Do-it-Yourself Smart Experiences
ITEA 2 project 08005

Service Ontologies
D3.1

Editor:
Sonia Bilbao, Robotiker-Tecnalia
Yan Tang, VUB STARLab, Belgium

Contributors:
Yan Tang, VUB STARLab, Belgium
Emmanuel Marilly, ALBLF, France
Philippe Dobbelaere, ALU, Belgium

Security: Public
Version: 1.0
Date: June 20, 2011
Number of pages: 60

The DiYSE Project

The Do-it-Yourself Smart Experiences project (DiYSE) aims at enabling ordinary people to easily create, setup and control applications in their smart living environments as well as in the public Internet-of-Things space, allowing them to leverage aware services and smart objects for obtaining highly personalised, social, interactive, flowing experiences at home and in the city.

www.dyse.org

Abstract of this deliverable

This document describes the methods of networked ontologies, ontology evolution, ontology alignment and matching, the applied smart spaces, and the experimentation in the IWT DiYSe project. It records the results from Task 3.1. In particular, the Belgian consortium has worked on ontology creation tasks concerning the meaning evolution support system. And, we have designed and developed an ontology-based component discoverer and recommender in order to show the usage of resultant hybrid ontology. The needs of ontology versioning and alignment are properly addressed in this report.

Document History

Version	Date	Person, Partner	Comment
0.1	14.01.2010	Sonia Bilbao, Robotiker-Tecnalia	Creation of the document
0.2	20.04.2010	Emmanuel Marilly, ALBLF	ToC update + contributions
0.3	**12.05.2010**	**Yan Tang, VUB STARLab**	**Add ontology evolution part – methodology and experiment**
0.4	**18.10.2010**	**Philippe Dobbelaere**	**Added section 3.7**
0.5	**02.11.2010**	**Philippe Dobbelaere, Claudio Forlivesi, ALU-BE**	**Added section 3.6 Physical Layer Onthologies**
0.6	**20.06.2011**	**Emmanuel Marilly, ALBLF**	**Added section 3.6.7.1**
1.0	**23.01.2012**	**Yan Tang, VUB STARLab**	**Final cleanup of document**
1.1	**24.01.2012**	**Yan Tang, VUB STARLab**	**Added section of DIY-CDR**

TABLE OF CONTENTS

1 Introduction

This deliverable records the results from Task 3.1, the objectives of which are the study of the semantics of Smart Spaces and the subsequent creation of dynamic and networked ontologies. In order to expose, describe and reason about the concepts dealt in a given Smart Space and to provide means for the different agents involved to interoperate, it is crucial to have ontologies.

The dynamic nature of the DiY environments allows agents to handle dynamic ontologies and consequently to deal ad hoc with new and changing pieces of information and try to integrate them with their current representation of the world. Hence, the concepts and relations from two agents will need to be mapped, aligned and reconciled. This deliverable records the techniques from fields such as `ontology matching' for aligning and 'ontology evolution' for keeping track of the different versions. Both the manual and automatic evolution of ontologies will benefit from the results of a (semi-) automated meaning-negotiation module, which will be developed in this task and that will serve to find the greatest common divider within different conceptualizations.

In this deliverable, we will show the following results.

- Creation of dynamic and networked ontologies (Section 2)
- Map, align and reconcile ontologies from different resources (Section 2)
- Study the semantics of Smart Spaces (Section 3)
- Use created ontologies for discovering smart components (Section 3).
- Different kinds of semantic web services (Section 4)

2 Dynamic Ontologies Change Management

Flouris et al. (2006) give an overview on the definitions and methods of ontology change. Ontology change occurs when the domain of interests change. It is similar to the term “ontology versioning” since both have the community aspect. New information, previously unknown, classified or otherwise unavailable may become available or different features of the domain may become important (Heflin et al., 1999). Ontology development, by its definition, is a collaborative process. Its sub-products need to be combined to produce the final ontology. Different people or group may have different ideas and understandings of a domain. Hence, ontology needs to be updated gradually in order to meet the requirements from all the stakeholders.

Accordingly, the reasons why ontologies need to change may be the following.
1) The insight of a domain of interests may evolve over time.
2) The community of domain experts is changed.
3) New discovery and concepts in the current domain of interests.
4) “Old” ideas and concepts are no longer valid or démodé.
5) Requirements from “new” agents, software and computer grids.

Ontology change covers several related research areas which are studied separately in the literature. The authors in (Flouris et al., 2006) identify nine such areas, namely ontology mapping, morphism, alignment, articulation, translation, evolution, versioning, integration and merging.

- **Ontology Mapping**: The purpose of which is a heterogeneity resolution and interoperability. The input is two (heterogeneous) ontologies and the output is a mapping between signatures. Its properties are: Output identifies related signature entities
- **Ontology Morphism**: The purpose and inputs are the same as for Ontology Mapping (see above). The output is mappings between signatures and axioms. Its properties are: Output identifies related signature entities and axioms.
- **Ontology Alignment**: The purpose and inputs are the same as for Ontology Mapping (see above).The output is a relation between signatures. Its properties are: Output identifies related signature entities.
- **Ontology Articulation**: The purpose and inputs are the same as for Ontology Mapping (see above). The output is an intermediate ontology and mappings between the signatures of the intermediate ontology and each source. Its properties are: Output is equivalent to a relation identifying related signature entities.
- **Ontology Translation**: It can be further divided into two sub research domains – Ontology translation (first reading) and Ontology Translation (second reading):
  - **Ontology Translation (First Reading):** The purpose of which is to use a different representation language. The input is an ontology and target representation language. The output is an ontology expressed in the target language. Its properties produces an equivalent ontology, if possible.
  - **Ontology Translation (Second Reading):** The purpose of which is the implementation of a signature mapping. The input is an ontology and a mapping. Its output is an ontology. Its properties are: Implements the mapping.
- **Ontology Evolution:** The purpose of which is to apply changes at the domain/conceptualization level. Its input is ontology and change operation(s). Its output is a resultant ontology. The properties are: implements change(s) to the source ontology.
- **Ontology Versioning:** The purpose of which is to transparent access to different versions. Its input is different versions of an ontology and its output is a versioning system. The properties are: version ids identify versions; transparent access to versions; compatibility determination.
- **Ontology Integration:** The purpose of which is to fuse ontologies from similar domains. The input is two ontologies covering similar domains. The output is an ontology. The properties are: fuses knowledge to cover a broader domain.
- **Ontology Merging:** The purpose of which is to fuse ontologies from identical domains. The input is two ontologies covering identical domains and the output is an ontology. The properties of Ontology Merging are: fuses knowledge to describe the domain more accurately

In this section, we will look at the literatures of ontology change, mainly in ontology matching (section 2.1), ontology evolution and versioning (Section 2.2). We do not want to have an exhausted discussion concerning all the listed types of ontology change seeing the needs of focusing. At the end of this section (Section 2.3), we will discuss how we manage ontologies in the DiYSe project based on sections 2.1 and 2.2.

2.1 Ontology Matching

The two ontologies can be different in the following aspects:

- Different representing languages. For instance, one is represented in OWL and the other in UML.
- From two subdomains of a domain of interests.
- From the same domain are developed in parallel and separately by different communities.
- Different versions.
- Designed for different kinds of agents
- Designed for different middle wares.

Normally, the term “ontology matching” is used to refer to an ontology mapping algorithm based on the linguistic properties of terms, using a thesaurus based on WordNet (Flouris et al., 2006).

Although Noy and Klein (2004) have discussed the difference between ontology evolution and schema evolution, it is worthy to use schema matching algorithms for ontology matching. Almost all the popular matching algorithms and tools deal with schema matching. It is because when we talk about ontology, it is more meaningful to deal with the schema instead of the instances.

There are several types of mismatches that can occur between different ontologies.
According to 0we can distinguish between two levels at which mismatches may appear. The first level is the **language or meta-model level**. This is the level of the language primitives that are used to specify an ontology. Mismatches at this level are mismatches between the mechanisms to define classes, relations and so on. The second level is the **ontology or model level**, at which the actual ontology of a domain lives. A mismatch at this level is a difference in the way the domain is modeled.

2.2 Ontology Evolution and Versioning

To support the sharing and reuse of formally represented knowledge among AI systems, it is useful to define the common vocabulary in which shared knowledge is represented. As already discussed in deliverable D1.1 [ref], section 6.1.2, an ontology is a specification of representational vocabulary within a domain.

In DIYSE, the ontology creation methodology involves community-aspect in the ontology capturing processes. Our methodology deals with community-centralized ontology versioning. In this chapter, we will first discuss what ontology versioning/Evolution is, and then we will illustrate the related work. Our work is a continuant research effort on the meaning evolution support systems (DOGMA-MESS, (de moor et al., 2006)). One of our main contributions in DIYSE is a DIYSE specific DOGMA-MESS methodology, which will be illustrated in this chapter. At the end of this chapter, we will illustrate how we will test, implement and experiment in the future within the scope of DIYSE.

2.2.1 What is Ontology Evolution?

First of all, we need to emphasize that ontology version is different from ontology evolution. Both of them belong to the ontology change management. People often tend to be confused with these two terms. According to Flouris et al. (2006), **Ontology evolution** is the process of modifying the ontology when there is a certain need for change or a change in the domain knowledge. **Ontology versioning** is the process of modifying ontology while keeping the original version intact. Ontology versioning is mostly used in CVS systems. We hereby discuss that the definition from Flouris et al. is partically correct. *The process of ontology versioning should be the one kind of ontology evolution processes.* The results of both kinds of processes are different versions of ontologies. It is very rare to see any ontology evolution processes do not support change logs.

Including the above definitions, there are some other definitions concerning ontology version and ontology evolution:

- **Ontology evolution** means modifying or upgrading the ontology when there is a certain need for change or there comes a change in the domain knowledge. (Khattak et al., 2009)
- **Ontology evolution** is the timely adaptation of an ontology to changed business requirements (Stojanovic et al., 2002)

Other types of ontology change management are **ontology merging** and **ontology integration**. The former is composition of new ontology from two or more ontologies covering highly overlapping or identical domains. The latter is composition of a new ontology from two or more ontologies covering related domains (Khattak et al., 2009).

Sometimes, people consider ontology population as a kind of ontology change management, such as in (Khattak et al., 2009). **Ontology population** is defined as introducing new instances to the defined concepts in an ontology. As the layered structure of DOGMA separates instances from concepts, in order to highly achieve **scalability** and **extensibility**, the problem of ontology population is not (and **should not** be) considered to be a problem of ontology change management.

2.2.2 Related Work

Khattak et al. (2009) list a few related work concerning ontology evolution and versioning.

2.2.2.1 User Driven Evolution

The first related work is called **User Driven Evolution** (Stojanovic et al., 2002). It mainly contains four phases of ontology evolution: **change representation**, **semantics of change**, **implementation**, and **propagation**. ()

- **Change representation**. There are three change actions – Add, Delete and Modify. The change action “Add” for different ontological elements is presented as, for instance, Add_Concpet, Add_Domain, Add_Axiom etc. The change action “Delete” is represented as, e.g., Delete_Concept. The action “modifiy”, which has only one representation, is Set_Property_Range.
- **Semantics of Change**. Semantic inconsistency arises when meaning of an ontology entity is changed due to changes performed in the ontology. one change can potentially trigger other changes and so on. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each induced change. The task of ‘semantics of change’ phase is to enable resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology. What Stojanovic et al. (2002) do is to simply make the ontology and the changes as transparent as possible, so that ontology engineers can have a clearer insight of the change.
- **Change implementation**. In this phase, in order to avoid performing undesired changes, before applying a change to the ontology, a list of all implications to the ontology should be generated and presented to the ontology engineers. He should be able to comprehend the list and approve or cancel the change. When the changes are approved, they are performed by successively resolving changes from the list. If changes are cancelled, the ontology should remain intact. This is more elaborated in the description of implementation.
- **Change propagation**. This phase is to propagate the instances of concepts.

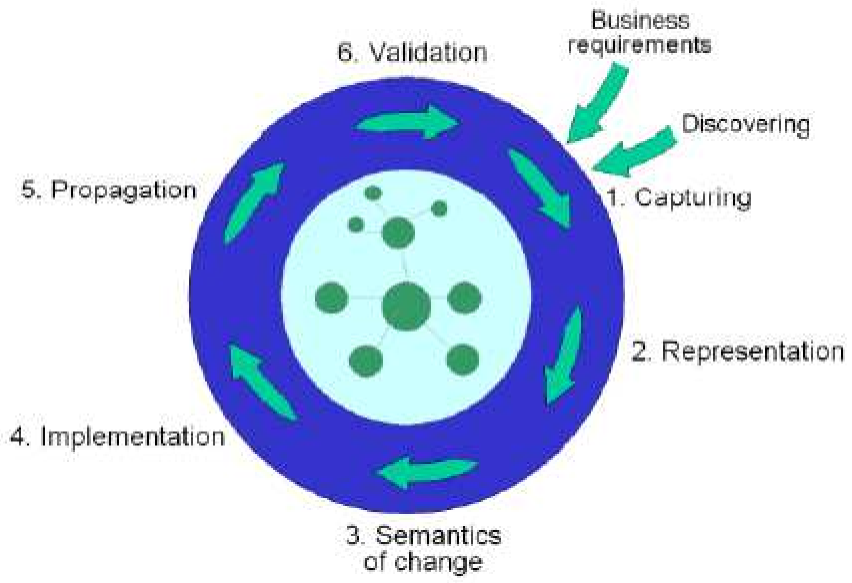


Figure 1: Ontology Evolution Process described in (Stojanovic et al., 2002)

Note that the "user" in this approach does not mean the "user" in our approach. The former is restricted to the ontology engineers, and the latter is the total of ontology engineers, knowledge engineers, domain experts and business/application end users.

Note also that we do not consider instance propagation as one process in the ontology versioning. The reason has been given before the starting of this section.

2.2.2.2 Evolution Framework for Distributed Ontologies and Change and Annotation Ontology

Change and Annotation Ontology (Klein and Noy, 2003; Klein, 2004; Noy et al., 2006) is introduced to represent ontology changes, which cover the tasks of **ontology transformation**[ii], **ontology access**, **update**, **consistent reasoning**, and **verification and approval**.

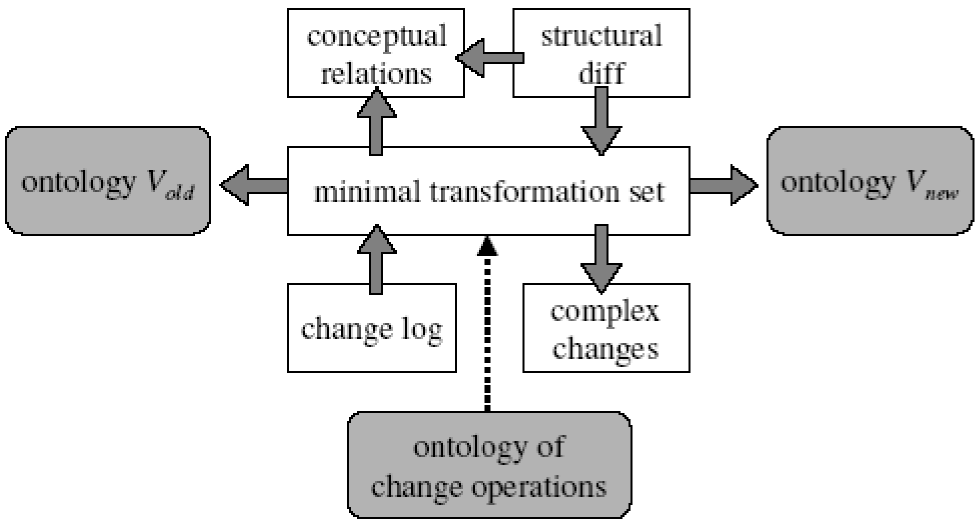


Figure 2: Evolution Framework for Distributed Ontologies for defining ontology change (Klein and Noy, 2003)

shows such an evolution framework, which covers below main items.

- A **structural diff.** is a map of correspondences (or dependences) between conceptual models in the ontology of the current version and the ontology of the next version. It shows a logical declarative view of changes, but not the actual operations to get from one version to another.
- A **minimal transformation set** is a set of operations that are necessary and sufficient to transform an ontology from version *old* to version *new*;
- A **change log** supports to trace the change. In particular, they transform logs into transformation sets by translating the operations into their vocabulary of basic changes and removing redundant changes.
- **Complex changes** are derived from basic changes, e.g., add or remove a concept
- The **conceptual relations**, for instance, property-of, are used to describe the relations between two concepts/entities.

The change log records identification of a changed resource, change author, timestamp and link to annotation about the change. shows the “Change” and “Annotation” classes.

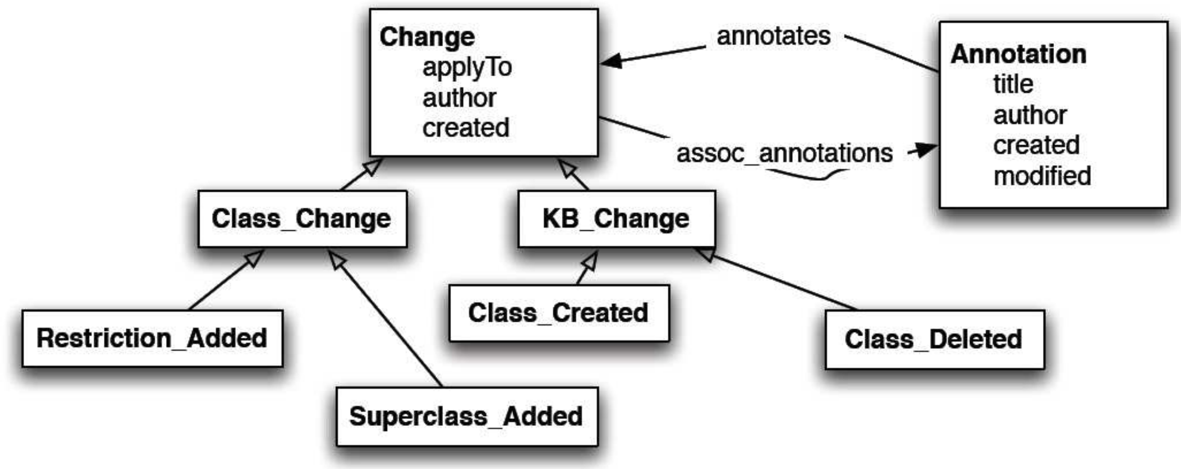


Figure 3: “Change” and “Annotation” classes for the Ontology Evolution Framework (Noy et al., 2006)

It is worthwhile to have a close look at the attributes of collaborative development listed in (Noy et al., 2006). One of these attributes is a set different scenarios:

- **Synchronous** vs. **asynchronous editing**. Ontology engineers works on the same version of ontology in the synchronous editing. In the asynchronous editing, ontology engineers have to always check out the ontology from the subversion.
- **Monitored** vs. **non-monitored**. In monitored editing, From the authors’ point of view (which we think is too narrow), the supporting tool records all the logs and make them available to the other tools. In the non-monitored editing, the supporting tool either discards the logs, or makes them unavailable to other tools.
- Continuous editing vs. periodic editing and Curation vs. no curation. We realize that they are not really relevant to the discussion and a bit too simplified in their scenario descriptions.

In DIYSEMESS, we deal with both synchronous and asynchronous editing. We define monitored and non-monitored scenarios in a different way as follows:

In the **monitored editing**, the key domain expert, with the help of knowledge engineer, is responsible to monitor the actions made by the normal domain experts from different enterprise within the community. He/she can reject or accept their changes. In the non-monitored scenario, everyone is spontaneously updating the ontology without any control of key domain experts. Nevertheless, the systems still does the basic work for checking the consistency of the updated ontology. Therefore, in our settings, “absolute free” solution is not allowed. Otherwise, it leads to a real messy ontology.

