can be used to change and extend the behavior of Protégé. Protégé itself is written as a collection of plug-ins and these can be replaced singly or as a whole to completely alter the interface and behavior of Protégé. Protégé has two core Java APIs. The first one is used to access basic Protégé functionalities and frame-based knowledge bases (such as those created with Protégé-Frames). Protégé also has an OWL API [54] that extends the core API to provide access to OWL ontologies (such as those created with Protégé-OWL). The Protégé API can be used directly by external applications to access Protégé knowledge bases and make use of Protégé forms without running the Protégé application. Both APIs can be used for the development of custom-tailored user interface components or arbitrary Semantic Web services.

Although there are several different kinds of APIs (according with theirs functionalities) our interest is on the possibilities it offers to infere things. Protégé provides the **Protege-OWL Reasoner API** [43] which allows programmatic access to a direct or a DIG-compliant reasoner (see 2.3.5.1 for more details).

It provides methods for consistency checking, classification, etc. of an ontology as well as methods for getting the inferred information for a particular OWL entity.

The reasoning API is encapsulated in the `edu.stanford.smi.protegex.owl.inference` package. The main classes that will be used are the ReasonerManager (used to obtain a reasoner) and ProtegeReasoner (an interface to the direct or DIG reasoner). **The Reasoner Manager** is used to obtain inferred information about a OWL model such as inferred superclasses, inferred equivalent classes and inferred types for individuals. The **ProtegeReasoner** manages communication with the direct or DIG reasoner, ensuring that it is always properly synchronized with the internal Protégé-OWL model. The Protege full installation comes with three reasoner implementations:

- **DefaultProtegeDIGReasoner**: provides a connection to an external DIG compliant reasoned.
- **ProtegePelletJenaReasoner**: provides a direct connection to the Pellet reasoner accessed through the Jena API.
- **ProtegePelletOWLAPIReasoner**: provides a direct connection to the Pellet reasoner accessed through the OWL-API.

Each of the above reasoner implement the ProtegeReasoner interface. Protégé take advantage of supporting OWL-DL. OWL-DL has its foundations in Description Logics, which are decidable fragments of First Order Logic. For a particular task, a logic is decidable if it is possible to design an algorithm that will terminate in a finite number of steps (i.e., the algorithm is guaranteed not to run forever). For example, in Description Logic it is possible to write an algorithm that calculates whether or not one concept is a subclass of another concept, which is guaranteed to terminate after a finite number of steps. Because an OWL-DL ontology can be translated into a Description Logic representation, it is possible to perform automated reasoning over the ontology using a Description Logic reasoner. A Description Logic reasoner performs various inferencing services, such as computing the inferred superclasses of a class, determining whether or not a class is consistent (a class is inconsistent if it cannot possibly have any instances), deciding whether or not one class is subsumed by another, etc.

A.IV.6. Some other Features of KAON2.

KAON2 [37] is an infrastructure for managing OWL-DL, SWRL and F-Logic ontologies. It was developed by the following institutions: Information Process Engineering (IPE) at the Research Center for Information Technologies (FZI); Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe; Information Management Group (IMG) at the University of Manchester. Although in previous paragraphs, we introduce KAON2 within the NeOn Toolkit framework we introduce here a more low level description of its reasoning features which will allow us to compare this API with the provided by Protégé.

It is a successor to the KAON project. The main difference to KAON is the supported ontology language: KAON used a proprietary extension of RDFS, whereas KAON2 is based on OWL-DL and F-Logic. The main features of KAON2 are listed in the previous point but we must also be considered that it provides a DIG interface, allowing access from tools such as Protégé.

Reasoning with KAON2 is based on special-purpose algorithms which have been designed for dealing with large A-Boxes. The underlying rationale of the algorithms is that algorithms for deductive databases have proven to be efficient in dealing with large numbers of facts [70]. The KAON2 approach utilizes this by transforming OWL DL ontologies to disjunctive datalog and by the subsequent application of the mentioned and established algorithms for dealing with disjunctive datalog.

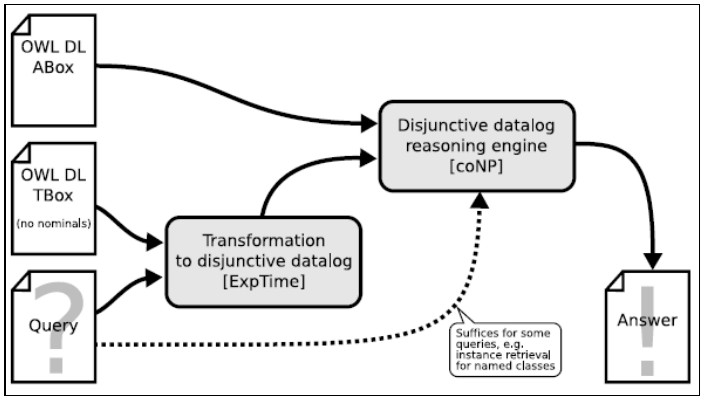


Fig. 45.- KAON2 Approach to Reasoning.

For reasoning, KAON2 supports the SHIQ(D) subset of OWL-DL. This includes all features of OWL-DL apart from nominals (also known as enumerated classes). Since nominals are not a part of OWL Lite, KAON2 supports all of OWL Lite.

KAON2 also supports the so-called DL-safe subset of the Semantic Web Rule Language (SWRL). The restriction to the DL-subset has been chosen to make reasoning decidable.

The API of KAON2 is capable of manipulating F-Logic ontologies. For reasoning, KAON2 supports the function-free subset of F-Logic, currently with limited support for default negation.

KAON2 supports answering conjunctive queries, albeit without true non-distinguished variables. This means that all variables in a query are bound to

individuals explicitly occurring in the knowledge base, even if they are not returned as part of the query answer.
Queries can be formulated using SPARQL. Much of, but not entire SPARQL specification is supported. In particular, only those queries are supported which correspond naturally to conjunctive queries. For example, queries with variables at predicate positions are currently not supported, because this would require answering conjunctive queries with variables at predicate positions, thus requiring second-order logic. Furthermore, OPTIONAL and GRAPH patterns are not supported, because they are difficult to formalize in logic. The implemented variant of SPARQL also extends the specification, by allowing for computed values and explicitly typed query literals. Alternatively, queries can be formulated in F-Logic.
Contrary to most currently available DL reasoners, such as FaCT, FaCT++, RACER, DLP or Pellet, KAON2 does not implement the tableaux calculus. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program. For an overview of these algorithms, please refer to [38]. A detailed (and quite lengthy) technical presentation of all algorithms is given in [39].
These novel algorithms allow applying well-known deductive database techniques, such as magic sets or join-order optimizations, to DL reasoning. According to the evaluation performed in [40], such algorithms make answering queries in KAON2 one or more orders of magnitude faster than in existing systems.
KAON2 is available as a precompiled binary distribution and is free of charge for universities for noncommercial academic usage (national laboratories are not considered universities). For commercial purposes, there is a commercial version of KAON2 called OntoBroker OWL.

A.IV.7. RACER.

RacerPro [49] stands for Renamed A-Box and Concept Expression Reasoner Professional. Its origins are within the area of description logics. Since description logics provide the foundation of international approaches to standardize ontology languages in the context of the so-called semantic web, RacerPro can also be used as a system for managing semantic web ontologies based on OWL (e.g., it can be used as a reasoning engine for ontology editors such as Protégé). However, RacerPro can also be seen as a semantic web information repository with optimized retrieval engine because it can handle large sets of data descriptions (e.g., defined using RDF). Last but not least, the system can also be used for modal logics such as Km.

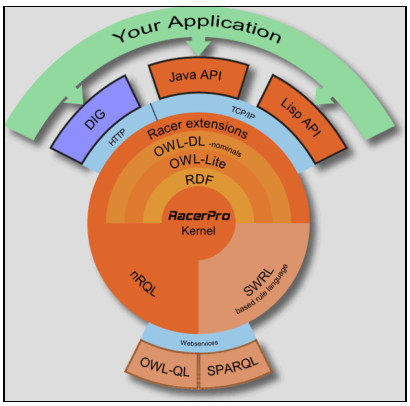


Fig. 46.- Racer Pro Architecture.