2.2.2.3 Discovery Driven Evolution

Castano et al. (2006) illustrate a methodology for ontology evolution by focusing on the specific case of multimedia ontology evolution. The new concept(s) discovery process proposed supports ontology enrichment activity for multimedia ontology. A multimedia object is first fragmented and then new resources are detected from the fragmented parts. Automatic discovery of change(s) using H-MATCH algorithm (a kind of ontology matching techniques) from multimedia objects with additional domain specific metadata is the main achievement. For discovery of new concepts from the fragmented objects, H-Match is used for finding the match for the new concept from the already existing concepts in the ontology. WorldNet thesaurus is used for additional metadata.

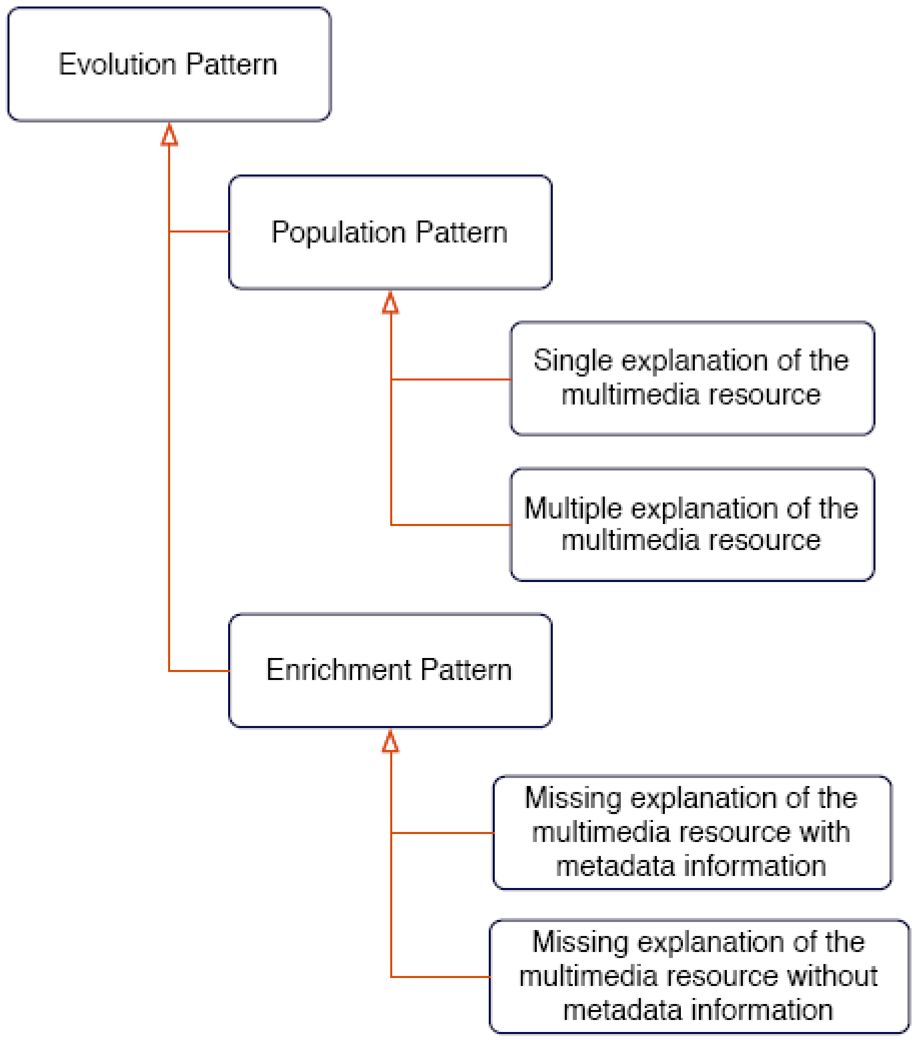

Evolution Pattern
Population Pattern
Single explanation of the multimedia resource
Multiple explanation of the multimedia resource
Enrichment Pattern
Missing explanation of the multimedia resource with metadata information
Missing explanation of the multimedia resource without metadata information

Figure 4 the evolution pattern (Castano et al., 2006))

The evolution pattern used by this approach is illustrated in . The population pattern records new instances of existing concepts in the ontology. The enrichment pattern is used to define new concepts, properties and/or semantic relations in the ontology.

2.2.3 DIYSE Specific DOGMA-MESS

In DIYSE, the ontology creation methodology involves community-aspect in the ontology capturing processes. Our methodology deals with community-centralized ontology evolution and versioning.

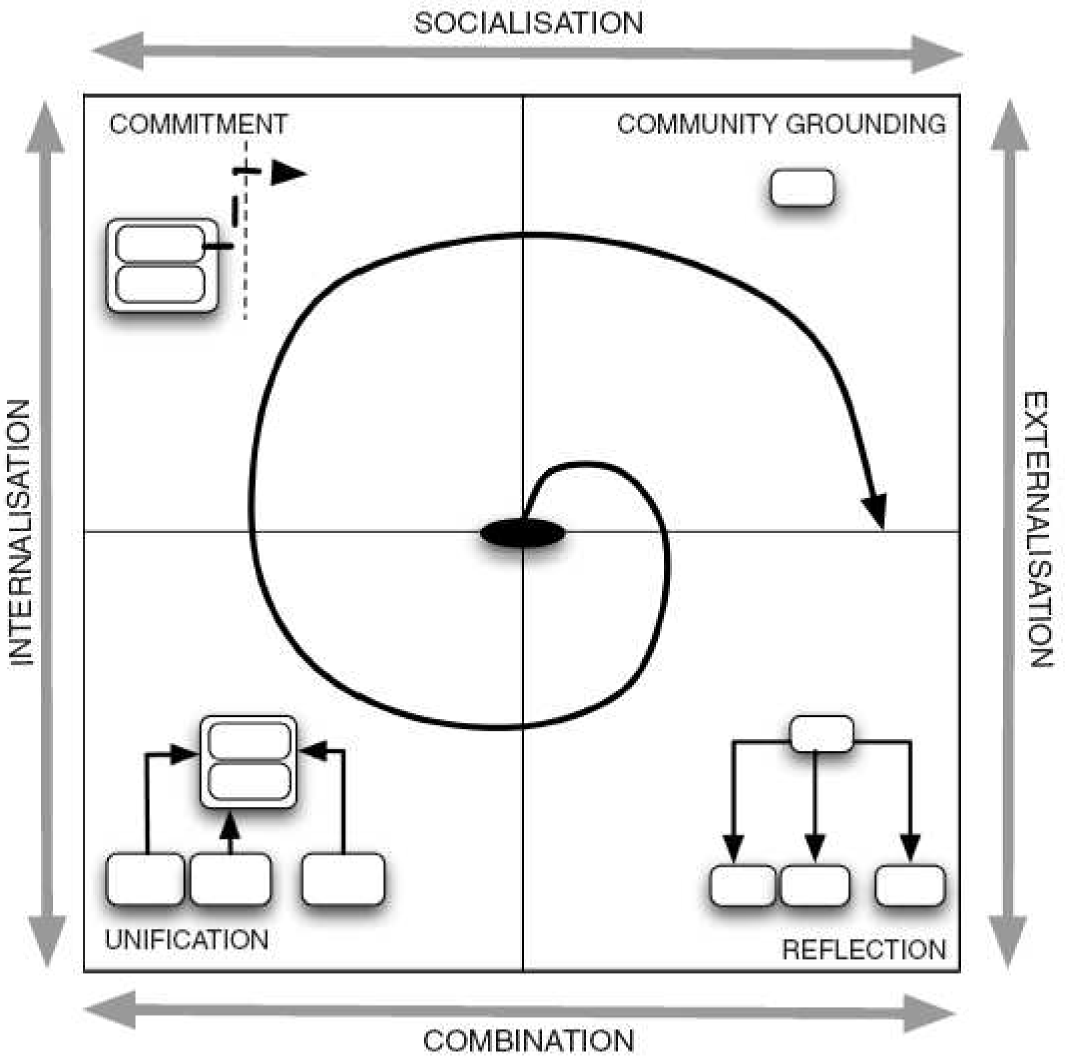


Figure 5: DOGMA-MESS ontology evolution spiral model (De Leenheer and Debruyne, 2008)

DOGMA-MESS (DOGMA – Meaning Evolution Support System) is a methodology to support interorganizational ontology engineering and to assist a community of domain experts to gradually enrich ontologies.

It is a teachable and repeatable collaborative ontology evolution methodology based on the DOGMA framework (Jarrar et al., 2003) using the fact-oriented approaches such as NIAM/ORM (Verheijen and Van Bekkum, 1982).

The DOGMA-MESS community evolution aspect is illustrated in **Figure 5**, based on Nonaka's four modes of knowledge conversion: socialization, externalization, combination and internalization (Nonaka and Takeuchi, 1995). The involved ontology evolution processes, such as community grounding, rendering, alignment, and commitment are inherently driven by the social knowledge conversion modes.

- **Community grounding.** The core domain experts, with the assistance of the knowledge engineer, need to identify the key conceptual patterns that are relevant to be further externalized in the ontology. The result of this activity is a general upper common ontology (UCO).

- **Respective rendering**. All participating stakeholder's domain experts render their models on the UCO, by specializing the conceptual patterns and extending them. This activity allows domain experts to syntactically and semantically nuance their intensions in a natural manner using their own vocabulary. In order to impose UCO reuse, perspective reuse policies can be formalized, such as articulation, specialization and application.
- **Perspective unification**. In the lower common ontology (LCO), a new proposal for the common ontology of the next version is produced, combining relevant materials from the UCO and various stakeholder's perspectives.
- **Perspective version commitment**. The conceptual models in LCO are aligned and combined by the community, which forms the legitimate UCO for the next iteration. All participating organisations finally internalize and commit their instance bases to the new version.

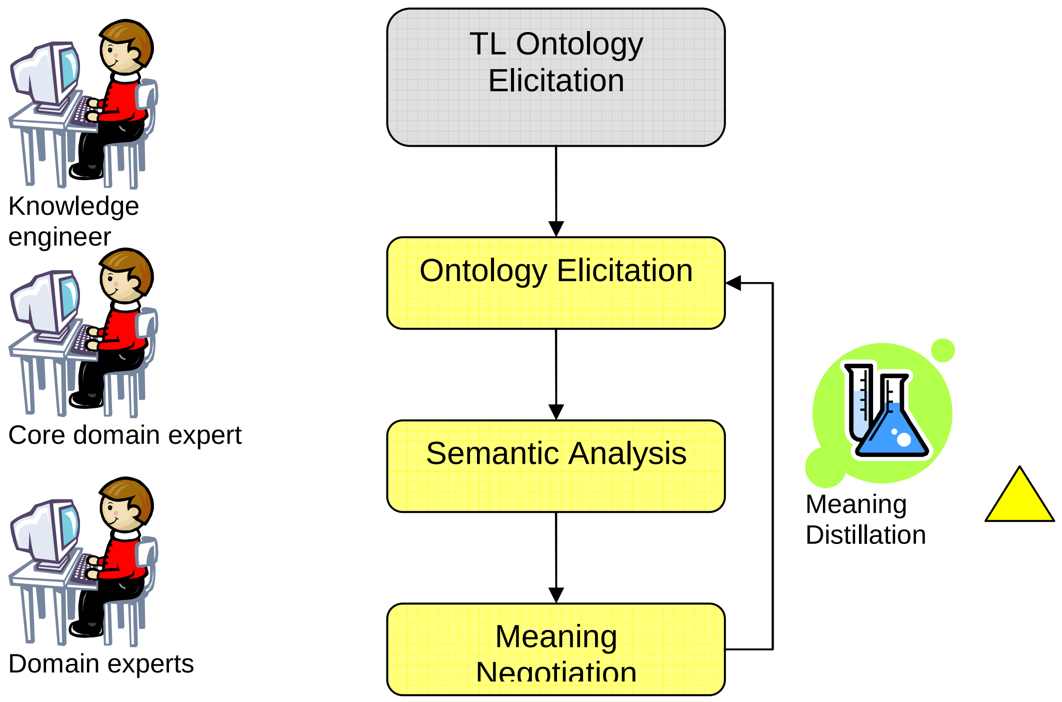


Figure 6: DOGMA-MESS iterative process

Figure 6 shows how a domain ontology is created and evolves among a group of domain experts.

The top level ontology is the ontology containing knowledge of the philosophical foundation of ontology creation. During the common meaning distillation iterations, validated versions of domain ontology are created.

The main focus in DOGMA-MESS is on how to capture relevant commonalities and differences in meaning of concepts. It is a community grounded methodology to address the issues of relevance and efficiency.

The major task of the ontology engineering track is to capture the ontologies that really matter the overlapping interests from different organizations or enterprises. The domain experts need to specify the domain templates that reflect the organizational perspective in their Organizational Ontologies. The domain experts are shielded from complexity issues by performing specific tasks in the elicitation process.

For instance, a template of “Competence” defines that a competence needs to have “Competence level”, “Action” and “Actor”. The “Action” has a certain “Quality”. When a domain expert gets the template, he may specify the “Competence” as “Speech clarity”, the “Competence level” as “Very good”, the “Action” as “Speak” and “Understand”, and the “Actor” as “Person actor”.

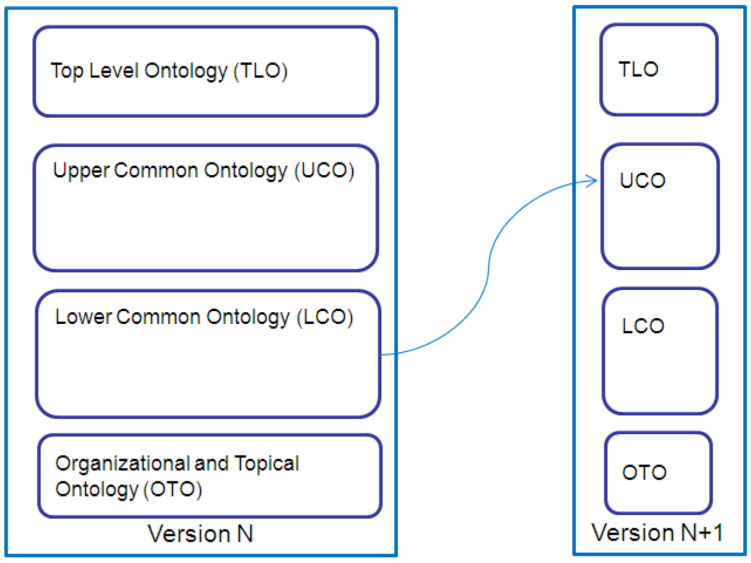

Top Level Ontology (TLO)
Upper Common Ontology (UCO)
Lower Common Ontology (LCO)
Organizational and Topical Ontology (OTO)
Version N
TLO
UCO
LCO
OTO
Version N+1

Figure 7: DOGMA-MESS evolutionary process

Figure 7 shows an overview of how an ontology evolves.

Within each version, four levels of ontologies need to be distinguished:

- *Organizational Ontology* and *Topical Ontology* (OTO) seeks to systematically represent the knowledge structure the individual domain experts have on given *themes* (or *tasks*). A Topical Ontology lays foundation for application (or task) specific ontologies and conceptual models. Its semantic space covers multiple subjects and dynamic evolution of the core concepts within a topic. Concepts within a topic represent terminology of an application's structure, assumption and framework. Within a version, every domain expert (or every enterprise-wise stakeholder group) is responsible to build his own OTO based on the ontology models in the UCO. For example, we have a Conceptual Graph model that describes the definition of "Teacher" and "Course" at the UCO level. Based on it, a domain expert may introduce a new relevant concept "Patience" by adding a conceptual relation "has skill" to "Teacher". Similarly, a new relevant concept "Oral comprehension" can be introduced at the OTO level.
- *Lower Common Ontology* (LCO) constitutes the most important and complex layer for DOGMA-MESS. The concept definitions at this level are the organizational and topical specializations, which are created at the Organizational Ontology or Topical Ontology (OTO) level after they are aligned[iii] and merged[iv]. This process happens within a version. The candidate concepts are analyzed by an SDT, which contains the decisions of whether a concept can be lifted (or merged) to the LCO or not. The concepts, which are not lifted, will keep in the OTO and wait for the next iteration of versioning process. In the next section, a concrete SDT example with a lifting rule will be demonstrated.
- Each domain has its own *Upper Common Ontology* (UCO); the *Upper Common Concept Type Hierarchy* organizes the (evolving) concept types that are common to the domain. For example, 'Actor' at the TLO level can be translated into the concepts 'employer' and 'employee' at the UCO level of the Human Resource Management (HRM) domain. Domain experts define domain canonical relations in terms of the domain. For example, the domain canonical relation 'hire' can be applied between 'employer' and 'employee'. When an ontology evolves, all the concepts in the UCO of *version N* are lifted to the concepts in the UCO of *version N+1*. The *core domain experts* are responsible to standardize the concept definitions at this level.
- *Top Level Ontology* (TLO) defines the abstract concept types, such as 'Actor', 'Object', 'Process' and 'Quality'. Conceptualization at this level is not allowed to be changed. The relations between these concept types fall into two categories: 1) the hierarchical relations, which are also called *subsumption* ontological relations. 2) Other Core Canonical Relations, such as "part-of" *merelogical* relation, "property-of" relation and "equivalent" relation.

2.2.3.1 Involved Roles in DIYSEMESS

DOGMA-MESS (Figure 6) supports users playing three main roles: the **domain experts**, the **core domain experts** and the **knowledge engineer**. In addition, we add the roles of **non-technical end user**s (e.g. normal grandmother, a house wife) and **semi-professional end user**s (e.g. geeks and nerds) in DIYSEMESS.

The domain experts are professionals in a particular domain. The core domain experts have thorough expertise in the cross interests between different enterprises. The knowledge engineer, who has excellent expertise in representing and analyzing formal semantics, is responsible to assist the domain experts and core domain experts in the processes of ontology creation, validation and evolution.

When we speak about different roles in DIYSEMESS, it is important to also look at the term "ontology" viewed by these different kinds of users.

Nodine and Fowler (2005) illustrate the conflicting goals of different kinds of users by clarifying their goals in the ontology creation processes.

The goal of **ontology designers** (they are ontology engineers and knowledge engineers) is: working towards maximizing the (re-)**usefulness** of his ontology to a wide variety of applications, is to completely characterize a particular domain at the semantic level. The ontology designer needs expertise in knowledge representation and in the domain of the ontology. His intent is to develop a comprehensive and up-to-date ontology, with a broad set of acceptable terms. His job is hampered by the fact that the different domain experts have different viewpoints of the domain, and these viewpoints must be reconciled within the ontology itself, placing pressure on the designer either to take a commonly agreed-upon but consistent subset of the domain, or to attempt to placate everyone by including everything. The likely result is either the ontology will express concepts at a higher level than can be used effectively by an application that seeks to attach suitable labels to real data, or the terminology within the ontology will not be logically consistent and thus will not be easy to incorporate into an application.

The goal of the **application user** (the end user of the final ontology-embedded DIYSE system – non technical end users and semi-professional end users) is to be able to do his job well and easily. Application users have expertise in their own jobs and potentially in the domain of the application, but may have minimal understanding of knowledge representation or application design issues. Typical users want an understandable, natural, easy-to-use interface to the application that facilitates their work. This implies that they want the scope of the ontology restricted to the exact area within the domain that they are specifically concerned with. Another issue is that a user may have a comfortable vocabulary of domain-related terms that do not map well to the ontology representation model, to the domain model that the ontology developer had in mind or to the needs of the application itself.

The goal of the **application designer** (e.g. the programmers who are developing the DIYSE platform) is to use the ontology to support an application. The application designer has expertise in building applications, especially within given domains. His intent is to develop an efficient and useful application. It is likely that the application will not cover the whole domain; conversely, it may span multiple domains. Also, application designers need to focus on some issues that are unimportant to the ontology designer. For instance, a failure to take sufficient care of value representation issues in the ontology may force the application designer to code this information directly into the application, even though it is really domain-related knowledge.

In DIYSE, the application designers are the ontology designers. Therefore, the boundary between them is blurred. In particular, the evolutionary needs of different DIYSE ontologies are different:

- Concerning the DIYSE **sensor ontologies**. The upper ontology, by definition, does not change. The common ontology is almost as small as the upper ontology and the lower common ontologies are large. It is because the lower common ontologies are sensor-brand bounded, which have very *few* overlapping parts with others. The evolutionary needs for the sensor ontologies are evoked at the lower common ontology level, but seldom at the upper common ontology level.

- Concerning the DIYSE **ontologies of components for semantic searching**. The evolutionary needs are high due to the fact that new solutions will be constantly introduced by the community.

There are currently two kinds of ontologies in DIYSE as discussed above. There might be some other ontologies in DIYSE, which we will come back to the same issue in the future deliverables.

2.2.3.2 Critical Changes in DIYSEMESS

In this section, we will discuss two main issues: critical changes in the environment and critical changes in an ontology/ontologies.

2.2.3.2.1 Critical Changes in the environment

We identify the following changes as the critical changes in the environment.

- **Changes in the community of stakeholders**. When new stakeholders come in or old stakeholders leave the community, the following problems may occur:
  - **Change of focus.** New people bring in new focus, especially when they introduce new problems that need to be solved by their ontology-based applications.
  - **Shift of responsibility for executing community activities.** The new stakeholders need to participate in the ontology creation processes. Community profiles need to be updated and new participants' profiles are inserted. In a community, we specify roles into domain experts and core domain experts. The change in the resulting ontology will be obvious when a core domain expert leaves. More specifically, we can further assign community management roles to domain experts by assigning privacy controlling tasks to different experts. When a responsible person leaves the community, the change will be obvious. In addition, consistency is necessary for executing or delegating the community activities
- **Changes in the goal and problem scope**. A vision is a compelling and inspiring view of a desired and possible usage of ontology. Initially, the stakeholders in a community develop a shared vision and common values, which have been drawn on a high executive level as the goal and the scope of the domain of the ontologies (Spyns et al., 2008). It shows the expectations from the community and provides a big picture of the long-range results of the planned process and the future accomplishments when the methodology put forward are actually implemented. It is clear that to change the goal and the scope will affect the final ontology types. In practice, during one project period, we do not recommend to have any critical changes in the goal and scope in order to avoid unnecessary time cost.
- **Changes in the resources**. By including irrelevant material in a corpus to be processed automatically, there is a greater chance that irrelevant but formally correct output will be generated. Therefore, it is important to address the usefulness of received documents, models and textual descriptions. As proposed by Spyns et al. (2008), when new resources are introduced, we need to take the following steps to weed out superfluous materials.
  - Separate the document into "core text" and "explanatory text"
  - Separate out definitions
  - Separate out paragraphs that are clearly not relevant to any potential application
  - From the "core text", try to pick out any parts which might possibly put a constraint or obligation on an application without having any particular application in mind.
  - Find exceptions to these constraints. The exceptions should be derived from the text (or possibly from the experts' knowledge)

In addition, we add two more activities as below

  - Separate the documents provided by the core domain experts from the "normal" domain experts
  - Separate the standards from the normal textual descriptions.

- **Changes in the expertise level of different roles**. When domain experts start to create their own ontological models, a few who learn quickly will be promoted to core domain expert level when they capture an excellent view of domain.

2.2.3.2.2 Critical Changes in an Ontology/ontologies

As discussed in (Castalo et al., 2006; Klein, 2004; Khattak et al., 2009), the critical changes while most of these changes are as below:

- **New Concept**: This is the most common change in any ontology. New concepts emerge and have to be accommodated in the concept hierarchy.
- **Concept Hierarchy**: In this case the concept in focus might have different hierarchical position to the existing one.
- **Concept with Changed Properties**: When the concept in focus is already present in the ontology but its properties are different from the existing one.
- **Concept with Changed Restrictions**: In this case, the concept in focus having restrictions that are dissimilar from those associated with existing concepts.
- **Simple vs. Aggregated Concept**: The concept in focus might be a combination of multiple existing concepts (or vice versa). The ontology evolution framework(s) shall properly detect and act accordingly to accommodate these types of changes.
- **Concept vs. Property**: The concept can either be a class in OWL or used as a property of some other existing class. For example, the concept deliverable could be a separate class or could be modeled as property of the concept project. In the first case it could have been implemented as a subclass of document and in the second case it could take the instances of software as its value.

In addition, we discuss that other critical changes in an ontology is as follows:

- **New/updated domain canonical relations**. The domain canonical relations, as defined in the previous subsections, are the relations that are common to a specific domain. For instance, in the domain of "university", the common ontological relations (such as "is-a", "part-of") and domain specific ontological relations (e.g. "teach" and "study") are considered as the domain canonical relations. These relations are defined before each iteration of an ontology evolution process. When ontology evolves, new domain canonical relations are constantly discovered. The core ontology engineers need to carefully specify the meta-model (meta-relations) of these relations. For example, duplicated relations (e.g. "teach" and "give course") have to be mapped and/or aggregated properly.
- **New/updated constraints**. When new constraints are introduced by different ontology engineers from the same community, these constraints can be conflict with each other. When the conflict happens at the level of individual ontology engineer from the same enterprise, we need to run the meaning negotiation process to solve it. When it happens at the whole community level, we keep the conflict constraints from different enterprises. We don't solve it because different enterprises have different kinds of applications, they make use of lexons by committing their own point of views through the ontological commitment layer. It is, from our point of view, the beauty of separating lexons (binary fact types) from the actual usage.
- **Uneven concept granularity**. This problem has been addressed by Nodine and Fowler (2005) as "… some ontologies have been developed with a focus on one particular aspect of a target domain. This aspect was well-developed and well-tested, but other aspects of the same domain was ignored. As a result, merging two such ontologies is likely to lead to differences in granularity between terms of the two." It results in situation where same physical sources of data have ontology engineers and application designers with different perspectives. Some people oversimplify the characterization of the important viewpoints of the others.
- **Updated domain dictionary** or **terminology**. This occurs when different roles use the same term for different concepts, or use different terms for same concepts. The first situation can also be caused by the fact that different ontology engineers use existing concepts defined in the ontology with their own intensions (or expectation) of the conceptual meanings. The second situation occurs when, for instance, different enterprises have different locally defined domain dictionary. We often use mapping commitments for the second situation; while for the first situation, it is often not very obvious (because people are not aware of this subtle problem) and it requires negotiation processes. Hence in practice, the solution for the first situation is often more complicated than the second one.
- **Application mismatch.** According to (Nodine and Fowler, 2005), this problem is caused by the fact that application designers need to update the code when an ontology evolves. We argue that the view point of the authors is limited to the application ontology. In their problem settings, the ontology defines the structure of functional applications. The application designer

needs to implement the ontology at the end. It is certain that the cost is relevantly high. In DIYSE, the sensor ontology and other ambient ontology based applications have the same problem. When ontology changes, our technicians need to “re-implement” the whole environment. When an ontology is used as domain dictionary, vocabulary or a manual for the content management systems, this problem does not exist any more. In the DIYSE ontology-based searching engine, the ontology is used as a knowledge base instead of an ontological structural design of an application, Hence the problem of application mismatch does not exist in the search engine.

2.2.3.3 Experimental results: bathroom ontology

In order to demonstrate how DIYSEMESS is executed in DIYSE, we have tried to model a bathroom ontology, which has been created during the ontology preparation meeting on May 6th, 2010. The community includes Yan Tang, Johan Criel, Laurance Claeys, Femke Ongenae and An Jacobs.

We started with two questions (scoping the domain):

1. Please list the reasons why you need to define “bath” (your answer should not be “because I have to”)
2. Would you please give/visualize an application that will use this ontology/definition? (in graph and/or text)

These two questions need to be answered before creating our ontology. If the answers are negative, then we should not create the ontology. We get the answers as below.

Table 1: domain scoping

Question ID	Answer(s)
Question 1	1. because it’s required by the project (not a good answer) 2. (use the standard textual in the SoA deliverable – the usage of ontology)
Question 2	1. Want to build a new bathroom (An) 2. use ICT to make bathroom smarter (Femke) 3. IKEA wants to build a perfect bathroom (Johan) 4. to improve care process and communication of what care to give (An) 5. to have context model, search components, rich description, measure (e.g. door open, door closed) (Johan) 6. to have common agreement of bathroom, to enhance shareability between people from different departments. (Laurance) 7. to help and make sure everyone has the same understanding of correct sequence of taking bath processes. (Johan)

Then we collect textual description of “bathroom” as illustrated in **Table 2**. The core domain expert asks every stakeholder to write down a piece of text of “bathroom” independently and send it back. The core domain expert analyzes these texts in order to give the advices to the stakeholders for better scoping. Irrelevant materials should be rejected. With these texts, the community can start creating ontologies.

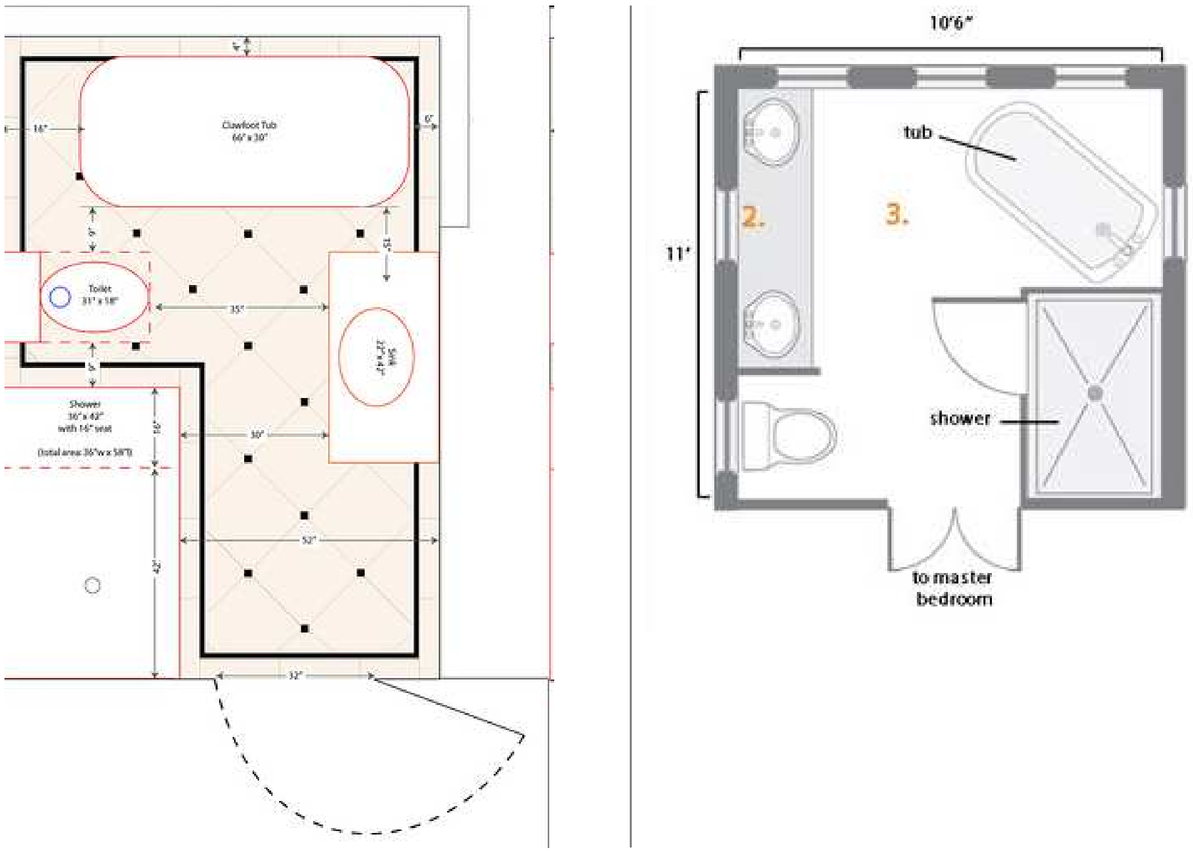
Table 2: resources to model a bathroom

ID	Description
1	A bathroom is a room that may have different functions depending on the culturalist context. In the most literal sense, the word bathroom means "a room with a bath". Because the traditional bathtubs have partly made way for modern showers, including steam showers, the more general definition is "a room where one bathes". There can be just a shower, just a bathtub or both; and often both plumbing fixtures are combined in the bathtub. The room may also

	contain a sink, often called a "wash basin" or "hand basin" (in parts of the USA) and often a "lavatory". In the United States, "bathroom" commonly means "a room containing a lavatory". In other countries this is usually called the "toilet" or alternatively "water closet" (WC), lavatory or "loo". The word "bathroom" is also used in the U.S. for a public toilet (the more formal U.S. term being "restroom"). In the United States, bathrooms are generally categorized as a "full bathroom" (or "full bath"), containing four plumbing fixtures: bathtub, shower, toilet, and sink; "half (1/2) bath" (or "powder room") containing just a toilet and sink; and "3/4 bath" containing toilet, sink, and shower, although the terms vary from market to market. In some U.S. markets, a toilet, sink, and shower are considered a "full bath". This lack of a single, universal definition commonly results in discrepancies between advertised and actual number of baths in real estate listings. An additional complication is that there are currently two ways of notating the number of bathrooms in a dwelling. One method is to count a half bathroom as ".5" and then add this to the number of full bathrooms (e.g., "2.5" baths would mean 2 full baths and 1 half bath). The other, newer method is to put the number of full bathrooms to the left side of the decimal point and to put the number of half bathrooms to the right of the decimal point (e.g., "2.1" would mean 2 full baths and 1 half bath; "3.2" would mean 3 full baths and 2 half baths). (from Wikipedia)
2	A bathroom is a room to bath. It is mostly a seperate room in a house or appartement. When it isn't a seperate room, it doesn't have a door, but is directly connected to another room of the house. A bathroom contains of a bath, a seperate shower and two lavabos. Sometimes a bathroom also contains a toilet. There is also specific furniture available in a bathroom, a towelrack, an anti-slip matje to help you not to fall when you come out of your bath, a mirror and a toiletpaper holder. The mirror hangs above the sink because people should be able to look at their face while e.g. combing their hair or shaving. The comb is then on a shelf below the mirror, but above the sink. The sinks, shower and bath also have a tap. The taps usually have one. handle and by turning it to one or the other direction you can get hot or cold or in between temperature water. The bathroom contains also a heater because people like to have warm when they come out of their bath. A bathroom also sometimes has a ventilation system.The walls of a bathroom are often built in moisture resistant material such as tiles. (Laurence's personal view)
3	A bathroom contains a bath and/or a shower or a combination of the two. In the last case you mostly have a splash screen around your bath. In at bathroom you find a lot of water taps (bath, shower, lavabo), mostly one for cold and warm water (at least if it is open for a while). You use the bathroom to get clean. Because you are most of the time quite naked in the bathroom it is mostly one of the warmest rooms in the house. In hotels however you have also public bathrooms. There you have to take all your materials to that place, other ways you can find it in the cupboards around it (Johan's personal point of view)
4	Help taking a bath. Start preparing the bathroom a quarter of an hour before you start taking a bath. Switch the heater on, place the antislip bathing mat in the bath, and optionally place a plastic duck in the tub. Then open the hot and the cold water tap to start filling the bath and adjust it to the temperature you want. Let the water flow for a minute and feel again if the temperature is still ok. Adjust again if needed. Put some bathingsoap just under the straal of the tap so you have some nice bubles. Switch the light off and go get the person you will join in his/her bath taking moment. Help undress the person till s/he only wears his/her underwear (in the sleeping room, but can also be in the bathroom), then go the bathroom, feel again the temperature of the water, help

	undress the remaining cloths. Take the persons his/her right arm, while the person can use his left arm for holding him/herself. Make sure the person can not slip away in the bath before leaving the bathroom. Make sure the person is not longer then 20 minutes in the bath. When the person wants to get out of the bath, take a towel, …. (An’s personal point of view)
5	A bathroom is a recent room in a house. The purpose of the room is to be able to wash your body with water using soap to clean the body. Therefore there is at least à Sink with à tab linked to pluming to fresh water and let dirty water leave this room. Towels are in room to dry body à heating system is added in today’s bathrooms as well as Choice for hot And cold water coming from à system inside or outside bathroom Other appliances to wash yourself are à bath and or shower linked to water in same way. Next to tabs there are fixed or moveable showerheads Bath and sink can be stopped with à closing device to keep water in The walls lighting and floor have to be able to withstand high humidity. So tiles are mostly used to finish high wet points. Tadelac tropical wood are alternatives for stone and ceramic The dominant color is white because gives association of hygiene. Sometimes à toilet is in bathroom as well other things kept in bathroom are medication often in separate cabinet with key or higher height to keep out of reach of children A basket for dirty laundry Cosmetics of all kind to give care to different parts of the bedroom. (Femke’s personal point of view)

The texts include online documents and personal views. In theory, it is also possible to include graphs and drawings (**Figure 8**). However, we discuss that this kind of drawings can have different interpretations by different people and it is difficult to scope down the problem while brainstorming them because it lead to open-world semantics.



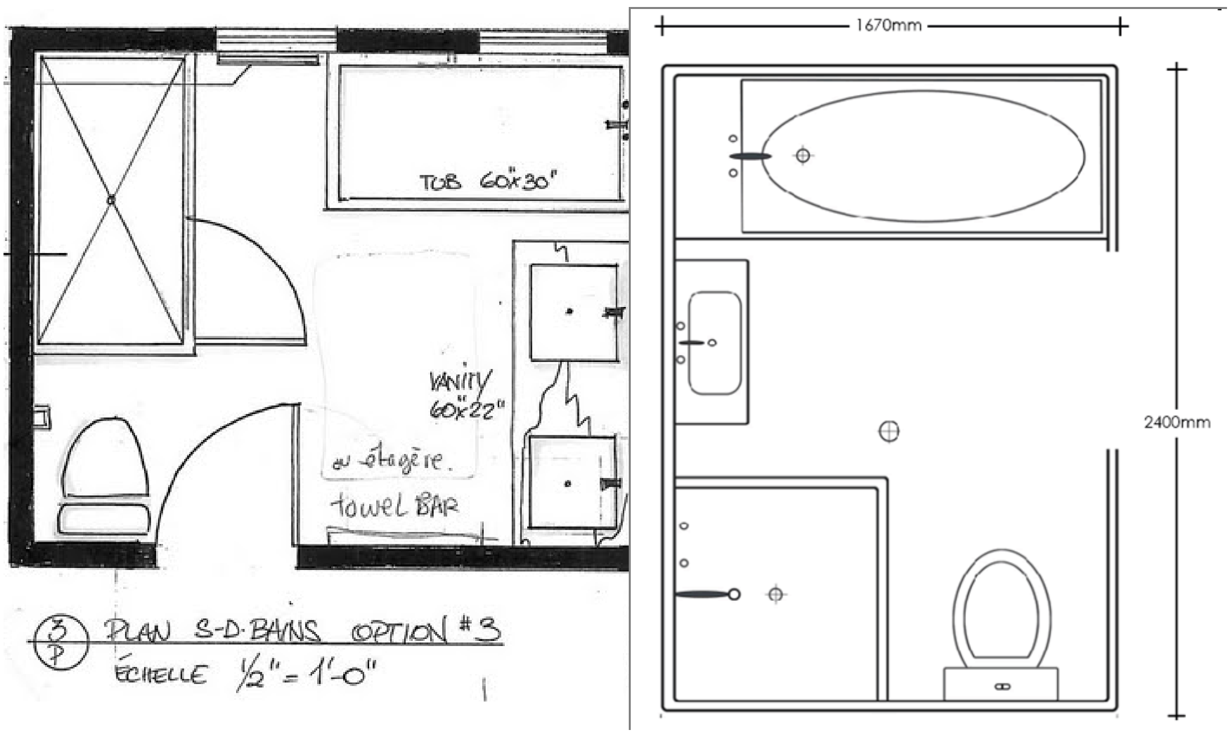


Figure 8 graphical designs of "bathroom" (provided by Laurence)

After the texts are collected, similar texts are combined and the stakeholders who have similar ideas are grouped together. In order to save time, we did not combine similar texts. Instead, we ask each group to select a "good" text.