RacerPro can process OWL Lite as well as OWL DL documents (knowledge bases). Some restrictions apply, however. OWL DL documents are processed with approximations for nominals in class expressions and user-defined XML datatypes are not yet supported. A first implementation of the semantic web rule language (SWRL) is provided with RacerPro1.9. The following services are provided for OWL ontologies and RDF data descriptions:

- Check the consistency of an OWL ontology and a set of data descriptions.
- Find implicit subclass relationships induced by the declaration in the ontology.
- Find synonyms for resources (either classes or instance names).
- Since extensional information from OWL documents (OWL instances and their interrelationships) needs to be queried for client applications, an OWL-QL query processing system is available as an open-source project for RacerPro.
- HTTP client for retrieving imported resources from the web. Multiple resources can be imported into one ontology.
- Incremental query answering for information retrieval tasks (retrieve the next n results of a query). In addition, RacerPro supports the adaptive use of computational resource: Answers which require few computational resources are delivered first and user applications can decide whether computing all answers is worth the effort.

RacerPro is a knowledge representation system that implements a highly optimized tableau calculus for very expressive description logic. It offers reasoning services for multiple T-Boxes and for multiple A-Boxes as well. The system implements the description logic ALCQHIR+ also known as SHIQ (see [9]). This is the basic logic ALC augmented with qualifying number restrictions, role hierarchies, inverse roles and transitive roles. In addition to these basic features, RacerPro also provides facilities for algebraic reasoning including concrete domains for dealing with:

- Min/max restrictions over the integers,
- Linear polynomial (in-)equations over the reals or cardinals with order relations,
- Equalities and inequalities of strings.

For these domains, no feature chains can be supported due to decidability issues.

RacerPro supports the specification of general terminological axioms. A T-Box may contain general concept inclusions (GCIs), which state the subsumption relation between two concept terms. Multiple definitions or even cyclic definitions of concepts can be handled by RacerPro.
RacerPro implements the HTTP-based **quasi-standard DIG for interconnecting DL** systems with interfaces and applications using an XML-based protocol [46]. RacerPro also implements most of the functions specified in the older Knowledge Representation System Specification(KRSS), for details see [50].
Given a T-Box, various kinds of queries can be answered. Based on the logical semantics of the representation language, different kinds of queries are defined as inference problems (hence, answering a query is called providing inference service). Next list enumerates the most important ones here:

- Concept consistency w.r.t. a T-Box: Is the set of objects described by a concept empty?
- Concept subsumption w.r.t. a T-Box: Is there a subset relationship between the set of objects described by two concepts?
- Find all inconsistent concept names mentioned in a T-Box. Inconsistent concept names result from T-Box axioms and it is very likely that they are the result of modeling errors.
- Determine the parents and children of a concept w.r.t. a T-Box: The parents of a concept are the most specific concept names mentioned in a T-Box which subsume the concept. The children of a concept are the most general concept names mentioned in a T-Box that the concept subsumes. Considering all concept names in a T-Box the parent (or children) relation defines a graph structure which is often referred to as taxonomy. Note that some authors use the name taxonomy as a synonym for ontology.

It must be remarked that whenever a concept is needed as an argument for a query, not only predefined names are possible, instead concept expressions allow for adaptive formulation of queries that have not been anticipated at system construction time.
If also an A-Box is given, among others, the following types of queries are possible:

- Check the consistency of an A-Box w.r.t. a T-Box: Are the restrictions given in an A-Box w.r.t. a T-Box too strong, i.e., do they contradict each other? Other queries are only possible w.r.t. consistent A-Boxes.
- Instance testing w.r.t. an A-Box and a T-Box: Is the object for which an individual stands a member of the set of objects described by a specified concept? The individual is then called an instance of the concept.
- Instance retrieval w.r.t. an A-Box and a T-Box: Find all individuals from an A-Box such that the objects they stand for can be proven to be a member of a set of objects described by a certain query concept.
- Retrieval of tuples of individuals (instances) that satisfy certain conditions w.r.t. an A-Box and a T-Box.
- Computation of the direct types of an individual w.r.t. an A-Box and a T-Box: Find the most specific concept names from a T-Box of which a given individual is an instance.
- Computation of the fillers of a role with reference to an individual w.r.t. an A-Box and a T-Box.

- Check if certain concrete domain constraints are entailed by an A-Box and a T-Box.

RacerPro provides another semantically well-defined query language (nRQL, new Racer Query Language), which also supports negation as failure, numeric constraints w.r.t. attribute values of different individuals, substring properties between string attributes, etc. In order to support special OWL features such as annotation and datatype properties, special OWL querying facilities have been incorporated into nRQL. The query language OWL-QL [4] is the W3C recommendation for querying OWL documents. nRQL has been used as a basic engine for implementing a very large subset of the OWL-QL query language (see above).

For some representation purposes, e.g., reasoning about spatial relations such as contains, touching, etc., relational algebras and constraint propagation have proven to be useful in practice. RacerPro combines description logics reasoning with, for instance, reasoning about spatial (or temporal) relations within the A-Box query language nRQL. Bindings for query variables that are determined by A-Box reasoning can be further tested with respect to an associated constraint network of spatial (or temporal) relationships.

Although RacerPro is one of the first systems supporting this kind of reasoning in combination with description logics (or OWL), we expect that international standardization efforts will also cover these important representation constructs in the near future. Note also that the semantically well-founded treatment can hardly be efficiently achieved using rule systems.

Racer Systems provides several different editions of the Racer technology.

- **RacerPro** is a server for description logic or OWL inference services. With RacerPro you can implement industrial strength projects as well as doing research on knowledge basis and develop complex applications. If you do not have a valid license, you are allowed to use RacerPro but some restrictions apply.
- **RacerPorter** The “Porter to RacerPro" is the graphical user client for RacerPro. RacerPorter uses the TCP/IP network interface to connect to one or more RacerPro servers and helps you manage them: You can load knowledge bases, switch between different taxonomies, inspect your instances, visualize T-Boxes and A-Boxes, manipulate the server and much more. RacerPorter is already included in the installer versions of RacerPro for Windows and Mac OS X and separately available for Linux systems with a graphic display.
- **RacerPlus** To minimize the performance overhead due to network-based communication between RacerPorter and RacerPro as well as to utilize the computing power offered by a single workstation they introduce RacerPlus, an integrated workbench which includes RacerPro and the RacerPorter graphical user interface in a single application.
- **RacerMaster** actually is RacerPro as an object code library “fast file”. You can develop your own application and use Racer technology without an external server application.

RacerPro is not freely distributed.

A.IV.8. FaCT.

FaCT (Fast Classification of Terminologies) [51] is a Description Logic (DL) classifier that can also be used for modal logic satisfiability testing. It was developed within the Camelot Project[52] and the system includes two reasoners, one for the logic SHF (ALC augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic SHIQ (SHF augmented with inverse roles and qualified number restrictions), both of which use sound and complete tableaux algorithms. FaCT's most interesting features are:

- its expressive logic (in particular the SHIQ reasoner): SHIQ is sufficiently expressive to be used as a reasoner for the DLR logic and hence to reason with database schemata;
- its support for reasoning with arbitrary knowledge bases (i.e., those containing general concept inclusion axioms);
- its optimised tableaux implementation (which has now become the standard for DL systems) and
- its CORBA based client-server architecture.

FaCT is written in Common Lisp and has been run successfully with several commercial and free lisps, including Allegro, Liquid (formerly Lucid), Lispworks and GNU. As the source code is available (under the GNU general public license), FaCT can be run on any system where a suitable Lisp is available. Binaries (executable code) are also available (in addition to the source code) for Linux and Windows systems, allowing FaCT to used without a locally available Lisp. Moreover, a FaCT server running on a suitable host machine can be used (via its CORBA interface) on any system that has network access to the server. A new FaCT DIG servlet is also available. As well as providing a lighter-weight HTTP interface, this has the advantage that, for Java applications, a direct in-memory connection (side-stepping the HTTP communication) can be used.