Then, we ask each group to highlight the most "important" textual descriptions that they think are useful for the definitions of "bathroom". It can be a sentence, a phrase, a term, a verb or any piece of information. For instance, **Figure 9** shows an example of highlighting.

Group	**Highlighting results**
Group 1	A bathroom contains a bath and/or a shower or a combination of the two. In the last case you mostly have a splash screen around your bath. In at bathroom you find a lot of water taps (bath, shower, lavabo), mostly one for cold and warm water (at least if it is open for a while). You use the bathroom to get clean. Because you are most of the time quite naked in the bathroom it is mostly one of the warmest rooms in the house. In hotels however you have also public bathrooms. There you have to take all your materials to that place, other ways you can find it in the cupboards around it
Group 2	A bathroom is a room to bath. It is mostly a seperate room in a house or apartement. When it isn't a seperate room, it doesn't have a door, but is directly connected to another room of the house. A bathroom contains of a bath, a seperate shower and two lavabos. Sometimes a bathroom also contains a toilet. There is also specific furniture available in a bathroom, a towelrack, an anti-slip matje to help you not to fall when you come out of your bath, a mirror and a toiletpaper holder. The mirror hangs above the sink because people should be able to look at their face while e.g. combing their hair or shaving. The comb is then on a shelf below the mirror, but above the sink. The sinks, shower and bath also have a tap. The taps usually have one. handle and by turning it to one or the other direction you can get hot or cold or in between temperature water. The bathroom contains also a heater because people like to have warm when they come out of their bath. A bathroom also

	sometimes has a ventilation system.The walls of a bathroom are often built in moisture resistant material such as tiles.
Group 3	A bathroom is a room to bath. It is mostly a seperate room in a house or appartement. When it isn't a seperate room, it doesn't have a door, but is directly connected to another room of the house.A bathroom contains of a bath, a seperate shower and two lavabos. Sometimes a bathroom also contains a toilet.There is also specific furniture available in a bathroom, a towelrack, an anti-slip matje to help you not to fall when you come out of your bath, a mirror and a toiletpaper holder. The mirror hangs above the sink because people should be able to look at their face while e.g. combing their hair or shaving. The comb is then on a shelf below the mirror, but above the sink. The sinks, shower and bath also have a tap. The taps usually have one. handle and by turning it to one or the other direction you can get hot or cold or in between temperature water. The bathroom contains also a heater because people like to have warm when they come out of their bath. A bathroom also sometimes has a ventilation system.The walls of a bathroom are often built in moisture resistant material such as tiles. A bathroom is a room for bathing

Figure 9: an example of highlighting

After every group has highlighted their texts, they're asked to rearrange the highlighted sentences into subject-verb-object and/or subject-verb style. For instance, "people smoke cigarette", or "person smokes". This step is the preparation phase of having lexons.

Table 3: subject-verb-object information provided by domain experts who are not ontology engineer

Subject	Verb	Object
A bathroom	is	a room to bath.
A bathroom	is	a room in a house or appartement
A bathroom	Is	a separate room
A bathroom	Has	a door
A bathroom	contains	a bath, a seperate shower and two lavabos.
Sometimes a bathroom	contains	a toilet
specific furniture	Is available	In a bathroom
a towelrack, an anti-slip matje, a mirror and a toiletpaper holder	are available	In a bathroom
an anti-slip matje	helps	not to fall
The mirror	hangs	above the sink
The comb	is	on the shelf
The shelf	is	above the sink and under the mirror
The sinks, shower and bath also	have	a tap
The taps and	have	one handle
by turning the handle you	can get	hot or cold or in between temperature water
The bathroom	contains	a heater
People	like	to have warm when in the bathroom
A bathroom also sometimes	Has	a ventilation system.
The walls of a bathroom	are built	built in moisture resistant material such as tiles

Table 3 shows an example of subject-verb-object style complied information provided by a domain expert who is not an ontology engineer. The ontology engineer should help these domain experts to further segment these phrases into atomic binary facts – the lexons, using domain canonical relations.

We collect the domain canonical relations listed as follows:

- **Is-a subsumption relationship**. It can be indicated by one of the relations in the set {is-a, is, subclass, superclass, subtype, supertype, parent of, child of}

- **Part-of mereological relationship**. It can be indicated by one of the relations {has, has part of, is part of, contains, part-of, part of, element of, substitute of, characteristic of}
- **Instrumental relationship**. It can be one in the relations set {is used for, uses, is a tool of, is a means of}
- **Action relationship**. It can be one in the relations set {acts on, executes, functions}.
- **Location relationship**. It can be one in {is connected to, located at, next to, adhere to}.

The ontology engineers help the domain experts to extract lexons from their subject-verb-object style complied phrases using the above relations. Below changes are made.

- Remove articles, e.g. a, an
- Segment complex subject
- Change plurals into singulars
- Remove unnecessary adjectives
- Remove adverbs
- Separate conjunctions
- Separate propositions
- Replace the instances with concepts, e.g. Jason Langridge -> person.

We call the above steps as "lexon engineering". **Table 4** shows an example of how to create lexons from the subject-verb-object information. **Table 5** contains more lexons.

Table 4: results of lexon engineering

Subject	**Verb**	**Object**
A bathroom	is	a room to bath.
Lexons		
Bathroom	Is-a	Room
Bathroom	Is used for	Bathing
A bathroom	is	a room in a house or appartement
Lexons		
Bathroom	Is part of	House
Bathroom	Is part of	Apartment
A bathroom	Has	a door
A bathroom	contains	a bath, a seperate shower and two lavabos.
Sometimes a bathroom	contains	a toilet
Lexons		
Bathroom	Has	Door
Bathroom	Has	Bath
Bathroom	Has	Shower
Bathroom	Has	Lavabo
…	.,..	…

During this activity, new relations will be discovered. In this case, the ontology engineers have to either group them to the existing relations sets, or propose to add them into the domain canonical relations sets for the ontology of the next version.

Table 5: lexons extracted based on Table 3.

Head term	Role	Tail term
Bathroom	Is-a	Room
Bathroom	Is used for	Bathing
Room	Is part of	House
Bathroom	Has	Door
Bathroom	Is connected to	Room
Bathroom	Has	Toilet
Bathroom	Has	Bath
Bathroom	Has	Shower

Bathroom	Has	Lavabo

Note that the constraints at the commitment layer were constantly invoked by the domain experts. When the ontology engineer gets the requests, he should note these constraints down when making the lexons. However, these constraints need to be already recorded in the textual information provided by the group, otherwise, they're required to provide extra information in texts and start another iteration of the methodology.

The ontology engineers specify these constraints into ORM models as illustrated in **Figure 10: An ORM model based on Table 5**. The subtyping constraint is specified for "is-a" relationship.

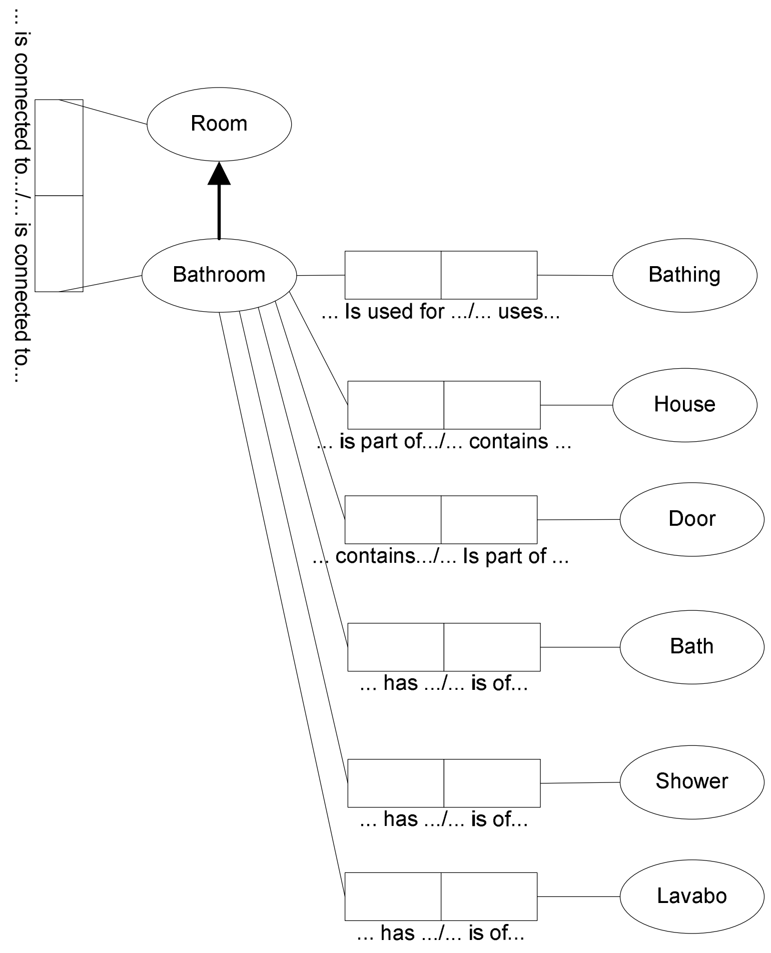

... is connected to.../... is connected to...
Room
Bathroom
Bathing
... Is used for .../... uses...
House
... is part of.../... contains ...
Door
... contains.../... Is part of ...
Bath
... has .../... is of...
Shower
... has .../... is of...
Lavabo
... has .../... is of...

Figure 10: An ORM model based on Table 5

Figure 11 shows an example of how to convert constraints in textual annotation into an understandable pseudo natural language. For instance, in the text (see **Table 2** and **Table 3**), one says that "a bathroom contains a bath, a shower and two lavabos". The ontology engineer converts it into "EACH bathroom contains ONE bath, ONE shower and TWO lavabos." When he passes it to the domain expert, the domain expert should accept it, update it or reject it with reasons. In the last situation, the rejection reasons need to be written down in texts.

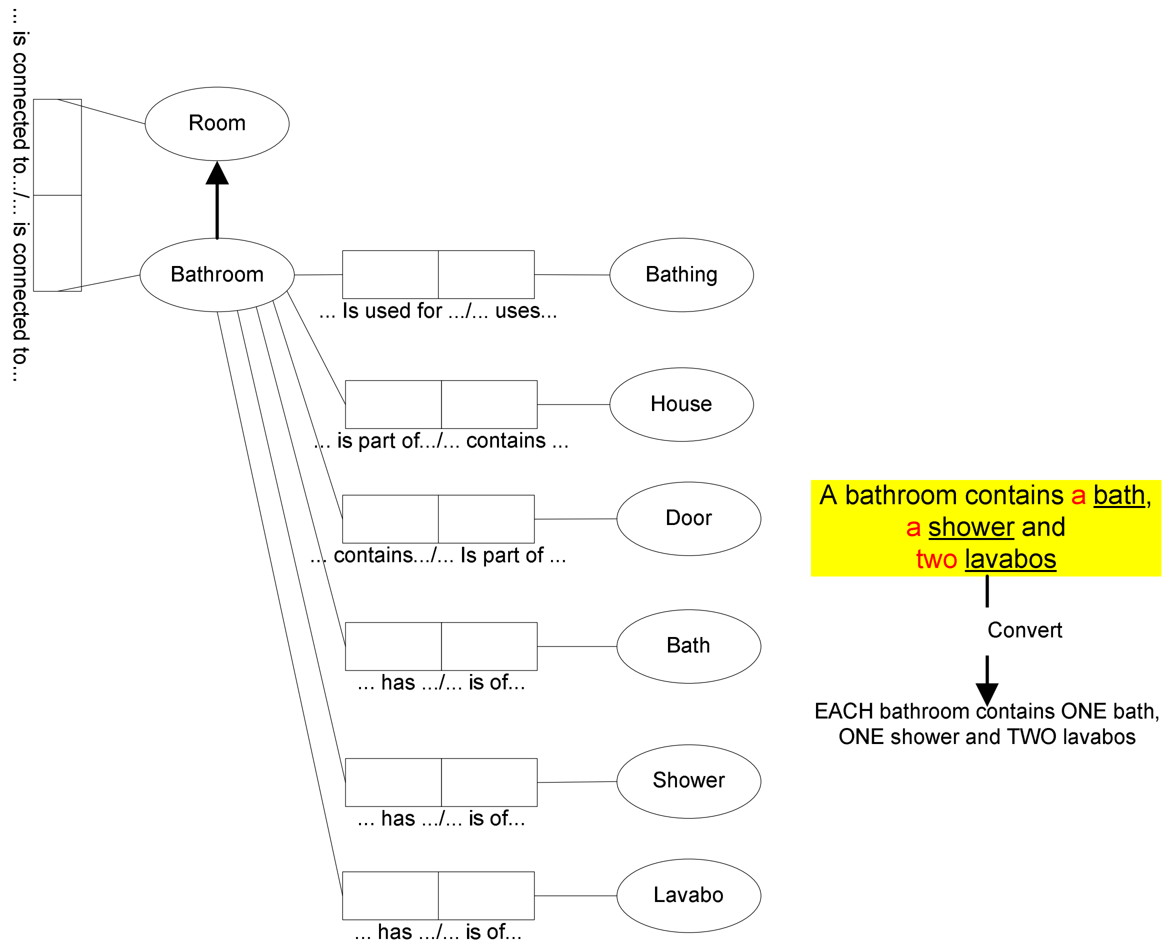

... is connected to.../... is connected to...
Room
Bathroom
Bathing
... Is used for .../... uses...
House
... is part of.../... contains ...
Door
... contains.../... Is part of ...
Bath
... has .../... is of...
Shower
... has .../... is of...
Lavabo
... has .../... is of...
A bathroom contains a bath, a shower and two lavabos
Convert
EACH bathroom contains ONE bath, ONE shower and TWO lavabos

Figure 11: from text to constraints

After all the above activities are executed properly, the ontology engineer translates these constraints into proper ORM models (**Figure 12**).

The resulting ontology should also contain concepts that are different but with same names, and the same concepts with different names. We call the first process "synonym discovery", the result of which is illustrated in **Table 6**. We call the second process "hypernym discovery", the result of which is demonstrated in **Table 7**.

Table 6: Synonym discovery

Name	Concept	Description
Toilet	Lavatory	It sometimes used by the general public to mean a bathroom or washroom, the plumbing industry uses lavatory to mean a bat: hroom washbowl or basin permanently installed with running water. The plumbing industry uses the term "sink" in reference to kitchen sinks. (www.safeplumbing.org)
	Bathroom	The bathroom contains a big washing machine, a small shelf, a vane, a toilet, and a big mirror in the wall. (www.absoluteastronomy.com)
	Privy	an outdoor toilet (en.wiktionary.org)

	…	…
…		

Table 7: hypernym discovery

Names	Concept
double glazing, fanlight, wooden frame window	Window
Water, freshwater, liquid	Water
…	…

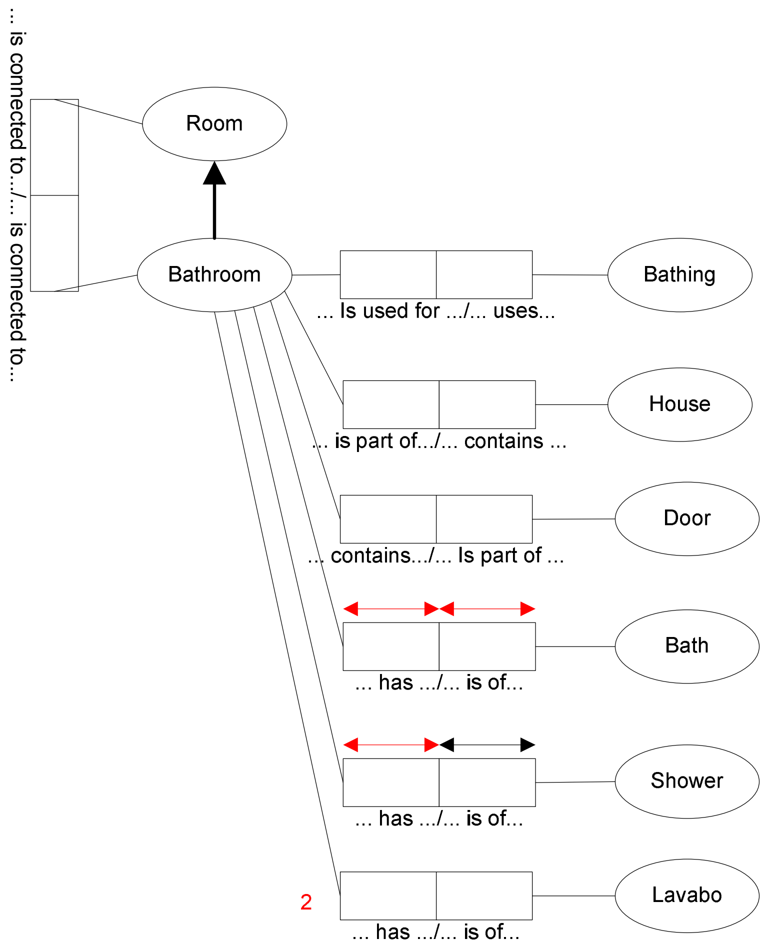


Figure 12: commitment in ORM

2.3 DiYSE Management Strategy

In DiYSE, we use DOGMA-MESS and its portal to manage ontology versioning. In practice, ontologies are developed separately by individual partners. And then, they are manually merged. It is possible because there are not many conflicts among concepts and relations.

3 Smart Spaces Applications and Components Semantics

3.1 Physical layer ontologies

At the lower side of the DIYSE architecture, ontologies play a role in following processes:

1) specification of sensor output / actuator input
2) specification of functionality of runtime components
3) composition constraints of components
4) composition hints of components
5) composition adaptors
6) mapping problem

Note that other metadata, which is not organized in a strict taxonomy, might play a role too. Consider e.g. the configuration data of runtime components, which can be stored together with ontology data in a free format.

3.1.1 Specifying sensor outputs and actuator inputs

We propose to specify sensor and actuator data by the same mechanism that we use internally in the runtime middleware. This makes interfacing to third parties really simple.
Whenever channel messages have to travel across multiple runtimes, they are encoded in Hadoop Avro binary encoding. This also means that they are described by a Hadoop Avro schema.
Avro schema's are much more flexible than XSD in the sense that an Avro encoded message can be decoded with another schema than the one that was used for encoding, as long as the two schemas are sufficiently compatible (examples are simple addition and deletion of record fields). Due to this, Avro encoding ensures much looser binding between components than what could be accomplished with XSD. This makes the system more robust in the case of change.

An avro schema as such only deals with the syntax of the message. We make use of the openness of JSON, which is used to describe an Avro schema to add semantic annotations to the individual message fields.