A.IV.9. FaCT++.

FaCT++ [53] is the new generation of the well-known FaCT OWL-DL reasoner. FaCT++ uses the established FaCT algorithms, but with a different internal architecture. Additionally, FaCT++ is implemented using C++ in order to create a more efficient software tool and to maximize portability. New optimizations have also been introduced and some new features added.

FaCT++ is released under a GNU public license and is available for download both as a binary file and as source code. To build FaCT++ you will need a C++ compiler (GNU gcc v3.3 and higher have been used successfully) and GNU make. In order to build a DIG version of a reasoner, the XML parsing library Xerces-C++ is also required. This is freely available at http://xml.apache.org/xerces-c.

A.IV.10. Pellet.

Pellet is an OWL 2 reasoner. It provides standard and cutting-edge reasoning services for OWL ontologies. It includes support for OWL 2 profiles including OWL 2 EL. It incorporates optimizations for nominals, conjunctive query, answering and incremental reasoning. It is implemented in Java and is open sourced under a liberal license; it is available under dual licensing terms:

- For open source applications, Pellet may be used under the terms of the AGPL version 3 license.
- For proprietary, closed source applications and other commercial applications, Pellet is available under alternative license terms.

Pellet has been the first reasoner to support all of OWL-DL, i.e. the Description Logic (DL) SHOIN(D) [56]. OWL 1.1 extends OWL-DL with qualified cardinality restrictions, complex subproperty axioms (between a property and a property chain), local reflexivity restrictions, reflexive, irreflexive, symmetric and anti-symmetric properties, disjoint properties. It offers a panoply of features including conjunctive query answering, rule support, E-Connection reasoning and axiom pinpointing, among others. To make its reasoning capabilities easily accessible to users, Pellet provides various interfaces including a commandline interface, an interactive Web form for zero-install use, DIG server implementation and API bindings for RDF/OWL toolkits Jena and Manchester OWL-API.

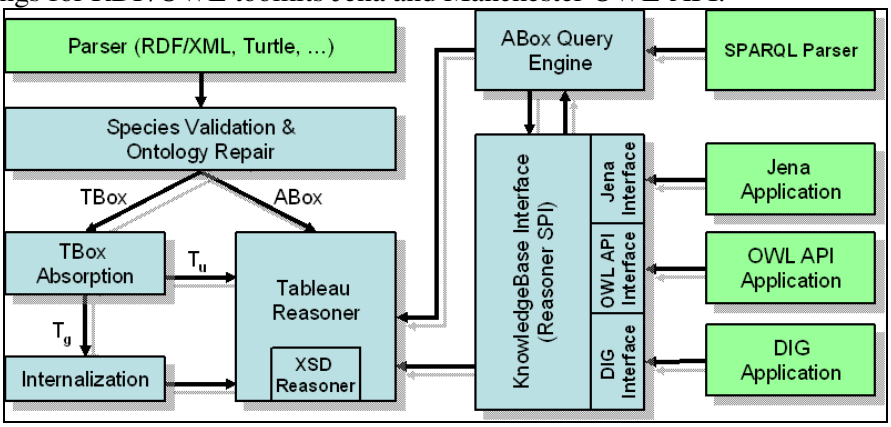


Fig. 47.- Pellet Architecture.

Pellet implements most of the state of the art optimization techniques provided in the DL literature including Normalization, Simplification, Absorption, Semantic Branching, Backjumping, Caching Satisfiability Status, Top-Bottom Search for Classification and Model Merging.

In addition, Pellet incorporates several novel optimizations to improve the reasoning performance in the presence of *nominals* (enumerated classes) and individuals. Reasoning with nominals is especially challenging since some of the existing optimizations in DL reasoners, such as Chain Contraction, are not applicable anymore. Moreover, in the presence of nominals, assertions about instances can affect the concept satisfiability and classification results. A new suite of optimizations were developed to tackle this problem. Two key ones are *Nominal Absorption*, a technique where axioms involving enumerations are absorbed into type assertionsand *Nominal-based Model Merging*, a technique to detect obvious non-subsumptions and non-instantiations by exploiting the fact that nominals always have a fixed interpretation in the domain.

Another novel optimization technique implemented in Pellet is for incremental reasoning against dynamic knowledge bases. In many contexts (ontology editors, web portals, sensor streams), the knowledge base is in constant flux. We have Techniques to reuse the reasoning results from previous steps to process updates

incrementally has been included in Pellet. They seem to work up to three orders of magnitude the speed after an A-Box addition or deletion.
Finally we brief introduce the supposed Features and Capabilities of Pellet:

- **Conjunctive A-Box Query**: Pellet includes a query engine that can efficiently answer conjunctive A-Box queries expressed in SPARQL or RDQL.
- **Datatype Reasoning**: Pellet uses the type system approach to support reasoning with datatypes. In particular, Pellet has a datatype oracle that can reason with XML Schema based datatypes. The datatype oracle can check the consistency of conjunctions of (built-in or derived) XML Schema datatypes.
- **Axiom Pinpointing and Debugging**: Axiom pinpointing is a nonstandard DL inference service that provides a justification for any arbitrary entailment derived by a reasoner from an OWL-DL knowledge base. Given an ontology and any of its logical consequences, the axiom pinpointing service determines the premises in the KB that are sufficient for the entailment to hold. The justification is useful for understanding the output of the reasoner, which is key for many tasks, such as ontology debugging, design and evolution: when an inconsistency is detected in the ontology, a single set of axioms causing the problem can be extracted.
- **Integration with Rules Formalism**: Pellet is coupled with a Datalog reasoner to implement the AL-Log framework for combining DLs with rules. AL-Log combines Datalog and DLs by allowing DL classes to be used in the body of a rule.
- **Multi-Ontology Reasoning using E-Connections**: In addition to the owl:imports mechanism, Pellet supports a novel ontology combination technique based on E-Connections to reason with multiple ontologies. The E-Connections are a general framework for combining several families of decidable logics. Using this technique, ontologies can be linked to each other without losing their context (in contrast to owl:imports which simply merges ontologies).
- **Non-monotonic Reasoning**: Non-monotonic logics have been generally successful in capturing several forms of common sense and database reasoning. A prominent family of non-monotonic formalisms is rooted in various forms of the closed world assumption (CWA). The DL ALCK [10] adds a non-monotonic K operator (which is a kind of necessity operator) to the DL ALC to provide the ability to “turn on” the CWA when needed. The reasoning support for ALCK language has been implemented in Pellet to answer CWA queries that use the K operator. We also admit a restricted use of K in the ontologies, in the form of an epistemic rule.

A.IV.11. SESAME.

Sesame [75] is an open source repository for storing and querying RDF and RDFS information. Sesame is being developed by Aidministrator Nederland b.v.3 as part of the European IST project On-To-Knowledge4 [76].
Sesame allows persistent storage of RDF data and schema information and provides access methods to that information through export and querying modulesOWL

ontologies are simply treated on the level of RDF graphs. Sesame enables the connection to DBMS (currently MySQL, PostgreSQL and Oracle) through the **SAIL** (*Storage and Inference Layer*) module and also offers a very efficient direct-to-disk SAIL called Native SAIL, which we used for our experiments. Sesame provides RDFS inferencing and allows querying through SeRQL, RQL, RDQL and SPARQL. Via the SAIL it is also possible to extend the inferencing capabilities of the system.