The example below

```
{
  "type": "record", "name":"temp",
"namespace":"com.alcatel_lucent.wtep.climotics.avro.data",
  "fields": [
            { "type": "double", "name": "value", "owlref":
"http://data.nasa.gov/qudt/owl/unit#DegreeFahrenheit" }
   ]
}
```

Can be parsed by every Avro schema parser, but carries the additional info that temp.value is tied to an ontology: http://data.nasa.gov/qudt/owl/unit#DegreeFahrenheit
By doing this, it becomes undeniably clear what kind of information is carried in the message.
It also becomes possible to insert type adaptors automatically that can convert the temperature to Celsius or Fahrenheit if this would be needed.

Note that this does however not imply that sensor and actuator devices need to send avro messages – what is important is that the datastructure they use can be represented by an Avro schema.

As an example, consider an EEML message that would typically be used to interface to the Pachube service:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<eeml xmlns="http://www.eeml.org/xsd/005"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.eeml.org/xsd/005         http://www.eeml.org/xsd/005/005.xsd"
version="5">
   <environment updated="2007-05-04T18:13:51.0Z" creator="http://www.haque.co.uk" id="1">
      <title>A Room Somewhere</title>
      <feed>http://www.pachube.com/feeds/1.xml</feed>
      <status>frozen</status>
      <description>This is a room somewhere</description>
      <icon>http://www.roomsomewhere/icon.png</icon>
      <website>http://www.roomsomewhere/</website>
      <email>myemail@roomsomewhere</email>
      <location exposure="indoor" domain="physical" disposition="fixed">
         <name>My Room</name>
         <lat>32.4</lat>
         <lon>22.7</lon>
         <ele>0.2</ele>
      </location>
      <data id="0">
         <tag>temperature</tag>
         <value minValue="23.0" maxValue="48.0">36.2</value>
         <unit symbol="C" type="derivedSI">Celsius</unit>
      </data>
      <data id="1">
         <tag>blush</tag>
         <tag>redness</tag>
         <tag>embarrassment</tag>
         <value minValue="0.0" maxValue="100.0">84.0</value>
         <unit type="contextDependentUnits">blushesPerHour</unit>
      </data>
   </environment>
</eeml>
```

If we would use the following Avro schema as an equivalent representation, it would become possible to automatically convert EEML messages into the corresponding Avro message format:

```
{ "comment":"emacs  -*-javascript-*- // $Revision: 1.1 $ // EEML sample schema",
  "type" : "record", "name": "environment", "fields":
         [
         {"name": "location", "type":
                  {"type": "record", "name": "location_struct", "fields":
                           [
                           {"name": "lat", "type": "float", "owlref":
"http://some_ontology/geo_coordinate/WGS84triple/#latitude"},
                           {"name": "lon", "type": "float", "owlref":
"http://some_ontology/geo_coordinate/WGS84triple/#longitude"},
                           {"name": "ele", "type": "float", "owlref":
"http://some_ontology/geo_coordinate/WGS84triple/#elevation"}
                           ]
                  }},
         {"name": "data0", "type": "float", "owlref": "http://some_ontology/derivedSI/#Celsius"},
         {"name": "data1", "type": "float", "owlref":
"http://some_ontology/contextDependentUnits/#blushesPerHour"}
         ]
}
```

Note that we have chosen to include location info as variable part of the message (to accommodate mobile sensors)– all the parts that are supposed to be “static configuration” have been removed from the schema.

Internal messages would have a sourcecode-equivalent structure similar to

```
struct data {
        struct location_struct {
        float lat;
        float lon;
        float ele;
} location;
float data0;
float data1
}
```

3.1.2 Specifying component functionality

In webservices, OWL-S was chosen as a way to semantically describe the service.

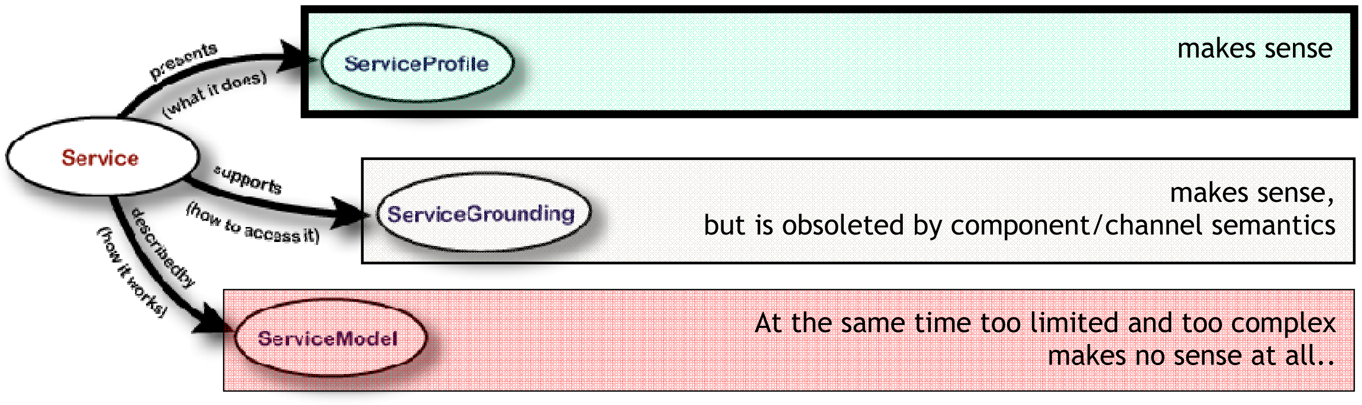

Service
presents (what it does)
ServiceProfile
makes sense
supports (how to access it)
ServiceGrounding
makes sense, but is obsoleted by component/channel semantics
describedby (how it works)
ServiceModel
At the same time too limited and too complex makes no sense at all..

Figure 13 OWL-S in DIYSE

In our DIYSE middleware, we are going to reuse the OWL-S ServiceProfile.
Although the ServiceGrounding makes sense for webservices, we do not need it here since our binding is always over internal channels that are fully described by an Avro schema.
The ServiceModel functionality is not relevant at all, since we are not into modeling of the services.

The ServiceProfile is a formal service specification with details about

- Functionals
  - Inputs and outputs (OWL classes, mapping to the avro schemas mentioned above)
  - Preconditions and results (**format not fixed**).
- Non-functional properties
  - ServiceCategory, serviceName, textDescription, serviceProduct, serviceClassification
  - ServiceParameter class: arbitrary properties that may accompany a profile description

We are using the free format capabilities of the ServiceParameter class to store our component configuration data.

This ServiceProfile will play a role in component exposure, and actual composition.

3.1.3 ServiceProfile and exposure

The creation environment GUI contains a metadata browser. Components are stored in the browser database, with following metadata derived from the ServiceProfile

1) nr of inputs and outputs
2) component class name
3) input port types
4) output port types
5) preconditions
6) effects
7) configuration parameters

The metadata browser will allow extraction of components based on a tag cloud over that metadata.

3.1.4 Composition constraints

The creation environment GUI will only allow the creation of a channel between port types that are « compatible ».
Compatibility in this scope means port types that are in the same ontology class, or that have a parent-child class relationship in the ontology.
Whenever a parent-child match is selected, the more specific porttype is pushed as a parameter to the component/port with the more generic type. Doing this will ensure that the component can configure itself to automatically handle the specific type (e.g. a temperature input which can accept Celsius, Kelvin and Fahrenheit will be configured for the appropriate temperature unit).

3.1.5 Composition hints

The creation environment GUI also has a built-in ontology reasoner and graph solver. That way, it will be able to search the component libraries for a sequence of components that are able to connect a particular porttype to another particular porttype. Multiple solutions are returned, sorted on most likeliest match. The application designer can then accept any solution which will be automatically added to the application graph.

3.1.6 Composition adaptors

Semantic annotation of port types also allows for automatic insertion of data adaptors.

Even more can be done. Starting from a published ontology that specifies derived units into base units by means of a linear transformation, it is possible to automatically generate thousands of type adaptors.

Consider
```
<qud:NotUsedWithSIUnit
rdf:about="http://data.nasa.gov/qudt/owl/unit#DegreeFahrenheit">
    <qud:abbreviation rdf:datatype="&xsd;string">degF</qud:abbreviation>
    <qud:code rdf:datatype="&xsd;string">0525</qud:code>
    <qud:conversionMultiplier
rdf:datatype="&xsd;double">0.5555555555555555556E0</qud:conversionMultiplie
r>
    <qud:conversionOffset
rdf:datatype="&xsd;double">255.37037037037037037E0</qud:conversionOffset>
    <qud:quantityKind>
      <owl:Thing
rdf:about="http://data.nasa.gov/qudt/owl/quantity#ThermodynamicTemperature"
/>
    </qud:quantityKind>
    <qud:symbol rdf:datatype="&xsd;string">degF</qud:symbol>
    <rdf:type>
      <owl:Class rdf:about="&qud;TemperatureUnit"/>
    </rdf:type>
    <rdfs:label rdf:datatype="&xsd;string">Degree Fahrenheit</rdfs:label>
  </qud:NotUsedWithSIUnit>

 <qud:DerivedUnit
rdf:about="http://data.nasa.gov/qudt/owl/unit#DegreeCelsius">
    …
    <qud:conversionMultiplier
rdf:datatype="&xsd;double">1</qud:conversionMultiplier>
    <qud:conversionOffset
rdf:datatype="&xsd;double">273.15</qud:conversionOffset>
</qud:DerivedUnit>
```

The following processing steps
# find all different **quantities**
# iterate over all quantities, create converters
## iterate over all n **units** within a quantity
## extract **offsets** and **multipliers**
## generate all variations of 2 out of n

Will generate a very large set of transformation specifications of the following form:

```
# automatically generated converter between
#  http://data.nasa.gov/qudt/owl/unit#DegreeFahrenheit and
#  http://data.nasa.gov/qudt/owl/unit#DegreeCelsius
{"value" => ( 0.555556 x input["value"] + 255.37 - 273.15 ) / 1}
```

This is actually a ruby script that does the data transformation. This script can be loaded in a component that just runs this ruby script on every incoming message, sending the transformed value on it's output.
The creation GUI solver will be able to insert this adaptor wherever the transformation is required.

3.1.7 Mapping problem

Most of the component mapping functionality deals with matching of component/interconnect requirements versus platform/network capabilities. Apart from the fact that the component specifications can be added to the OWL-S ServideProfile, there is no big role for OWL-S in this process.

There is however an aspect of this mapping that is relying strongly on ontologies, more specifically the taxonomy describing the properties used in expressing requirements and capabilities.

Similarly as with the port mapping, generic requirements ("a screen") can be provided by more specific capabilities ("the screen of this mobile phone"). It would be very difficult to handle this without ontology support. Note that this ontology is not necessarily interconnected with the OWL-S profiles, since the component specifications are using free-form entries in the OWL-S specifications. Any correspondence is just an interpretation of our mapping tools that understand to map a string "MIPS"="500" in the component OWL-S specification to a resource class http://some_ontology/resources/#MIPS in the resource ontology.

3.2 Semantics and ontologies for multimedia analysis

A lot of efforts have already been done so far on ontologies, semantic and reasoner [Lacot05, Antoniou04, Passin04]. We will present in this section the main existing basis and their links with hypermedia. As detailed in [OntologyWiki], ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain.

The TV Anytime specification provides very little low level information but provides a large panel of semantically higher level information (e.g., title, synopsis, genre, awards, credits, release information, etc.).

3.2.1 TV-AnyTime

[TV-Anytime] is initially dedicated to TV services. It handles the segmentation and the repeatable broadcast of multimedia resources or programs. It is also interesting to note that TV Anytime uses a subset of the MPEG-7 standard for low-level information. The CRID (Content Reference ID) is the solution TV-Anytime has standardized for referring to and obtaining content. All CRIDs are published by an authority. In the meantime, in order to provide a unique, internationally recognized and permanent reference number for each audiovisual work, the ISAN (International Standard Audiovisual Number) is a voluntary numbering system for the identification of audiovisual works. During the registration of a work, mandatory descriptive information are requested such as the title of audiovisual work, thee list of original language(s) of audiovisual work, the year of reference, the type of work and the live action or animation. Moreover, optional information can be added such as the full name of main producer and other language version(s). The final objective is to link the multimedia description to a unique allocated multimedia identifier.

3.2.2 RDF / RDFS

At the opposite, the Resource Description Framework (RDF), as its name implies, is a general framework for describing and interchanging metadata. RDF statements define relationships between concepts (graph nodes) which are subparts of semantic domain. As detailed in [Bray98], the RDF framework is built on the following rules:

1. A Resource is anything that can have a URI. This includes the entire world's Web pages, as well as individual elements of an XML document.
2. A PropertyType is a Resource that has a name and can be used as a property, for example Author or Title.
3. A Property is the combination of a Resource, a PropertyType, and a value.
4. There is a straightforward method for expressing these abstract Properties in XML.

The framework evolved and a newer introduction to RDF can be found in [Tauberer06]. As an example, in the following description “A man is driving the taxi”, the triple is constituted of a RDF/XML could be like:

```
<rdf:Description rdf:about=« http://person.org#man »>
<drive>
    <rdf:Description rdf:about=« http://car.org#taxi »>
</drive>
</rdf:description>
```

However, there are still limitations that prevent from using this standard to develop description framework. In fact, since RDF does not precise keywords to use to define triples, reasoning is required to resolve synonymy or equivalence between concepts. Moreover, particular attention is required during domain definition since there is no distinction between schemas and data in triples. Finally, Notation simplicity requires higher number of triples to avoid ambiguity but it introduces reading and computation disadvantages.

RDF for Schema (RDF-S) tries to resolve this problem by providing a data model. FOAF specified in [Brickley07] is an example of RDF-S which provides a template to describe a person. As illustrated in [RDFS_Wiki], an instance of foaf:Person is a resource linked to the class using an rdf:type predicate, such as in the following formal expression of the natural language sentence: “John is a Person”.

Nevertheless, RDFS mainly remains a catalog format created in XML context. Since we need to create, enrich, and represent media knowledge, the more expressive language Web Ontology Language (OWL) might be the standard to use as it allows processing the content of information. However, Many RDFS components are included in the Web Ontology Language.

3.2.3 OWL

As detailed in [OWL04], OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. 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 has three increasingly-expressive sublanguages. The simplest one is OWL Lite which supports those users primarily needing a classification hierarchy and simple constraints. OWL DL 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). Finally, OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.
Ontologies development is divided into six tasks [Isaac05]:

- o *Specification*: Specify the use of built ontology;
- o *Conceptualization*: Build a data model according to domain knowledge;
- o *Formalization*: Convert conceptual model to formal model;
- o *Integration*: Reuse as most as possible existing ontologies;
- o *Implementation*: Build corresponding computer comprehensible execution model;
- o *Maintenance*: Keep ontology up to date as required.

Tools like [Protégé] allows building ontologies as detailed in [Horridge04]. The next consist in verifying ontology syntax and coherency using WonderWeb OWL Ontology Validator [Bechhofer04] as an example to facilitate interoperability and reasoning.

3.2.4 Semantic multimedia description model

In theory, ontologies and semantic bring to multimedia description huge capabilities of description's enrichment. We should be able to describe multimedia content and create a "virtual representation" of the multimedia through descriptors. However, we are not yet able to watch a film by “playing” its semantic description. Thus, it is not yet possible to retrieve a scene in a list of multimedia using constraint natural language description like “Every media where Bill is fighting his disciple”. This is due to the semantic gap as illustrated in Figure 22.

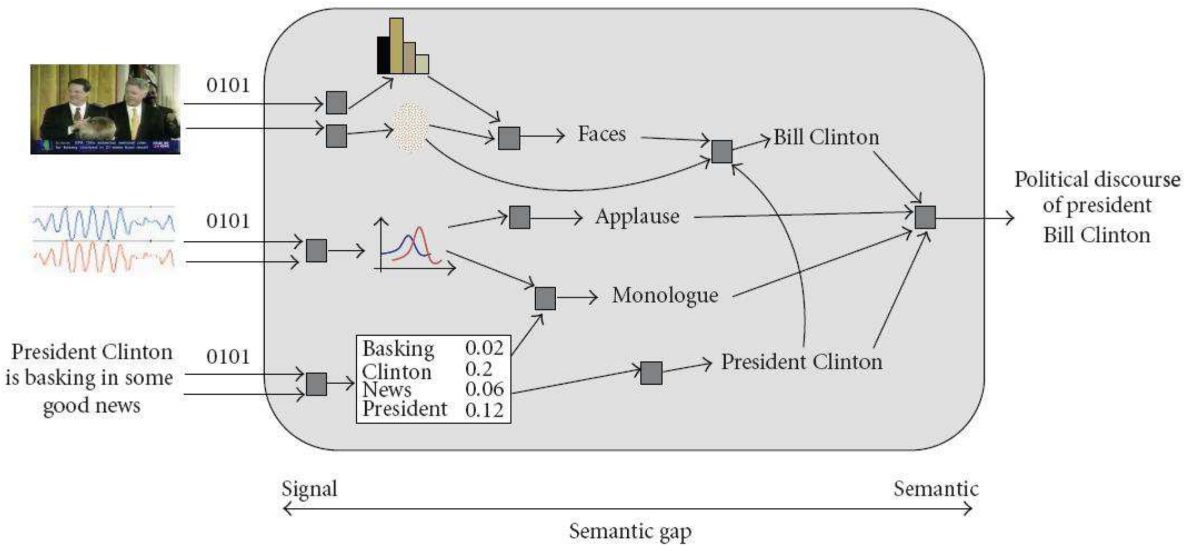


Figure 22: Semantic gap illustration [Ayache07]

The semantic gap which is a well known problem in multimedia as reported in [Smith07]. To give references, as said earlier, Google improved its image search engine to allow results to correspond with user's keywords by implementing a human computing solution to describe web images. However, we don't want to say that semantic doesn't provide solutions. People are already using semantic without knowing it. As an example, Microsoft deployed Embodied Conversational Agent (ECA) in Microsoft OFFICE Assistant [Cassell01] and more recently in user's FAQ. Those ECA are based on semantic reasoner to classify user's requests and automatically analyze keywords and their relationships to find best topics or links to send back to users.

The main problem as detailed in [Delezoide06] is the combination of descriptors at different granularity level to build virtual context multimedia description. The author concludes on the following states. Based on [Martin98] and [Bennett02] as a first step, fusion of concepts and high level concept classification is not possible due to the huge amount and complexity of information and relationships to consider and maintain. Moreover, the difficulty to use concepts to describe low level descriptors like colors as an example makes it also impossible to maintain. On the other hand, fusion of concepts and low level descriptors allows combining low level descriptors describing an object to determine higher abstraction level concepts as illustrated in Figure 23.

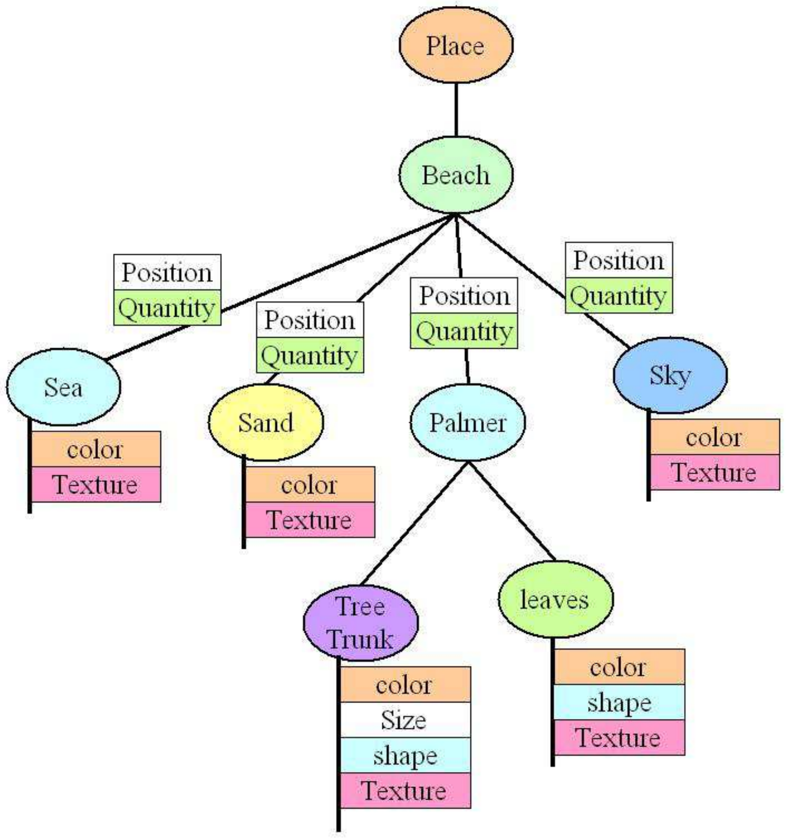


Figure 23: Example of ontology to describe a beach

The main idea, named paradigm annotation, is to manually label a set of concepts present in a multimedia (image, video…) to establish relationships between correlated low lever descriptions and higher level concepts. It is therefore possible to train a concept detector to retrieve known concepts in other multimedia.

3.2.5 From world description to analyzers

Existing standard and formats provide a wide range of tools. However, main unresolved difficulties still require investigations such as the management of the granularity of the multimedia descriptions, the expression of the input and output formats, the pre requirements and effects specifications. Accordingly to existing world representation using ontologies, the Figure 35 illustrates a representation the different levels of granularity to describe a domain comparatively to multimedia analyzer capabilities.

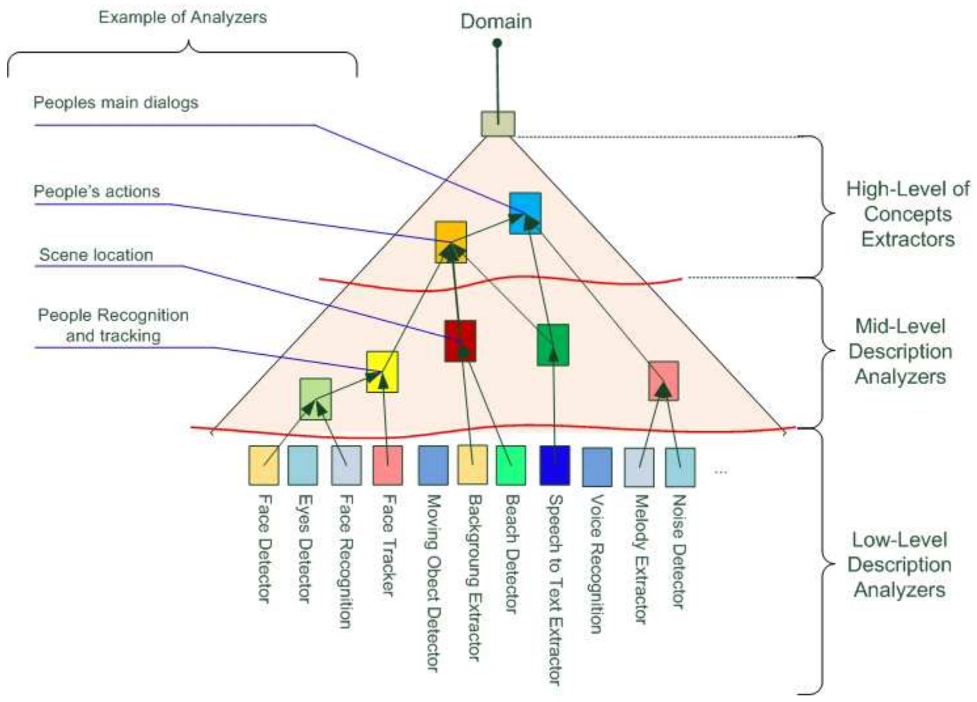


Figure 35: Description domain at different levels of Granularity

In the Figure 35, every square represents a multimedia analyzer providing information to enriched multimedia description. The bottom of the pyramid represents the low-level descriptors. We consider as low-level descriptors the analysis results only based on multimedia analysis (main color, histogram extraction, shape detection…). The middle of the pyramid represents mid-level of multimedia analysis. We consider as mid-level the analyzers which combines results from other multimedia analyzers.
Therefore, since we are not able to directly describe the world, the idea is to split world description into "small parts" according to concerned domains using ontologies.

3.2.6 Contributions

We propose a modular architecture enabling the combination of both descriptive approaches: low level descriptors and ontologies in order provide a higher capacity of description for concepts to detect. As introduced in [Arndt07], we choose to implement the MPEG-7 standard to implement low-level descriptors and ontologies through OWL to describe higher abstraction level of concepts.

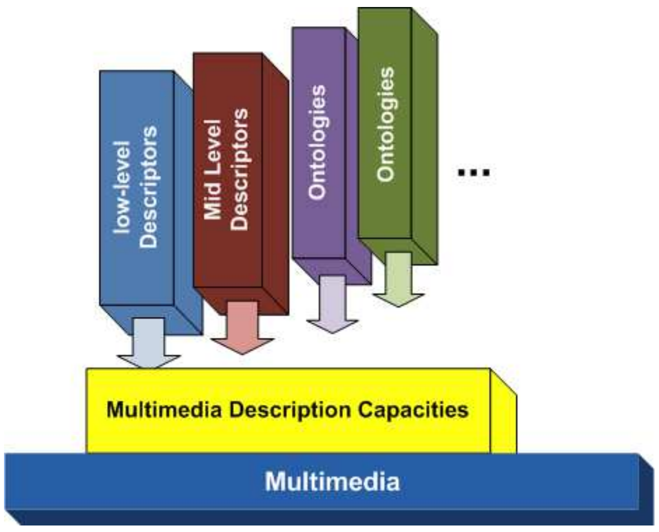


Figure *14* : Extensible description model

It is clear that this model does not solve the semantic gap issues in the media analysis area. Our objective is to transfer the semantic gap on the media analyzer. Only the analyzers have the capabilities to provide results of a media analysis. This model provides, through inference engine, a way to link analyzers through the available description tools.

3.2.6.1 Metadata aggregator

The Metadata Aggregator is the database storing the output data of the analyzers (face recognition, gesture recognition, etc..). By providing the capabilitiy to combine information coming from different analyzers or by providing this information to other “higher-level” analyzers, the metadata aggregator provides a better comprehension of the environment and then enables to take decisions adapted to the situation.

The Metamodel

To get a first low-level of metadata description, the aggregator supports a first draft metamodel as described figure15. This model is based on several inputs coming from MPEG7.

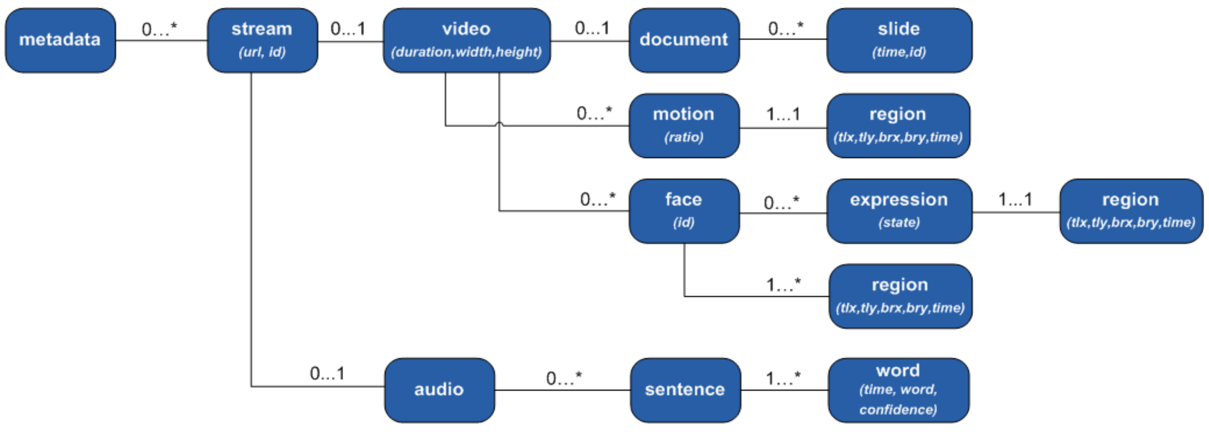


Figure 15 : DiY multimedia metamodel

Lightweight solution and low cost integration

The first approach, when the target is to get a single integration point, is to build a framework that developers of analyzers use. In this case they use the same mechanism to publish their data. However, this solution implies that a programming language is selected and shared by all the developers. If it is not the case, developers have to write wrappers to adapt their software to the framework language and this work is not attractive for providers of multimedia analyzers.

It is for this reason that we select a solution which has a lower cost of integration.

To achieve our goal, it was relevant to build the metadata aggregator as a network element: It is a lightweight webserver which proposes basic APIs to manage metadatas:

- A push-oriented method enables analyzers to put their information in the aggregator
- A pull-oriented method exposes information to other components
- An event-oriented method enables consumers to register to events and then, to be triggered as soon as new information is available.

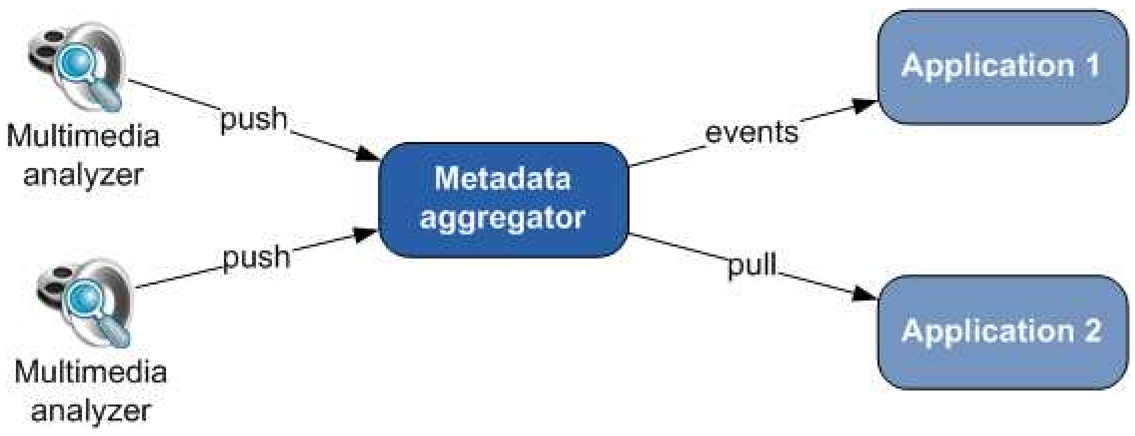


Figure 16 : Metadata Aggregator

In order to get a high-level of compatibility with any technologies, some simple protocols have been chosen:

- HTTP (Hypertext Transfer Protocol - http://www.w3.org/Protocols/): the most famous request-response protocol is used in three methods
- Yaml (Yaml Ain't Markup Language - http://www.yaml.org/): language used for messages in the push-oriented and the events-oriented method
- XML (Extensible Markup Language - http://www.w3.org/TR/REC-xml/): language used for exposing all information in the pull-oriented method.
-

Push-oriented method

The metadata aggregator enables to push metadatas using a YAML message over HTTP. Developers of multimedia analyzers haven't to spend so much time to be compatible with the aggregator since these standards are available in every programming language.

Pull-oriented method

To retrieve information, a pull method is provided. Then, consumers can get the whole metadatas tree in XML and an option lets them specify that they only want information acquired in the last xxx seconds.

Events-oriented method

To be efficient and responsive, the pull approach is not enough: by getting only that one, it would imply that some applications have to often poll the aggregator and even with that, it might be not reactive enough if a low-latency is required. The events-oriented method is the implementation of the Hollywood principle (http://en.wikipedia.org/wiki/Hollywood_Principle). That's the reason why consumers can register to some events on the aggregator when they want to be triggered on them. Then, each time new information is available, the aggregator will send an event to the requester using HTTP and YAML.

The metadata aggregator can be fed using the following URL: http://'aggregatorip'/info/add.

Example of the metadata format:

- **Set the size of the video:** ==|| {streamId: bigbrother, uri: /metadata/stream/video, width: 640, height: 480} ||==
- **Set the url of a stream:** ==|| {streamId: bigbrother, uri: /metadata/stream, url: 'rtsp://172.25.70.73/mpeg4'} ||==
- **Add a motion node:** ==|| {streamId: bigbrother, uri: /metadata/stream/video/motion, ratio: 555, region: {tlx: 5, tly: 5, brx: 600, bry: 400, time: 1298645801790}} ||==
- **Add a face node:** ==|| {streamId: bigbrother, uri: /metadata/stream/video/face, id: alice, region: {tlx: 5, tly: 5, brx: 600, bry: 400, time: 1298645801790}} ||==
- **Add a face expression node:** ==|| {streamId: bigbrother, uri: /metadata/stream/video/face, id: bob, expression: {state: smile, region: {tlx: 5, tly: 5, brx: 600, bry: 400, time: 1298645801790}}} ||==

3.3 Ontologies to support application composition

Semantic technology can help the composition process in following ways:

3.3.1 Discovering reusable components when building applications

In a DIY environment, we expect lot's of components to come from designers that only cover a very small fraction of the potential component market ("long tail" for components). This makes discovering what is available a big challenge. Exposure and discovery mechanisms already exist, but the environment we consider in DIYSE has the added problem that what can be learned about the components from these discovery frameworks, is not published in some kind of industry standard but now comes from a plethora of individual contributors. Clearly, there is a need to be able to confer meaning as intended by a developer to all potential users and developers of related components. Secondly, there is a need to merge post-factum meanings as defined by multiple designers working on related components into a single encompassing framework. Ontologies, and ontology management frameworks, carry the promise to cover both requirements.

Note that some gains can already be accomplished by deploying "folksonomies" (based on a simpler concept of tags i.s.o. a very formal hierarchy of concepts). Tag browsing on component features, functionality, interface types and interface pre-or postconditions can already significantly help in selecting appropriate components for a particular application.

To this purpose, ALU has made it's PecMAN (meta-)data management infrastructure available as part of the application creation GUI. More details can be found in D1.3, Section 4.1.

3.3.2 Verification interface compatibility between components

When component interfaces are syntactically and semantically tagged, it becomes possible to check in the creation GUI if two interfaces can connect to each other.
Strictly speaking, syntactical typing will already allow most of this functionality.
Semantic typing allows us to push the functionality up:

1) ontologies can express concept hierarchy – now it becomes possible to attach component interfaces that are not identical but have a hierarchical relationship (like “temperature”, and the more specific “temperature_Celsius”)
2) semantic types allow for the automatic insertion of type adaptors (in the above example, a component that would translate temperature_Celsius to a temperature interface that requires temperature_Fahrenheit, as an example)
3) semantics can also help by describing not only interface types, but interface pre- and post-conditions. (see OWL-S for an example). A connection would only be allowed if the postcondition on the output satisfies the precondition on the input.

3.3.3 To propose a subgraph of components that allows to connect particular interfaces output_A and input_B

In the situation where all components are maximally semantically annotated, it becomes feasible to deploy reasoner algorithms to solve for a connected set of components (a graph) that will allow to connect the output of a component to the input of another component. The reasoner can come up with a series of possible solutions, leaving the final choice to the discretion of the application designer.

3.4 DIY-CDR: An Ontology-based, Do-it-Yourself Components Discoverer and Recommender

In order to answer the question “what can DiYSE benefit from ontologies?”, VUB STARLab from the Belgian consortium has designed and implemented an ontology-based DIY component discoverer and recommender (DIY-CDR).

DIY-CDR supports users to DIY their personalized Internet-of-Things (IoT) applications. Before a person starts his DIY processes, it is important for him to find proper components that meet his needs. This article records our recent results concerning how to automatically discover and recommend existing components to users. The Controlled Fully Automated Ontology Assisted Matching Strategy (C-FOAM, (Tang et al., 2010)) is a matching strategy that contains algorithms at three different levels – String, Lexical and Conceptual (or Graphical). In this section, we extend C-FOAM and embed it in a component discovery and recommendation module, which is called Do-It-Yourself Component Discoverer and Recommender (DIY-CDR). DIY-CDR also uses semantic decision tables (SDT, (Tang, 2010)) to gather user specific decision rules and set up parameters for C-FOAM.

3.4.1 Background Knowledge

In this section, we will discuss the background knowledge, which covers semantic decision tables and our motivation.

3.4.1.1 Semantic Decision Table