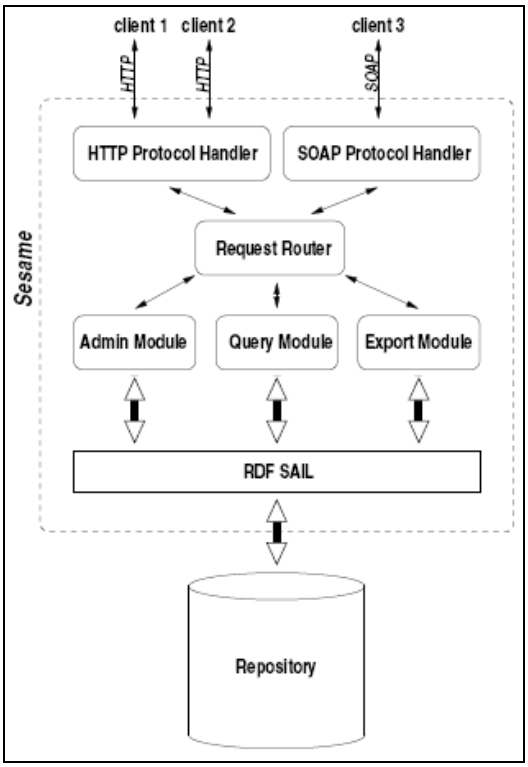


Fig. 48.- SESAME Architecture.

The RQL query module As we have seen, one of the three modules currently implemented in Sesame is an RQL query engine. The Sesame version of RQL features better compliance to W3C specifications, including support for optional domain- and range restrictions as well as multiple domain- and range restrictions. It does, however, not feature support for datatyping as proposed in the original language proposal.

The Query Module follows the path depicted in next figure when handling a query. After parsing the query and building a query tree model for it, this model is fed to the query optimizer which transforms the query model into an equivalent model that will evaluate more efficiently. These optimizations mainly consist of a set of heuristics for query subclause move-around. Notice that these preevaluation optimizations are not dependent on either domain or storage method.

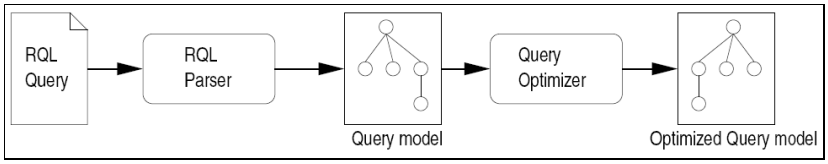


Fig. 49.- SESAME: RQL query module.

The optimized model of the query is subsequently evaluated in a streaming fashion, following the tree structure into which the query has been broken down. Each object represents a basic unit in the original query and evaluates itself, fetching data from the SAIL where needed. The main advantage of this approach is that results can be returned in a streaming fashion, instead of having to build up the entire result set in memory first. In Sesame, RQL queries are translated (via the object model) into a set of calls to the SAIL. This approach means that the main bulk of the actual evaluation of the RQL query is done in the RQL query engine itself.

A.IV.12. OWLIM.

OWLIM[77][78] is semantic repository and reasoner, packaged as a SAIL for the Sesame RDF database. OWLIM uses the TRREE (Triple Reasoning and Rule Entrainment Engine)[79] engine to perform RDFS and OWL DLP reasoning. It performs forward-chaining of entailment rules on top of RDF graphs and employs a reasoning strategy, which can be described as total materialization.
OWLIM offers configurable reasoning support and performance: it is available in several versions, tailored to meet different requirements. The two major flavors of OWLIM are:

- **SwiftOWLIM**, performing reasoning and query evaluation in-memory, while a reliable persistence strategy assures data preservation and consistency. It scales to about 10 mill. statements on a 32-bit notebook and further, when proportional amount of memory is allocated. SwiftOWLIM is an open-source library, under LGPL.
- **BigOWLIM**, operating with file-based indices, which allows it to scale to billions of statements even on desktop machines. In the ”standard” version of OWLIM (referred to as SwiftOWLIM) reasoning and query evaluation are performed in-memory, while a reliable persistence strategy assures data preservation, consistency and integrity. BigOWLIM is available under an RDBMS-like commercial license on a per-server-CPU basis; it is neither free nor open-source.

OWLIM uses Sesame as a library, taking advantage of its APIs for storage and querying, as well as the support for a wide variety of query languages (e.g. SPARQL and SeRQL) and RDF syntaxes (e.g. RDF/XML, N3, Turtle). The development of OWLIM is partly supported by SEKT, TAO, TripCom, LarKC and other FP6 and FP7 European research projects.

A.IV.13. HermiT.

HermiT[81] is a freely available theorem prover for description logics. The reasoner currently fully handles the DL SHIQ. The support for SHOIQ is currently being worked on. The main supported inference is the computation of the subsumption

hierarchy. HermiT can also compute the partial order of classes occurring in an ontology. HermiT implements a novel hypertableau reasoning algorithm. The main aspect of this algorithm is that it is much less non-deterministic than the existing tableau algorithms.

The HermiT system also serves as a platform for prototypical implementations of new language features. For example, HermiT already includes support for reasoning with ontologies which include description graphs.
HermiT is available as an open-source Java library and includes both a Java API and a simple command-line interface. We use the OWL API both as part of the public Java interface and as a parser for OWL files; HermiT can process ontologies in any format handled by the OWL API, including RDF/XML, OWL Functional Syntax, KRSS and OBO

A.IV.14. SCREECH: an Approximate Reasoner.

SCREECH [70] is an approximate OWL reasoning system. It builds on the KAON2 system and performs OWL A-Box reasoning in an approximate manner. It trades soundness of reasoning for efficiency, with resulting polynomial worst-case data complexity. It has been developed for use in time-critical applications where quick response time is more important than a full guarantee of correctness of answers.
The SCREECH approach is based on the fact that data complexity is polynomial for non-disjunctive datalog, while for OWL DL it is NP complete even in the absence of nominals. SCREECH utilises the KAON2 algorithms but rather than doing exact reasoning over the resulting disjunctive datalog knowledge base, it does approximate reasoning by treating disjunctive rules as if the where non-disjunctive ones, they are approximated by Horn rules.
SCREECH is presented in three different variants:

- **SCREECH ALL**: it is complete, but may be unsound incases. Its data complexity is polynomial. It uses a modified notion of the *split program* in order to deal with disjunctive datalog. Given a rule:

  $$H1 \vee \cdot \cdot \cdot \vee Hm \leftarrow A1, \ldots, Ak,$$

  As an output of the KAON2 transformation algorithm the *derived split rules* are defined as:

  $$H1 \leftarrow A1, \ldots, Ak \quad \vdots \quad Hm \leftarrow A1, \ldots, Ak$$

  For a given disjunctive program P its *split program* P′ is defined as the collection of all split rules derived from rules in P. It can be easily shown that for instance retrieval tasks, the result obtained by using the split program instead of the original one is complete but may be unsound.
- **SCREECH-NONE**: is defined by simply removing all disjunctive rules (and all integrity constraints) after the transformation by the KAON2-algorithm.
- **SCREECH-ONE**: is defined by replacing each disjuntive rules by exactly one of the split rules. This selection can be done randomly, but will be most

useful if the system has some knowledge – probably of statistical nature – on the size of the extensions of the named classes.

A.IV.15. Access to Other Reasoners.

We present here some of the most popular reasoners at the moment although there are many others which optimize different aspects of the reasoning process and perhaps can best fit the reader requirements. During the researching process we followed to create this documents we find the comprehensive list of Description Logic reasoners which is maintained by Uli Sattler at the University of Manchester [58]. We recommend you to visit it to obtain a 360º view of the reasoners.

A.IV.16. Benchmarking of OWL Reasoners.

From the previous paragraph it could be followed that today there is a huge quantity of reasoners that are based on quite different design decisions in addressing the tradeoff between complexity and expressiveness on the one hand and scalability on the other hand: **Classical Description Logic reasoners** based on tableau algorithms are able to classify large, expressive ontologies as often found e.g. in the bio-medical domain, but they often provide limited support in dealing with large number of instances. **Database-like reasoners** that materialize inferred knowledge upfront are able to handle large amounts of assertional facts, but are in principle limited in terms of the logic they are able to support. Deciding for an appropriate reasoner for a given application task is far from trivial. In order to support such decisions, comparisons of reasoners based on benchmarks are required.

We briefly introduce the comparison Bock and col. performed in [68]. While number performance evaluations for OWL reasoners have already been performed previous to their work, all of them so far targeted only special purpose tasks, e.g. focusing on classical description logic reasoning tasks. They aimed to go a step further and intend to provide guidance for selecting the appropriate reasoner for a given application scenario. In order to do so, they provided a survey of the ontology landscape, discuss typical reasoning tasks and define a comprehensive benchmark. Based on the benchmark results they identify which reasoners are most adequate for which classes of ontologies and corresponding reasoning tasks.

In order to evaluate each of the reasoners they considered two different Performance Measures:

- **Load Time (P)**: Includes the time to do some important preparation before querying, e.g. load ontologies and check A-Box consistency.
- **Response Time (Q)**: Starts with executing the query and ends when all the query results were stored into a local variable. Usually, the query time means when a query is executed while not including the time for iterating the results.

The evaluation is based on the use of already well know ontologies: VICODI[71], SWRC[72], LUMB[73] and Wine[74] which are defined on different OWL subsets (i.e. have different DL capabilities) and the queries some benchmarking studies performed against this ontologies. Although they did not consider all the reasoners we present in this document we think their conclusions could guide us in the selection of a reasoner. Their benchmarks showed that it is important to understand

the strengths and weaknesses of the different approaches in order to select an adequate reasoner for a given reasoning task. It does not come as a surprise that there is no clear "winner" that performs well for all types of ontologies and reasoning tasks. The main conclusions they showed are:

- Reasoners that employ a simple rule engine scale very well for large A-Boxes, but are in principle very limited to lightweight language fragments.
- Classical tableau reasoners scale well for complex T-Box reasoning tasks, but are limited with respect to their support for large A-Boxes.
- The reasoning techniques based on reduction to disjunctive datalog as implemented in KAON2 scale well for large A-Boxes, while at the same time they support are rich language fragment.
- If nominals are important for a given scenario, Pellet is the only reasoner in this benchmark, which has adequate support.

The next picture summarizes the results of their studies:

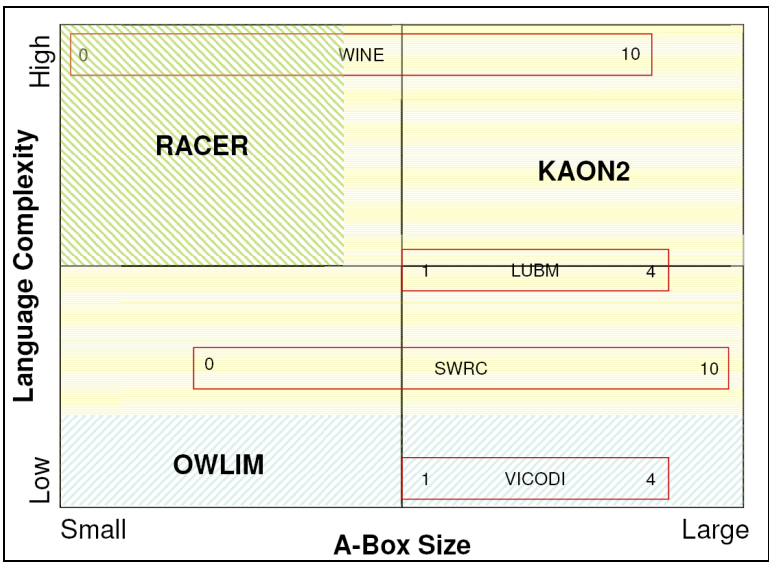


Fig. 50.- J.Bock's Benchmarking Selection Recommendation.

As it was mentioned in this article, the Framework introduced in [69], which is based on the use of the DIG interface, is a good tool for performing reasoner test.

A.IV.17. A Broker of Reasoners: An Interesting Investment in the Future.

The upcoming W3C standard for OWL 2 (Web Ontology Language version 2) [66] distinguishes different Profiles [67], namely OWL 2 EL, OWL 2 QL and OWL 2 RL, which trade language expressivity for reduced reasoning complexity. Currently there is a number of reasoning systems being developed focusing on efficient reasoning on those fragments. Apart from traditional sound and complete reasoning systems, such as Pellet [44], FaCT++ [53], or KAON2 [37], there are approximate reasoners, such as SCREECH [165] and AQA [166], which trade soundness and/or completeness for runtime performance. In certain scenarios where speed is highly desired, such a tradeoff can be tolerated. Bock and col. introduce in [65] an

interesting approach: a reasoning broker framework, which connects to different existing reasoning systems and intelligently delegates reasoning requests.
The idea behind reasoning brokerage is to use different existing reasoners in the background while providing a single interface to the user or application. The broker invokes connected remote reasoners in an intelligent manner such that their strengths and weaknesses are considered and the shortest possible runtime for a particular reasoning task (or a sequence of tasks) is achieved. The ideal defined broker will provide the features listed below:

- **Parallel Reasoner Invocation**: the parallel execution of reasoning tasks on a set of reasoning systems.
- **Reasoner Selection**: due to benchmarks [68] and knowledge about the implementations of different reasoning systems, some of the available reasoners can be selected prior to actually executing the reasoning tasks (depending on the ontology the query).
- **Query Decomposition:** Queries containing complex class or property expressions can be analyzed if those expressions decompose into several subexpressions which can be answered by different reasoners in parallel. It must be ensured, though, that the combination of the results delivered by different reasoners does not bear an unacceptable overhead compared to the performance gained by parallel computation.
- **Partitioning of Ontologies**: The notions of *conservative extensions* and *locality* provide means to partition ontology into several semantically independent modules. Executing a query on such a module instead of the whole ontology can result in run-time performance improvements for reasoning requests. Moreover, in combination with intelligent query decomposition more complex queries can be executed by different reasoners on different ontology modules in parallel.
- **Load Balancing and Scheduling:** to this end an asynchronous reasoner interface would allow for acceptance of more than a single query at once from an application. Query answering can then be scheduled to be processed by the different remote reasoners according to their strengths and language conformance, in order to ensure maximum throughput of queries.
- **Anytime Reasoning**: *Anytime Algorithms* are designed to gradually improve the quality or quantity of their results as computation time increases and end with providing the whole answer if complete computation is required.
- **Real-time Benchmarking:** both performance measurements and measuring the quality of answers have to be taken into account by and ideal broker infrastructure.

The HERAKLES system, coded in the JavaTM programming language and based on the OWL API[54] is the first implementation of an reasoning broker. It is implemented in a client/server architecture to ensure modular decoupling of remote reasoners:

- The HERAKLES client implements the OWLReasoner interface of the OWL API and can thus be used like any standard reasoner from within an OWL API based application. The HERAKLES client furthermore maintains

a reasoner registry to record attached remote reasoners that can be used by the broker.

- Remote reasoners are wrapped into a remote reasoner adapter, which allows them to be run as reasoning servers connected to the HERAKLES client. This adaptation has not only been realized for OWL API compliant reasoners, but also for the KAON2 reasoner with its own API and the KAON2 based approximate reasoning systems SCREECH and AQA.

The communication between client and servers has been realized using JavaTM RMI3. Hence reasoning servers can be run on remote machines, which allows for exclusive provision of computational resources for each reasoner.

HERAKLES can be selected and used the same way as any standard reasoner from within Protégé. The plug-in provides additional functionality, namely (i) selection of remote reasoners to be used, (ii) strategy selection and configuration, (iii) anytime querying with asynchronous result delivery and (vi) real-time statistics of run-time performance of the attached remote reasoners.