A semantic decision table (SDT) is a decision table properly annotated with an ontology. Table 8 is an SDT example, which decides whether a student studies or visits his friends depending on the weather and exams. It contains three parts – a decision table, a set of lexons and commitments. A lexon $l l$ is a binary fact type $\langle \gamma, t_1, r_1, r_2, t_2 \rangle \langle \gamma, t_1, r_1, r_2, t_2 \rangle$ where $t_1 t_1$ and $t_2 t_2$ are the two terms that present two concepts; $\gamma \gamma$ is a context identifier that points to the resource where $t_1 t_1$ and $t_2\ t_2$ are originally defined; $r_1 r_1$ and $r_2 r_2$ are the two roles that these two concepts can play with.

An example is $\langle \gamma_1,\ \text{Student},\ \text{takes}, \text{is taken by}, \text{Exam} \rangle\ \langle \gamma_1,\ \text{Student},\ \text{takes}, \text{is taken by}, \text{Exam} \rangle$, which

expresses a fact that within the context that is identified by γ_1, γ_1, a student takes an exam and an exam is taken by a student.
A lexon in the annotation set can be either a lexon selected from the ontology, or a lexon that contains an annotation.

Table 8 An SDT on deciding whether a student studies or visits friends

Condition				
Weather	Sunny	Sunny	Raining	Raining
Exam	Yes	No	Yes	No
Action				
Study	*		*	*
Visit friends		*		
SDT Commitments in DECOL				
1 P1 = [Student, takes, is taken by, Exam]: MAND(P1)				
2 P2 = [Study, is a, supertype of, Action]: SUBTYPE (P2(Study), P2(Action))				

For example in Table 8,
$\langle\gamma_1, \text{Student, takes, is taken by, Exam}\rangle\langle\gamma_1, \text{Student, takes, is taken by, Exam}\rangle$ is a lexon selected from the ontology. The lexon $\langle\gamma_2, \text{Study, is a, supertype of, Action}\rangle$ $\langle\gamma_2, \text{Study, is a, supertype of, Action}\rangle$ is an annotation. Suppose “Action” is already defined in the ontology; we annotate the decision item “Study” by building an “is-a” taxonomical relation with “Action”. This annotation lexon can as well be added to the ontology if the community of domain experts agrees. Note that this process is called ontology versioning (or “ontology evolution”), which is out of scope of this section.

A commitment (also called an “ontological commitment”) is a statement of making use of lexons through a formal agreement within the community of stakeholders (also called a “group of interests”). It contains constraints and ontological axioms. We need to use a language with well defined syntax, for instance, Decision Commitment Language (DECOL, (Tang and Meersman, 2009)), to express a commitment, or a graphical language, e.g. Semantic Decision Rule Language (SDRule-L, (Tang et al., 2009)), as its modelling means. DECOL and SDRule-L can be translated into a controlled natural language for verbalization, and published in an XML format. Table 8 illustrates two commitments in DECOL. The first one contains a constraint of mandatory, which means that each student takes at least one exam. The second one contains a subtyping.

Although an SDT commitment is a commitment, there exists a difference, which is the level of shareability and compliance. Within a community (which defines/uses the ontology), all the ontological commitments must be used by all the stakeholders (and/or software agents) from this community in a consistent and ambiguous way. An SDT commitment can be application specific. Hence stakeholders are not required to use or comply with it except the ones who define it. We can add an SDT commitment to the ontology if agreed, the process of which we also refer to ontology versioning.

3.4.1.2 Motivation

In a DIY world, semi-technical people play an important role. Pro-Am (Professional-Amateur) is a kind of semi-technical. It is defined as an amateur who practices to have the same dedication as a technician. Its birth is the result of the shift of knowledge (from centralized to distributed). Pro-Am is sometimes also called “lead user” and “bricoleur”. They are often networked. A Pro-Am can became a technician if he is enough knowledgeable.

The DIY aspect in Onto-DIY (Tang et al., 2010’) is required not only by Pro-Am or technicians, but also by non-technical users. They need to have a means to create their own ambient applications using their own semantics.

It is important for them to find appropriate hardware and/or software components before buying smart objects. It is the first motivation of DIY-CDR, which is to discover and recommend the building blocks. It should happen right after they have rough ideas of what to be built.

The second motivation is to continuously stir DIY ideas by allowing users to access to the community-based web portal (such as forums, polls and wiki pages). This kind of DIY activities is called “co-creation” and considered as a trend of DIY. In co-creation, we consider users as prosumers. They produce ideas, and consume the ideas produced by others. When people start to share their new DIY solutions, more and more ideas will be inspired and created. When users share their innovations, others benefit from it and gain access to novel extensions to the original products or services. DIY-CDR can be used to find similar DIY solutions and group them together if necessary.

In this section, we have discussed the background knowledge about ontology engineering and semantic decision tables. We have also pointed out two important motivations. We will focus on the design of DIY-CDR in the next section.

3.4.2 Design of DIY-CDR

The design of DIY-CDR is illustrated in Figure 17. Our framework contains a community portal, a matching engine and a rule engine. The community portal provides an easy access to end users. When it gets the input from users directly (via a webpage) or indirectly (e.g., through a smart device or a remote desktop editor), the matching engine will consult the rule engine. Based on the rules, the matching engine will calculate matching scores using the knowledge base and feed them back to the community portal. At the end, the community portal will transform the scores into a list of recommended components. The recommendation can be further passed to the output devices if necessary.

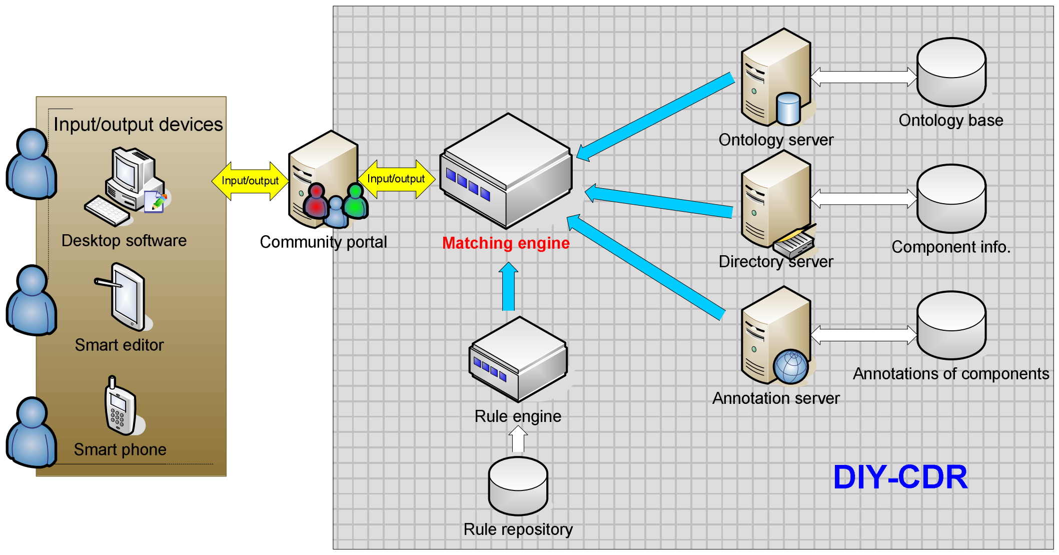


Figure 17 the design of Do-It-Yourself Component Discoverer and Recommender (DIY-CDR)

The knowledge base contains an ontology base, a user specified application rule repository and a set of component information databases. The matching engine uses the ontologies through an ontology server. It communicates with a directory server for the information of a component. An annotation server provides the matching engine with the annotation of a component.

Note that this design is general. Within the scope of this article, we use a lexon base and a commitment layer to construct the ontology base; the rule repository contains a set of SDTs. The component information databases are vendor/company specific databases of the components, with which a user builds his personalized smart environment.

Note that we use two kinds of ontologies – a context ontology and domain ontologies. The domain ontologies evolve when a new concept is introduced and/or an old concept is obsolete. The context ontology is built based on existing context ontologies. It evolves much slower than the domain ontologies. During a project, we can consider it unchanged.

In the following two subsections, we will illustrate the context ontology and an example of domain ontology.

3.4.2.1 The Context Ontology

The concept of *context* (Dey, 2009; Schilit et al;, 1994; Strang et al., 2004) has been always important in context-aware systems during different evolutionary periods, namely the epochs of *distributed computing*, *ubiquitous systems*, *pervasive computing* and *IoT*.

We model the contexts in the context-aware systems for our DIY scenarios in a context ontology. Strang and Linnhoff-Popien (2004) illustrate a survey on context modeling in general.

The context model illustrated in (Henricksen et al., 2003) is an extension to ORM (Halpin, 2001), which allows *fact types* (also called *lexons* in the previous subsection) to be categorized based on the persistence and source. CONON (Wang et al., 2004) is a context ontology, which models basic and generic concepts concerning a context. The authors in (Ou et al., 2006) use a psychological analysis method to study the difference between recall and recognition using contextual information, which is normalized across domains. Other remarkable related work is aspect-scale-context information model (ASC, (Strang, 2003)), OMG's Model Driven Architecture (MDA) based Context Ontology Model (COM, (Ou et al., 2006)) and context ontology language (CoOL, (Wang et al., 2004)).

We build the context ontology based on the above related work, the open geospatial (OGC) standards[1] and our previous work in (Tang et al., 2009). Figure 18 shows basic concepts and their relations (also called *lexon roles* in the previous section) that are used to describe a context. In order to distinguish them from the concepts from the domain ontologies, we call these concepts (see what follows) as "upper level concepts".

- **Person** is an individual of human beings. It is the basic and atomic constituent of a community (see below).
- **Community** is any group of human beings collectively. Within a context, the group is specified with a common role that all the members play. Each community has *at least one* person.
- **Time** is an identifiable instance or a period of a moment of an action executed by a person. A **timestamp** is a string that indicates a single point in time and is also called an *instance* of time. A **time span** is defined as a path through time or *duration* of time. A time is measured by a time span or a timestamp.
- An **object** is a tangible thing used by a person. As a subtype of object, a **device** (also called "hardware component") is an instrument or equipment for a particular purpose. **Software component** is another subtype of object, which corresponds to a software piece, such as a software module or a plug-in. Each object is used by *at least one* person.
- An **activity** is executed by a person through space and time. A **task** (which is a subtype of activity) in is often seen as a specific piece of work required to be done. Each activity is executed by *at least one* person.
- A **location** is a point or extent in space. It determines the place where an activity that should happen or a task that needs to be executed. A person executes an activity during *an exact one* time span, in *an exact one* location. A person also uses an object during *an exact one* time span, in *an exact one* location.

[1] http://www.opengeospatial.org/standards

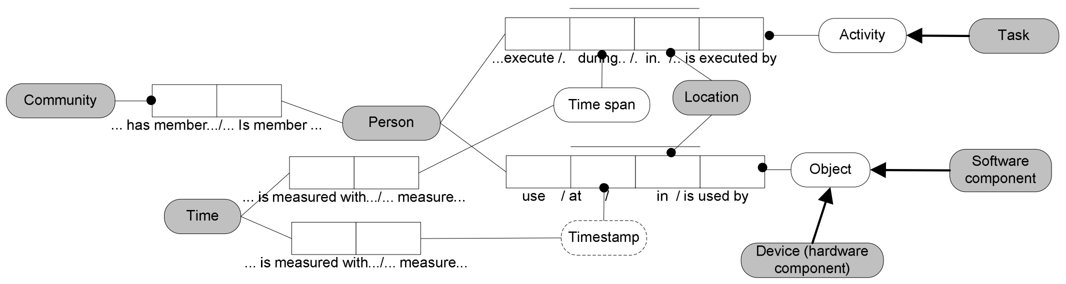

Activity
Task
...execute /. during.. /. in. /.. is executed by
Community
... has member.../... Is member ...
Person
Time span
Location
Object
Software component
Time
... is measured with.../... measure...
use / at / in / is used by
Timestamp
... is measured with.../... measure...
Device (hardware component)

Figure 18 a context model in SDRule-L

Note that a context ontology is called "upper ontology" in (Wang et al., 2006) when it is used as a generic reference model for domain ontologies. It is an ontology that contains *generic patterns* for creating, updating and using domain ontologies. The method of creating domain ontologies using a context ontology can be found in our previous work (the PAD-ON methodology, (Tang et al., 2010)), with which we have created domain ontologies (see the next subsection).

3.4.2.2 An Example of Domain Ontology

Based on the context ontology, we build the domain ontologies from a domain dictionary, vocabulary, taxonomy, business process models, expert's consultancy, standards/specifications and WordNet.

The concepts in a domain ontology are the subtypes (or subclasses) of the concepts in the context ontology (see the previous section).

The relations between the concepts, generally speaking, should not be limited to certain types. However, we need to identify a few domain canonical relations in order to use them for matching. A normal relation is, for example, "study" in the lexon $\langle \gamma_1, \text{Student}, \text{study}, \text{is studied by}, \text{Book} \rangle$ $\langle \gamma_1, \text{Student}, \text{study}, \text{is studied by}, \text{Book} \rangle$. Unlike a normal relation, a domain canonical relation is easily interpreted, utilized and reasoned by an ontology-based agent. Note that when a normal relation is interpreted by an engine, it becomes a domain canonical relation. The procedure of mapping relations to execution code belongs to the process of ontological commitment.

Including the ones illustrated in Figure 18 (e.g., has member/is member, is measured with/measure, use/is used by, etc.), the domain canonical relations also contains the "is-a" taxonomical/subtype/subset relation, the "part-of" mereological relation, the relation of "has" (e.g., "property of").

Table 2 ontological resources

Document Reference	Document Type	Ontology/Domain
IBM Rational Unified Process (RUP®)[2]	Standard/spec.	Ontology of software dev.
Software structure (e.g., resource 1[3])	Expert's consultancy	Software ontology
Open document ISO/IEC 26300	Standard/spec.	Ontology of Software doc.
OGC 05-005 web map context doc.	Standard/spec.	Map ontology
Nabaztag online resource[4]	Vocabulary	Nabaztag Rabbit ontology
Yahoo! Pipes online resource[5]	Vocabulary/taxonomy	Yahoo pipes ontology
…	…	…

We have used the resources illustrated in Table 2 to build the domain ontologies.

[2] http://www-01.ibm.com/software/awdtools/rup/
[3] http://users.cecs.anu.edu.au/~kambara/old_site/software/software_standards.html#28036
[4] http://help.nabaztag.com/
[5] http://pipes.yahoo.com/pipes/

Figure 19 is an example of the model that describes Yahoo pipes in the Yahoo pipe ontology. The concepts in gray are the ones defined in the context ontology (Figure 18). Each "Person" has exactly one "Name". The type "Yahoo pipe" is a subtype of "Online service", which is a subtype of "Software component". Each "Yahoo pipe" has exactly one "Pipe title" and at least two "Modules". Each "Module" has at least one "Terminal". "Terminal" is a subtype of "Software component".

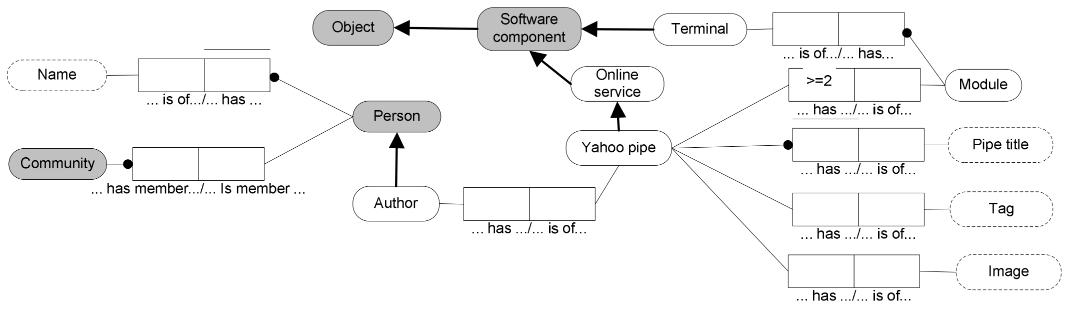


Figure 19 an SDRule-L model that describes Yahoo! pipes

In a domain ontology, there are many models like the one shown in Figure 19. Each model must use the domain canonical relations.

Note that we category a type into Non-Lexical Object Type (NOLOT) and Lexical Objet Type (LOT) (Halpin, 2001). A NOLOT represents a set of non-represent-able entities having common properties, e.g., "Student". A LOT represent a set of values of an entity, such as names and properties in String (e.g., "Mark"). A newly introduced NOLOT must be a subtype of a type in the context ontology (or in the domain ontologies). We do not restrict the structure of the newly introduced LOTs.

We use lexons and SDRule-L models to model a domain ontology. SDRule-L and its markup language (SDRule-ML) can be mapped into RDFs6, OWL7 and SWRL8. We will show an example of mapping SDRule-L to OWL/RDFs in the following sections.

The difference between these languages is the purposes of usage. SDRule-L is used for sharing semantic decision rules. RDFs and OWL are ontology presentation languages. We use SWRL to reason OWL/RDFs models.

After we have the ontologies, we can annotate the components. We use the annotation relations (such as "has label", "has tag", "is an instance of" and "has type", "equivalent to") to build the annotation lexons9 and/or select lexons directly from the domain ontologies. With these annotations, DIY-CDR can call C-FOAM and discover/recommend components. We will explain C-FOAM in the next section.

3.4.3 C-FOAM: Controlled Fully Automated Ontology Assisted Matching Strategy

The C-FOAM strategy contains two important modules (an interpreter and a comparator, Figure 20). The interpreter makes use of a lexical dictionary, annotations and a string matching algorithm to interpret users' inputs. Given a term, the interpreter will return the correct concept defined in the ontologies or the lexical dictionary, or a context, which corresponds to an annotation set.

[6] http://www.w3.org/TR/rdf-schema/
[7] http://en.wikipedia.org/wiki/Web_Ontology_Language
[8] http://www.w3.org/Submission/SWRL/
9 E.g., $\langle \gamma_2, YP_1, is\ an\ instance\ of, has\ instance\ of, Yahoo\ pipe \rangle$

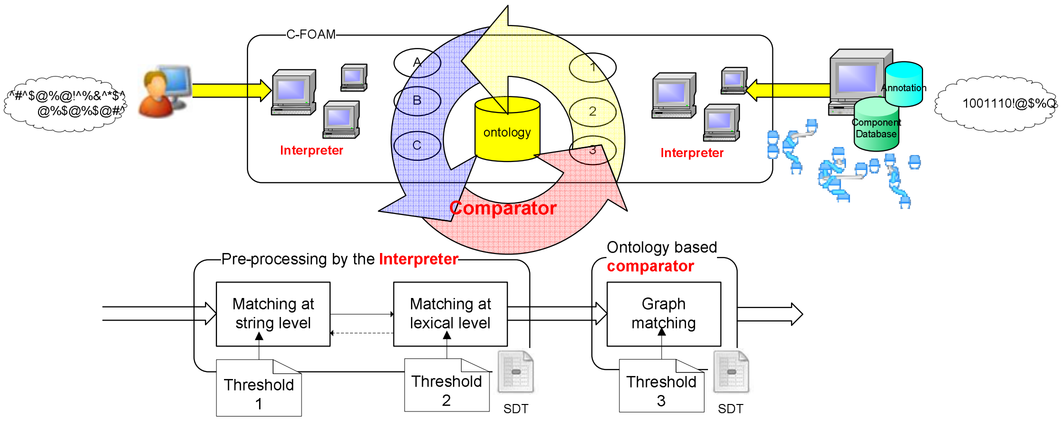


Figure 20 the design of advanced C-FOAM

There are three thresholds in C-FOAM. Two are in the interpreter. The first one is for the internal output using string matching. The filtered terms will be the input for lexical searching. The second one is to filter the output of the lexical searching. The third threshold is in the comparator. It is to filter out the components that are not very relevant.