The behaviour of the broker and thus the implementation of the features previously introduced is controlled by exchangeable broker **strategies**. There is a **load strategy** to control the loading of ontologies into the different remote reasoners and an **execution strategy** to control the execution of reasoning tasks by those reasoners. The strategy concept allows for easy substitution of both load and execution strategy by different implementations depending on the usage scenario of the reasoning broker. Furthermore the strategy concept allows for the implementation and use of customised strategies for specific use cases. Implementation of strategies in HERAKLES is simplified by several strategy components, which encapsulate core broker tasks such as parallelisation, reasoner selection, partitioning, or ontology analysing. These strategy components can then be used and combined to assemble new broker strategies. The following paragraphs describe interfaces and currently available implementations of strategy components and strategies for HERAKLES:

- **Paralleliser**: this component invokes the execution of a reasoning task on a selection of reasoners in parallel. Currently there are two implementations: a competing paralleliser, which delivers the result of the reasoner that finishes first and a blocking paralleliser, which waits until all reasoners have finished.
- **Selector:** this component selects a set of reasoners out of the ones registered by the broker. Different implementations can apply different selection criteria, such as ontology properties, reasoning task to be executed, or query properties. Selection of reasoners in this way requires knowledge about the capabilities of the different reasoners, which are currently recorded by the remote reasoner adapters.
- **Modulariser:** this component provides means to partition an ontology into several modules, which can ideally be processed by different reasoners concurrently.
- **Analyser:** this component is supposed to be used in load strategies which perform an analysis of the ontologies to be loaded. The information gained by this analysis, i.e. characteristics of the ontologies, can then be used e.g. by selectors in the execution phase.

- **Basic/Analysing Load Strategy**: This load strategy loads the ontology into all available remote reasoners. The analysing load strategy extends the basic load strategy by additionally analysing the ontologies and recording their characteristics.
- **Basic/Fault-tolerant Parallelisation Strategy:** this execution strategy performs a reasoning request on all available remote reasoners, which have loaded the ontologies. A fault-tolerant parallelisation strategy extends the basic parallelisation strategy by being insensitive to failing remote reasoners. In case of a failure, it waits for more reasoners to become available and fails on a particular query only if all remote reasoners fail.
- **(Fault-tolerant) Task Selection Strategy:** This execution strategy selects remote reasoners according to the reasoning task. Selected reasoners are then invoked. A fault-tolerant task selection strategy extends the task selection strategy by being insensitive to failing remote reasoners. In the case all selected reasoners fail on a particular query, it also selects reasoners not matching the selection criteria in order to try and have the query succeed.
- **Anytime Strategy:** this execution strategy simulates anytime reasoning behaviour by using approximate reasoning systems. It selects distinct sets of remote reasoners respecting soundness, completeness and both soundness and completeness. All reasoners are invoked in parallel. Results are delivered from the fastest reasoner of each set, characterising results as sound, complete, or sound/complete rsp. Anytime behaviour arises from the faster run-times of the approximate reasoners and thus from the early delivery of (potentially) unsound or incomplete answers.
- **Benchmark Strategy:** this execution strategy can be used for simple run-time performance benchmarking of reasoners. It invokes all available remote reasoners in parallel without any prior selection.

Appendix V. References.

We now proceed to introduce some available open source format transformer tools in order to chose the one which better fits the problem we want to solve: to transform the e-Contract Legal XML representations of the contracts (which are thought to be human readable) to an OWL file (which is machine readable).

A.V.1 Languages for Markup Document Transformation.

A.V.1.1 DOM a Primitive Way to Transform Markup Documents.

The Document Object Model [30][31] is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented document (page).

As it was presented in [31], the DOM specification defines the Document Object Model, a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

A Web document is therefore represented as a document tree. Each node in the tree is an object in the object-oriented programming sense. Data are hidden so that they cannot be manipulated directly and functions (methods) are associated with the object to manipulate the data it contains. The DOM thus identifies:

- The interfaces and objects used to represent and manipulate a document.
- The semantics of these interfaces and objects - including both behaviour and attributes.
- The relationships and collaborations among these interfaces and objects.

DOM is being designed at several levels:

- Level 1. This concentrates on the core, HTML and XML document models. It contains functionality for document navigation and manipulation.
- Level 2. Includes a style sheet object model and defines functionality for manipulating the style information attached to a document. It also enables traversals on the document, defines an event model and provides support for XML namespaces. DOM Level 1 is contained in DOM Level 2.

- Level 3. Will address document loading and saving, as well as content models (such as DTDs and schemas) with document validation support. In addition, it will also address document views and formatting, key events and event groups.
- Further Levels. These may specify some interface with the possibly underlying window system, including some ways to prompt the user. They may also contain a query language interface and address multithreading and synchronization, security and repository.

We now introduce some of the basic features of DOM as they were presented in [32]. DOM provides methods for:

- Methods for traversing the document tree.
- Methods for changing the structure of the document tree.
- Methods for retrieving and changing the values associated with the nodes of the tree.
- In addition, the HTML DOM provides methods specific to particular types of node, for example to get the attribute values of an HTML element node.

In addition the structure of a DOM document was introduced. The root of an HTML document is an HTMLDocument object. Properties of this object give links into elements in the document and the object provides methods which allow the structure of the document to be extended or replaced using strings of unparsed HTML. The HTML DOM defines objects corresponding to the HTML element types. HTML attributes are then exposed as properties on the element objects.

A.V.1.2 SAX an Introduction.

SAX (Simple API for XML) [27[28] is a serial access parser API for XML. It provides a mechanism for reading data from an XML document. A SAX parser is implemented as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include:

- XML Text nodes
- XML Element nodes
- XML Processing Instructions
- XML Comments

Events are fired when each of these XML features are encountered and again when the end of them is encountered. XML attributes are provided as part of the data passed to element events. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

Unlike DOM, there is no formal specification for SAX. The Java implementation of SAX is considered to be normative and implementations in other languages attempt

to follow the rules laid down in that implementation, adjusting for the differences in language where necessary.

The event-driven model of SAX is useful for XML parsing, but it does have certain drawbacks. Certain kinds of XML validation require access to the document in full. For example, a DTD IDREF attribute requires that there be an element in the document that uses the given string as a DTD ID attribute. To validate this in a SAX parser, one would need to keep track of every previously encountered ID attribute and every previously encountered IDREF attribute, to see if any matches are made. Furthermore, if an IDREF does not match an ID, the user only discovers this after the document has been parsed; if this linkage was important to building functioning output, then time has been wasted in processing the entire document only to throw it away.

Additionally, some kinds of XML processing simply require having access to the entire document. XSLT and XPath, for example, need to be able to access any node at any time in the parsed XML tree. While a SAX parser could be used to construct such a tree, the DOM already does so by design.

A.V.1.3 XSLT a Short Introduction.

We now briefly introduce XLST in order to be able to evaluate its capabilities. We follow [24][25] expositions of XLST were it was firstly presented following the history of the Markup languages for information presentation and then its main characteristics were presented.

The XSLT is an official recommendation of the World Wide Web Consortium (W3C) known as W3C. XSLT 2.0, the most recent specification, was edited by Michael Kay and published on January 23, 2007. The previous specification, XSLT 1.0, was edited by James Clark and published on November 16, 1999. The released 2.0 of XSLT and Xquery were published as candidates by the W3C.

XSLT has its origins in the aspiration to separate information content from presentation on the Web. Drawing on experience with SGML in the print publishing world, XML was defined early in 1998 as a markup language to represent structured content independent of its presentation. Unlike HTML, which uses a fixed set of concepts (such as paragraphs, lists and tables), the tags used in XML markup are entirely user defined and the intention is that they should relate to objects in the domain of interest (such as people, places, prices and dates). Whereas the elements in HTML are essentially typographic (albeit at a level of abstraction), the aim with XML is that the elements should describe real-world objects.

The W3C defined two families of style sheet standards. The first, known as CSS (Cascading Style Sheets), is widely used with HTML, though it can also be used with XML. CSS has no ability to perform computations, to rearrange or sort the data, to combine data from multiple sources, or to personalize what is displayed according to characteristics of the user or session. For these reasons W3C embarked

on the development of a richer style sheet language to be known as XSL (Extensible Stylesheet Language), taking many of the intellectual ideas from DSSSL (Document Style, Semantics and Specification Language), as developed in the SGML community.

During the development of XSL (and this had already been foreshadowed in DSSSL), it emerged that the tasks to be performed in preparing an XML document for display could be split into two stages: transformation and formatting. Transformation is a process of converting one XML document into another. Formatting is the process of converting the transformed tree structure into a two-dimensional graphical representation, or perhaps a one-dimensional audio stream. XSLT was developed as a language to control the first stage, transformation. Development of the second stage, formatting, is work still in progress.

Separating transformation into one language and formatting into another proved to be a really good decision, because it turned out that there are lots of applications for a transformation language that have nothing to do with displaying documents to the user. As XML becomes more widely used as a data interchange syntax in electronic business, there is an increasing need for applications to convert data from one XML vocabulary to another. There are many useful data transformations in which the source and target vocabularies are the same. These include data filtering, as well as business operations such as applying a price increase. Increasingly therefore, as data starts to flow around the system in XML syntax, XSLT starts to become a ubiquitous high-level language for manipulating it.

Some of the key features of the XSLT language are outlined below.

- **An XSLT style sheet is an XML document**. It means that all the lexical apparatus of XML (for example Unicode character encoding and escaping, use of external entities and so on) is available automatically. It means that it is easy to make an XSLT style sheet the input or output of a transformation, giving the language a reflexive capability. It also makes it easy to embed chunks of the desired XML output within the style sheet: many simple style sheets can be written essentially as a template for the desired output document, with occasional instructions embedded in the text to insert variable data from the input, or to compute a value.
- **The basic processing paradigm is pattern matching.** An XSLT style sheet consists of a set of template rules, each of which takes the form "if this condition is encountered in the input, then generate the following output." The order of the rules is immaterial and there is a conflict-resolution algorithm applied when several rules match the same input. One respect in which XSLT differs from serial text processing languages, however, is that the input is not processed sequentially line by line. Rather, the input XML document is treated as a tree structure and each template rule is applied to a node in the tree. The template rule itself can decide which nodes to process next, so the input is not necessarily scanned in its original document order.

The next picture summarizes the process of an XML document by an XSLT Processor which is explained bellow:

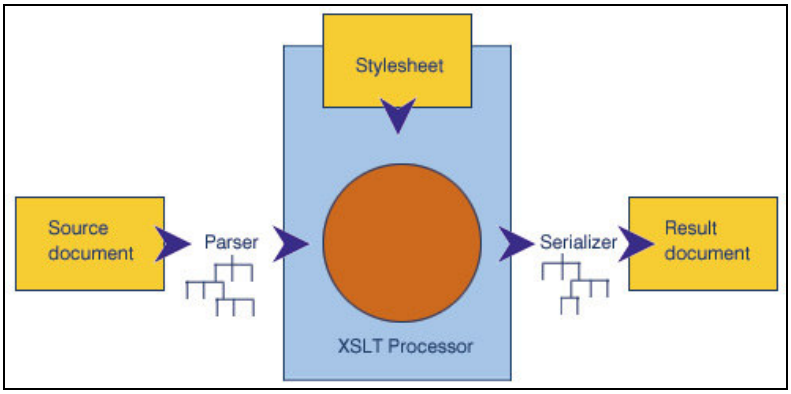


Fig. 51.- A XSLT Processor schema.

- The input tree structure will often be produced by parsing an XML document and the output tree structure will often be serialized into another XML document. But the XSLT processor itself manipulates tree structures, not XML character streams. First, it means that distinctions in the source document that are irrelevant to the tree structure are not accessible to the XSLT processor. More subtly, it means that processing an input element, or generating an output element, is an atomic operation. It is not possible to separate the processing of the element's start tag and end tag into separate operations, because an element is represented atomically as a single node in the tree model.
- XSLT uses a sublanguage called XPath to refer to nodes in the input tree. XPath is a separate W3C recommendation and can be used outside of XSLT. XPath is essentially a query language matched to XML's hierarchic data model. It is capable of selecting nodes by navigating the tree in any direction and applying predicates based on the value and position of the node. It also includes facilities for basic string manipulation, numeric computation and Boolean algebra. A simplified form of XPath expression, the pattern, is also used in template rules to define which nodes a particular template rule applies to.
- XSLT is based on the concepts of functional programming. A style sheet is made up of templates that are essentially pure functions -- each template defines a fragment of the output tree as a function of a fragment of the input tree and produces no side effects. The no-side-effects rule is applied quite strictly (with the exception of escapes into external code written in languages such as Java). The XSLT language allows variables to be defined, but does not allow an existing variable to change its value -- there is no assignment statement. Because of this style sheets can be applied incrementally. The theory is that if the language is free of side-effects, then when a small change is made to an input document it should be possible to

compute the resulting change to the output document without performing the entire transformation from scratch (at least in theory).

Finally we are going to state the main advantages of XSLT:

- XSLT gives you all the traditional benefits of a high-level declarative programming language, specialized to the task of transforming XML documents.
- The usual benefit cited for higher-level languages is development productivity. But in truth, the real value comes from *potential for change*. An XSLT application for transforming XML data structures can be made much more resilient to changes in the details of the XML documents than a procedural application coded using the low-level DOM and SAX interfaces. In the database world this feature is known as *data independence*.

Some critical opinion about XSLT may be found [26]:

- Some people may say that as with all declarative languages, there is a performance penalty. But for the vast majority of applications, the performance of today's XSLT processors is already good enough to meet the application requirements and it is getting better.
- Its syntactic verbosity
- The lack of key features from the functional paradigm,
- The lack of some general purpose programming mechanisms and concepts.

A.V.1.4. Languages Selection.

From the previous paragraphs it can be followed that at the moment XSLT seems to be the best choice to convert XML documents, because:

- DOM was designed to change HTML/XML document presentation to users. It seems to work right in most of the Web Browsers but it does not seem to be used for other more complex conversion.
- Although SAX is an Open Source initiative it is not supported for any standardization entity and it does not seems to be updated since the 2000 year. Moreover it was supported by Saxon a tool which implements XSLT functionalities too.

A.V.2 Some XLST Based Tools for XML Document Conversion.

These paragraphs' objectives are to introduce and compare our choice of a XLST interpreter that we are going to use in order to transform documents using this format converter language.

A.V.2.1 Xalan.

As it was defined in the official web pages the Apache Xalan Project is a collaborative software development project dedicated to providing robust, full-featured, commercial-quality and freely available XSLT support on a wide variety of platforms.

Apache Xalan consists of a set of components that transform XML documents. Where appropriate, these components plug into other XML components using standard APIs (formal, de facto, or proposed).

Although Xalan components are developed both in C++ and Java, we studied the later one as our intention is to use it in a Java environment.

Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types:

- Includes an Interpretive processor for use in a tooling and debugging environment and a Compiling processor (XSLTC) for use in a high performance runtime environment.
- Implements the relevant W3C specifications: XSL Transformations (XSLT) Version 1.0 and XML Path Language (XPath) Version 1.0.
- Implements Java API for XML Processing (JAXP) 1.3 and builds on SAX 2 and DOM level 3.
- Xalan-Java also implements the javax.xml.xpath interface in JAXP 1.3, which provides an object-model neutral API for evaluation of XPath expressions and access to the evaluation environment.
- May be configured to work with any XML parser, such as Xerces-Java, that implements JAXP 1.3 (see Plugging in an XML parser).
- Can process Stream, SAX or DOM input and output to a Stream, SAX or DOM.
- Transformations may be chained (the output of one transformation may be the input for another).
- May be run from the command line for convenient file-to-file transformations.
- Includes an applet wrapper.
- May be used in a servlet to transform XML documents into HTML and serve the results to clients.
- Supports the creation of Java and scripting language extensions and provides a growing library of extension elements and functions.
- Xalan-Java implements the javax.xml.transform interface in Java API for XML Processing (JAXP) 1.3. This interface provides a modular framework and a standard API for performing XML transformations and utilizes system properties to determine which Transformer and which XML parser to use.
- Xalan-Java also builds on SAX 2 and DOM level 3.

A.V.2.2 Saxon.

Saxon is an open source XSLT processor developed by Michael Kay of Saxonica Limited.

The Saxon package is a collection of tools for processing XML documents. The main components are:

- An XSLT 2.0 processor, which can be used from the command line, or invoked from an application, using a supplied API. This can also be used to run XSLT 1.0 stylesheets.
- An XPath 2.0 processor accessible to applications via a supplied API.
- An XQuery 1.0 processor that can be used from the command line, or invoked from an application by use of a supplied API.
- An XML Schema 1.0 processor. This can be used on its own to validate a schema for correctness, or to validate a source document against the definitions in a schema. It is also used to support the schema-aware functionality of the XSLT and XQuery processors. Like the other tools, it can be run from the command line, or invoked from an application.
- On the Java platform, when using XSLT, XPath, or XML schema validation, Saxon offers a choice of APIs. If you need portability across different vendor's tools, you can use the JAXP API for XSLT, XPath and XML Schema processing and the XQJ interface for XQuery. On the other hand, if you want a more integrated and complete API offering access to all Saxon's facilities, the s9api interface is recommended. You can also dive down deeper into the Saxon internals if you need to: there has been no particular attempt to make interfaces private and all public interfaces are documented in the JavaDoc. Clearly, the deeper you go, the greater the risk of interfaces changing in future releases.
- On the .NET platform, Saxon offers an API that enables close integration with other services available from .NET, notably the XML-related classes in the System.Xml namespace. It isn't possible to use Saxon as a transparent plug-in replacement for the System.Xml.Xsl processor, because the API for the Microsoft engine using concrete classes rather than abstract interfaces. However, it is possible to use it as a functional replacement with minor changes to your application code.

In addition, Saxon provides an extensive library of extensions, all implemented in conformance with the XSLT and XQuery Recommendations to ensure that portable style sheets and queries can be written. These include the EXSLT extension libraries **common**, **sets**, **math** and **dates-and-times**. Many of these extensions were pioneered in Saxon and have since become available in other products.

These extension functions are in general accessible from XQuery and XPath as well as XSLT, except where they depend on style sheet information. Many extensions are available in Saxon-PE only and some only in Saxon-EE.

The latest version of Saxon is version 9.2. It is available on Java and .NET. Saxon 9.2, on Java it requires Java 5 (also known as JDK 1.5) or later. Saxon 9.2 on .NET requires .NET framework 2.0 or later. This release brings a significant change to the package structure of the product. Instead of the two versions of the product (Saxon-B and Saxon-SA) produced previously, there are now three editions:

- **Saxon-HE (home edition)** is an open source product available under the Mozilla Public License. It provides implementations of XSLT 2.0, XQuery 1.0 and XPath 2.0 at the basic level of conformance defined by W3C. It is available for both Java and .NET.
- **Saxon-PE (professional edition)** is a commercial product available at modest prices from Saxonica Limited. It adds a number of features to Saxon-HE, including support for Saxon extensions and extensibility mechanisms, support for new features defined in XQuery 1.1 including higher-order functions and easier configuration of features such as localization for different languages and support for external object models such as JDOM, XOM and DOM4J.
- **Saxon-EE (enterprise edition)** is the fully-featured commercial product, essentially a renaming of the previous Saxon-SA to reflect the fact that it now offers much more than just schema-awareness. As well as a fully conformant XSD 1.0 schema processor and support for schema-aware XSLT and XQuery processing, it offers many other features including streaming in XSLT and XQuery, support for XQuery updates and advanced query optimizer, compilation of XQuery code to Java bytecode and much more. For full details, see the Saxonica web site.

Saxon-EE 9.2, the commercial Enterprise Edition from Saxonica Limited, supports XSLT 2.0, XPath 2.0, XQuery 1.0 and XML Schema 1.0, together with early support of draft specifications XQuery Update 1.0, XQuery 1.1, XSD 1.1 and XSLT 2.1. The source code for Saxon-EE is not available.

Saxon-PE 9.2, the commercial Professional Edition from Saxonica Limited, supports XSLT 2.0, XPath 2.0 and XQuery 1.0 together with early support of selected features from the draft specification XQuery 1.1. Source code for Saxon-PE is not available

Saxon-HE 9.2 is the latest open-source implementation of XSLT 2.0 and XPath 2.0 and XQuery 1.0. This provides the "basic" conformance level of these languages: in effect, this provides all the features of the languages except schema-aware processing. This version reflects the syntax of the final XSLT 2.0, XQuery 1.0 and XPath 2.0 Recommendations of 23 January 2007.

There is also a restricted license that works with Saxon-EE but restricts the capability to Schema Validation only; this is referred to below as Saxon-VE (Validation Edition).

The functionality of these variants is summarized in the table below.

Feature	**Saxon HE**	**Saxon PE**	**Saxon EE**	**Saxon VE**
Basic XSLT	yes	yes	yes	yes
Basic XQuery 1.0 including XQJ interface	yes	yes	yes	yes
Schema-Aware XSLT	no	no	yes	no
Schema-Aware XQuery	no	no	yes	no

XQuery Updates 1.0	no	no	yes	no
XQuery 1.1 (subset)	no	yes	yes	no
Schema Validation	no	no	yes	yes
Java Code Generation from XQuery	no	no	yes	no
Separate compilation of XQuery modules	no	no	yes	no
Extensibility using integrated extension functions	yes	yes	yes	yes
Extensibility using reflexive extension functions	no	yes	yes	no
Extensibility using XSLT extension instructions	no	yes	yes	no
Saxon Extension Functions (see Note 1)	no	yes	yes	no
Saxon Extension Instructions in XSLT (see Note 2)	no	yes	yes	no
Advanced Optimizer (see Note 3)	no	no	yes	no
Built-in support for JDOM, XOM and DOM4J (see Note 4)	no	yes	yes	yes
Built-in localization support for date and number formatting (see Note 5)	no	yes	yes	yes
Streaming Extensions for Large Documents	no	no	yes	no
Binary Document Storage (PTree)	no	yes	yes	no

Note 1: Extension functions in the Saxon namespace, together with EXSLT extension functions where available, require Saxon-PE, with a few exceptions such as saxon:stream() that relate to functionality only available in Saxon-EE. Details of these extensions, together with the Saxon edition they require, are in Extension functions

Note 2: This covers the availability of extension instructions in the Saxon namespace (for example, saxon:assign and saxon:while); extension instructions in the SQL namespace (for example, sql:connect and sql:insert; and the basic mechanisms for extending Saxon with new extension elements.

Note 3: This includes a number of optimizations, notable in the handling of joins. Saxon-HE and Saxon-PE always implement joins using a nested-loop strategy, which takes increasingly long as the documents become larger. Saxon-EE, where possible, uses a hash-join algorithm. This applies whether the join is expressed as a path expression, an XQuery FLWOR expression, or a nested set of xsl:for-each instructions in XSLT. In some of the queries in the XMark benchmark, the effect is to reduce query time on a 10Mb source document from 16 seconds to 45 milliseconds.

Note 4: Saxon-PE and Saxon-EE provide support for access to (and in some cases output of) DOM, JDOM, DOM4J and XOM object models "out of the box", as well as allowing user-defined object models to be supported by writing an adapter. Saxon-HE supports DOM "out of the box" and allows user-written adapters; the adapters for JDOM, DOM4J and XOM are available as open-source code which can be compiled and integrated in the same way as a user-written adapter.

Note 5: All editions allow support for languages other than English in xsl:number and format-date() by means of a user-written adapters. Saxon-PE and Saxon-EE include adapters for a number of European languages "out of the box". Currently these adapters are all available as open-source code, which means they can also be used in conjunction with Saxon-HE.