In this report, we indicate two situations of use cases in C-FOAM.

Situation one: If the result of the interpreter is a concept, then C-FOAM will not continue with the graph matching algorithm, but illustrates the components that contain this concept in the annotation set. For example, if the input is “bringing”, then the string matching algorithm in C-FOAM will find “bring”, based on which the lexical matching algorithm will find its synonym “fetch”. C-FOAM will consult the ontology and find that “fetch” is annotated with “Module”, which is defined in the ontology of Yahoo pipes. At the end, C-FOAM will return a list of pipes that contains the “fetch” module.
We explain this procedure as shown in Figure 21. C(i) is a concept, which can be both LOT and NOLOT in the ontology (e.g., C(i) can be a NOLOT such as “Module”, “Terminal” and “Author” etc., or a LOT such as “Pipe title”, “Tag” and “Name” etc. in
Figure 19).

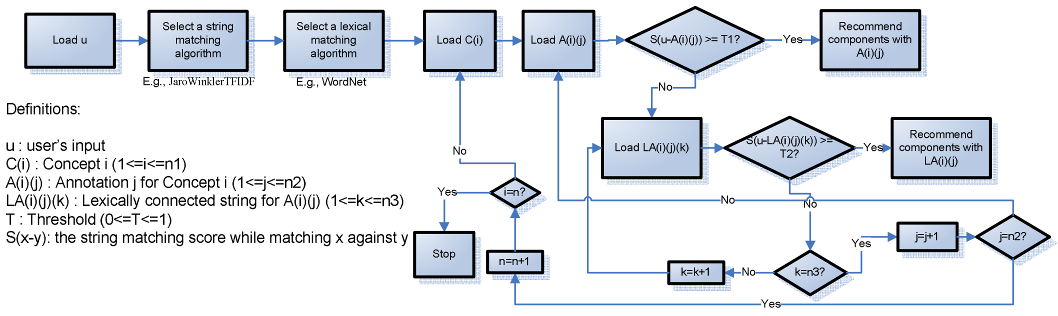

Load u
Select a string matching algorithm
E.g., JaroWinklerTFIDF
Select a lexical matching algorithm
E.g., WordNet
Load C(i)
Load A(i)(j)
S(u-A(i)(j)) >= T1?
Yes
Recommend components with A(i)(j)
No
Load LA(i)(j)(k)
S(u-LA(i)(j)(k)) >= T2?
Yes
Recommend components with LA(i)(j)
Definitions:
u : user's input
C(i) : Concept i (1<=i<=n1)
A(i)(j) : Annotation j for Concept i (1<=j<=n2)
LA(i)(j)(k) : Lexically connected string for A(i)(j) (1<=k<=n3)
T : Threshold (0<=T<=1)
S(x-y): the string matching score while matching x against y
No
i=n?
Yes
Stop
n=n+1
No
No
k=k+1
No
k=n3?
Yes
j=j+1
j=n2?
Yes

Figure 21 the flowchart for the situation that the interpreter finds a concept in the ontology

In general, we can select any string matching algorithms for matching strings, such as JaroWinklerTFIDF (Jaro, 1989, 1995; Winkler, 1999), TFIDF (Jones, 1972) and UnsmoothedJS (Jaro, 1989, 1995; Winkler, 1999),We use JaroWinklerTFIDF to demonstrate the idea.

C-FOAM is also not limited to certain lexical matching algorithms as well. In this article, we use WordNet as the lexical dictionary and the WordNet Synset relations for matching.

As illustrated in Figure 21, when JaroWinklerTFIDF cannot find a concept that is annotated by a similar term, C-FOAM loads the lexically connected terms of all the annotated terms of this concept. Then, it uses the following equation to calculate the similarity score.

$$S_{t_1-t_3} = W_{relatioType} \times S_{t_1-t_2}$$

Given two strings $t_1 t_1$ and t_2 t_2, the value $S_{t_1\text{-}t_2}$ $S_{t_1\text{-}t_2}$ is the matching score generated by JaroWinklerTFIDF when comparing $t_1 t_1$ to t_2 t_2; $w_{relationType}$ $w_{relationType}$ is a weight that depends on the type of the lexical relation. We use an SDT to monitor the setting of $w_{relationType}$ $w_{relationType}$.
Table 3 shows an example of such SDTs. The first SDT commitment in DECOL defines the value type of Weight as Float and its value range is $[0,1]$ $[0,1]$. The second one indicates that, if the relation type is "Antonym", then the weight must be 0. The last one expresses a rule – if the relation type is "Holonym", then the weight but be the minimum nonnegative value among all the weights.
Table 3 an SDT that decides $w_{relationType} w_{relationType}$ based on the WordNet relation types

Condition	1	2	3	4	5	6	7
Relation Type	Antonym	Synonym	Holonym	Hypernym	Hyponym	Instance	Meronym
Action							
Weight	0	0.9	0.1	0.9	0.5	0.6	0.2
SDT Commitments in DECOL							
1	(P1= [Weight, has, is of, Value], P2 = [Weight, has value type, is value type of, Float]): P1 (Value)>=0, P1 (Value) <=1.						
2	(P1 = [Weight, has, is of, Value], P2 = [Concept, has, is of, Relation Type]): IMPLIES (P2 (Relation Type) = "Antonym", P1 (Value) =0).						
3	(P1 = [Weight, has, is of, Value], P2 = [Concept, has, is of, Relation Type]): IMPLIES (P2 (Relation Type) = "Holonym", MIN_POS (P1 (Value))).						

For example, when we compare "bringing" to "bring" using JaroWinklerTFIDF, we get $S_{bringing\text{-}bring} =$ $S_{bringing\text{-}bring}$ $=$0.925. The word "bring" is connected with its synonym "fetch". According to Table 3, $w_{Synonym} w_{Synonym}$ = 0.9 (see column 2 in Table 3). We get a similarity score $S_{bringing\text{-}fetch} = 0.925 \times 0.9 = 0.8325$ $S_{bringing\text{-}fetch} = 0.925 \times 0.9 = 0.8325$.
If we set the threshold lower than this score, then DIY-CDR will return a list of 86075[10] recommended components annotated with "fetch". The list contains, for example, the pipe with the title "Add Feed Label to Each Item Title", which has a module called "fetch".

Situation two: If the result of the interpreter is a context, which corresponds to a set of lexons, then C-FOAM will consult a graph matching algorithm. Let us look at the following example. If the input is "finding place of news", which is a requirement from an end user of DIY-CDR, then the string matching algorithm will find the contexts identified with "news location finder", "news producer finder" and "TV news reader". Then, C-FOAM finds that "location" is the synonym of "place" using the lexical matching algorithm. It will select "news location finder" as the output.

The requirement "news location finder" is annotated with the lexons shown as in Table 4. Afterwards, C-FOAM will select a graph matching algorithm for matching the graph that contains the lexons from Table 3 with the graphs that contain the annotations from all the components. An example of an annotated component is shown in Table 5. It is called "RSS 2 Geo".

Table 4 a lexon table that contains the annotation of "news location finder"

ID	t_1	r_1	r_2	t_2
L1_1	Person	use	is used by	Newspaper
L1_2	Person	execute	is executed by	Read
L1_3	Person	execute Task during	Task is executed by during	Time span
L1_4	Daytime	is a	supertype of	Time span
L1_5	Morning	is a	supertype of	Time span
L1_6	News	is of	has	Newspaper
L1_7	News	has	is of	Location
L1_8	Person	use	is used by	Map
L1_9	Map	has	is of	Location

Table 5 a lexon table that contains the annotation of "RSS 2 Geo"

[10] The data is retrieved on Nov. 20, 2010 (http://pipes.yahoo.com/pipes/search?r=module:fetch)

ID	t_1	r_1	r_2	t_2
L2_1	Yahoo pipe	has	is of	Pipe title
L2_2	RSS 2 Geo	is an instance of	has instance	Pipe title
L2_3	Person	use	is used by	webservice
L2_4	Geoname webservice	is a	supertype of	webservice
L2_5	Person	execute	is executed by	Add
L2_6	Person	use	is used by	RSS
L2_7	Person	execute	is executed by	Creation
L2_8	Person	use	is used by	GeoRSS
L2_9	GeoRSS	is a	supertype of	RSS feed
L2_10	Person	use	is used by	Result
L2_11	Person	use	is used by	Map

We use Lexon Matching Algorithm (LexMA) as an example of graph matching algorithms to demonstrate C-FOAM. It is explained as follows.

Suppose we have two graphs G_1 G_1, which contains $n_1 n_1$lexons, and$G_2 G_2$, which contains $n_2 n_2$ lexons. We use $\cdot$ $\cdot$ to indicate the source of a lexon. For example, $G_1 \cdot l_1 G_1 \cdot l_1$ is a lexon from G_1 G_1. Two lexons can have four different relations.

The first one is **equivalence**. We consider $l_1 l_1$ and $l_2 l_2$ equivalent and note it as "$l_1 = l_2 l_1 = l_2$" if it can be deduced using the following formula.

$(l_1 \cdot t_1 = l_2 \cdot t_1) \cap (l_1 \cdot r_1 = l_2 \cdot r_1) \cap (l_1 \cdot r_2 = l_2 \cdot r_2) \cap (l_1 \cdot t_2 = l_2 \cdot t_2) \cup (l_1 \cdot t_2 = l_2 \cdot t_1) \cap (l_1 \cdot r_2 = l_2 \cdot r_1) \cap (l_1 \cdot r_1 = l_2 \cdot r_2) \cap (l_1 \cdot t_1 = l_2 \cdot t_2) \rightarrow l_1 = l_2 (l_1 \cdot t_1 = l_2 \cdot t_1) \cap (l_1 \cdot r_1 = l_2 \cdot r_1) \cap (l_1 \cdot r_2 = l_2 \cdot r_2) \cap (l_1 \cdot t_2 = l_2 \cdot t_2) \cup (l_1 \cdot t_2 = l_2 \cdot t_1) \cap (l_1 \cdot r_2 = l_2 \cdot r_1) \cap (l_1 \cdot r_1 = l_2 \cdot r_2) \cap (l_1 \cdot t_1 = l_2 \cdot t_2) \rightarrow l_1 = l_2$

.

The second one is **inequality**. We consider $l_1 l_1$ and $l_2 l_2$ unequal and note it as "$l_1 \neq l_2 l_1 \neq l_2$" if it can be deduced using the following formula.

$(l_1 \cdot t_1 \neq l_2 \cdot t_1) \cap (l_1 \cdot t_2 \neq l_2 \cdot t_1) \cap (l_1 \cdot t_1 \neq l_2 \cdot t_2) \cap (l_1 \cdot t_2 \neq l_2 \cdot t_2) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_2) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_2) \rightarrow l_1 \neq l_2 . (l_1 \cdot t_1 \neq l_2 \cdot t_1) \cap (l_1 \cdot t_2 \neq l_2 \cdot t_1) \cap (l_1 \cdot t_1 \neq l_2 \cdot t_2) \cap (l_1 \cdot t_2 \neq l_2 \cdot t_2) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_2) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_2) \rightarrow l_1 \neq l_2 .$

The third one records the situation of "same vertex, different edges" (**very similar**). We say $l_1 l_1$ is very similar to $l_2 l_2$ and note it as $l_1 \cong l_2$ $l_1 \cong l_2$ if it can be deduced using the following formula.

$(l_1 \cdot t_1 = l_2 \cdot t_1) \cap (l_1 \cdot t_2 = l_2 \cdot t_2) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_2) \cup (l_1 \cdot t_1 = l_2 \cdot t_2) \cap (l_1 \cdot t_2 = l_2 \cdot t_1) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_2) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_1) \rightarrow l_1 \cong l_2 (l_1 \cdot t_1 = l_2 \cdot t_1) \cap (l_1 \cdot t_2 = l_2 \cdot t_2) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_1) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_2) \cup (l_1 \cdot t_1 = l_2 \cdot t_2) \cap (l_1 \cdot t_2 = l_2 \cdot t_1) \cap (l_1 \cdot r_1 \neq l_2 \cdot r_2) \cap (l_1 \cdot r_2 \neq l_2 \cdot r_1) \rightarrow l_1 \cong l_2$

The last relation is to describe the situation of "connected with one vertex" (**connected**), which we indicated as $l_1 \sim l_2$ $l_1 \sim l_2$. The formula is illustrated as:

$\neg(l_1 = l_2) \cap \neg(l_1 \neq l_2) \cap \big((l_1 \cdot t_1 = l_2 \cdot t_1) \cup (l_1 \cdot t_1 = l_2 \cdot t_2) \cup (l_1 \cdot t_2 = l_2 \cdot t_1) \cup (l_1 \cdot t_2 = l_2 \cdot t_2)\big) \rightarrow l_1 \sim l_2 \neg(l_1 = l_2) \cap \neg(l_1 \neq l_2) \cap \big((l_1 \cdot t_1 = l_2 \cdot t_1) \cup (l_1 \cdot t_1 = l_2 \cdot t_2) \cup (l_1 \cdot t_2 = l_2 \cdot t_1) \cup (l_1 \cdot t_2 = l_2 \cdot t_2)\big) \rightarrow l_1 \sim l_2$

The similarity score of comparing $G_1 G_1$ and $G_2 G_2$ is calculated using the following equation.

$$S_{G_1-G_2} = w_1 \times \frac{m_{G_1 \cdot l = G_2 \cdot l}}{n_{G_1} + n_{G_2} - m_{G_1 \cdot l = G_2 \cdot l}} + w_2 \times \frac{m_{G_1 \cdot l \cong G_2 \cdot l}}{n_{G_1} + n_{G_2} - m_{G_1 \cdot l \cong G_2 \cdot l}} + w_3 \times \frac{m_{G_1 \cdot l \sim G_2 \cdot l}}{n_{G_1}} + w_4 \times \frac{m_{G_1 \cdot l \sim G_2 \cdot l}}{n_{G_2}}$$

The parameters w_1 w_1, w_2 ,w_3 w_2 ,w_3 and w_4 w_4 are the Real number weights
(
$w_1, w_2, w_3, w_4 \in R)$, where $0 \le w_1, w_2, w_3, w_4 \le 1 w_1, w_2, w_3, w_4 \in R)$, where $0 \le$
$w_1, w_2, w_3, w_4 \le 1$
, and $w_1 + w_2 + w_3 + w_4 = 1$ $w_1 + w_2 + w_3 + w_4 = 1$. n_{G_1} $n_{G_1}$is the total number of the lexons in G_1 G_1; n_{G_2} $n_{G_2}$is the total number of the lexons in G_2 G_2.
The value $m_{G_1 \cdot l = G_2 \cdot l}$ $m_{G_1 \cdot l = G_2 \cdot l}$ is the size of a subset of G_1 G_1. We denote this subset as $G_{G_1 \cdot l = G_2 \cdot l}$ $G_{G_1 \cdot l = G_2 \cdot l}$, which is defined
as
$\{\forall G_{G_1 \cdot l = G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_{G_1 \cdot l = G_2 \cdot l} \cdot l_i = G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l = G_2 \cdot l} \subseteq G_1\}$ $\{\forall G_{G_1 \cdot l = G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_{G_1 \cdot l = G_2 \cdot l} \cdot l_i = G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l = G_2 \cdot l} \subseteq G_1\}$
. The value $m_{G_1 \cdot l \cong G_2 \cdot l}$ $m_{G_1 \cdot l \cong G_2 \cdot l}$ is the size of a subset of G_1 G_1, which we denote as $G_{G_1 \cdot l \cong G_2 \cdot l}$ $G_{G_1 \cdot l \cong G_2 \cdot l}$. It is defined
as
$\{\forall G_{G_1 \cdot l \cong G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_1 \cdot l_i \cong G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l \cong G_2 \cdot l} \subseteq G_1\}$ $\{\forall G_{G_1 \cdot l \cong G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_1 \cdot l_i \cong G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l \cong G_2 \cdot l} \subseteq G_1\}$
. The value $m_{G_1 \cdot l \sim G_2 \cdot l}$ $m_{G_1 \cdot l \sim G_2 \cdot l}$ is the size of a subset of G_1 G_1, which we denote as $G_{G_1 \cdot l \sim G_2 \cdot l}$ $G_{G_1 \cdot l \sim G_2 \cdot l}$ and defined
as
$\{\forall G_{G_1 \cdot l \sim G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_1 \cdot l_i \sim G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l \sim G_2 \cdot l} \subseteq G_1\}$ $\{\forall G_{G_1 \cdot l \sim G_2 \cdot l} \cdot l_i \exists G_2 \cdot l_j | G_1 \cdot l_i \sim G_2 \cdot l_j, 1 \le j \le n_{G_2}, G_{G_1 \cdot l \sim G_2 \cdot l} \subseteq G_1\}$
.

We use an SDT to monitor the values of w_1, w_2, w_3 w_1, w_2, w_3 and w_4 w_4 (see Table 6). For example, if G_1 $G_1$is the annotation set as shown in Table 4 and G_2 G_2 is the annotation set as shown in Table 5. The total number of the lexons in G_1 G_1 is 9 ($n_{G_1} = 9$ $n_{G_1} = 9$) and the number of the lexons in G_2 G_2 is 11 ($n_{G_2} = 11$ $n_{G_2} = 11$). The number of equivalent lexons is 1 ($m_{G_1 \cdot l = G_2 \cdot l} = 1$ $m_{G_1 \cdot l = G_2 \cdot l} = 1$, the lexon with ID L1_8 in Table 4 is equivalent to the lexon with ID L2_11 in Table 5). We do not have any lexons that have same vertex and different edges ($m_{G_1 \cdot l \cong G_2 \cdot l} = 0$ $m_{G_1 \cdot l \cong G_2 \cdot l} = 0$). The number of the lexons from G_1 G_1 that are connected with G_2 G_2 is 5 ($m_{G_1 \cdot l \sim G_2 \cdot l} = 5$ $m_{G_1 \cdot l \sim G_2 \cdot l} = 5$). The number of the lexons from G_2 G_2 that are connected with G_1 $G_1$is 7 ($m_{G_2 \cdot l \sim G_1 \cdot l} = 7$ $m_{G_2 \cdot l \sim G_1 \cdot l} = 7$). If we take the weights defined by a “balanced” profile (see column 3 in Table 6), then we will get the similarity
score
$S_{G_1 \cdot G_2} = 0.25 \times \frac{1}{9+11-1} + 0.25 \times \frac{0}{1+11-0} + 0.25 \times \frac{5}{9} + 0.25 \times \frac{7}{11} \approx 0.3112$ $S_{G_1 \cdot G_2} = 0.25 \times \frac{1}{9+11-1} + 0.25 \times \frac{0}{1+11-0} + 0.25 \times \frac{5}{9} + 0.25 \times \frac{7}{11} \approx 0.3112$
.
Table 6 an SDT that decides w_1, w_2, w_3 w_1, w_2, w_3 and w_4 w_4 based on profiles

Condition	1	2	3	4
Profile	Optimistic	Pessimistic	Balanced	Customized
Action				
w_1	1	0.1	0.25	0.8
w_2	0	0.1	0.25	0.2
w_3	0	0.4	0.25	0
w_4	0	0.4	0.25	0
SDT Commitments in DECOL				
1	(P1= [Weight, has, is of, Value], P2 = [Weight, has value type, is value type of, Float]): P1 (Value)>=0, P1 (Value) <=1.			
2	(P1 = [Weight, has, is of, Value]): P1 (Weight) = {w_1, w_2, w_3, w_4 w_1, w_2, w_3, w_4},			

$w_1 + w_2 + w_3 + w_4 = 1 w_1 + w_2 + w_3 + w_4 = 1$.

If this score is larger than the threshold, then the final result of DIY-CDR (a list of recommended components) will contain this component – the Yahoo Pipe with the title “RSS 2 Geo”.
The flowchart that is used for situation two is illustrated in Figure 22.

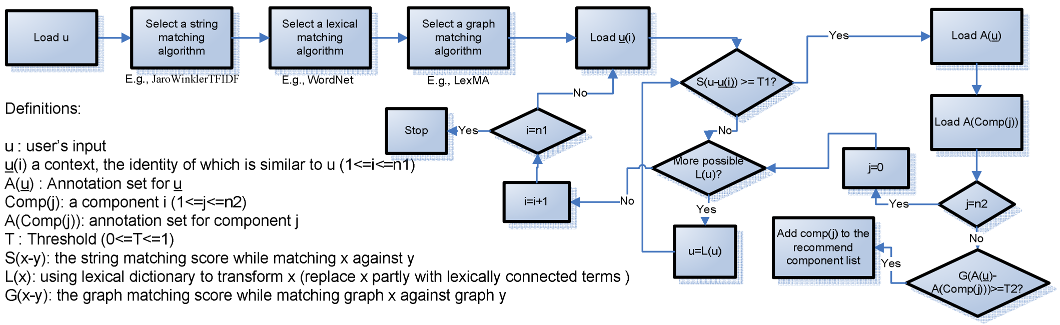

Load u
Select a string matching algorithm
E.g., JaroWinklerTFIDF
Select a lexical matching algorithm
E.g., WordNet
Select a graph matching algorithm
E.g., LexMA
Load u(i)
S(u-u(i)) >= T1?
Yes
Load A(u)
Load A(Comp(j))
No
Stop
Yes
i=n1
No
More possible L(u)?
j=0
i=i+1
No
Yes
Yes
j=n2
No
u=L(u)
Add comp(j) to the recommend component list
Yes
G(A(u)-A(Comp(j)))>=T2?
Definitions:
u : user's input
u(i) a context, the identity of which is similar to u (1<=i<=n1)
A(u) : Annotation set for u
Comp(j): a component i (1<=j<=n2)
A(Comp(j)): annotation set for component j
T : Threshold (0<=T<=1)
S(x-y): the string matching score while matching x against y
L(x): using lexical dictionary to transform x (replace x partly with lexically connected terms)
G(x-y): the graph matching score while matching graph x against graph y

Figure 22 the flowchart for the situation that the interpreter finds a context in the ontology

Compared with the previous version of C-FOAM, we have extended it with the following items.

- **SDT**. We use SDTs to tune the parameters in the lexical matching algorithm and the graph matching algorithm. By doing so, we can provide a *flexible* way to end users for using the algorithms. In addition, SDTs can ensure the *correctness* of the decision rules in a decision table using meta-rules stored in an ontology. For instance, if a user wants to update Table 6 and set the profile of “customized” with the following weights – $w_1 = 0.2 w_1 = 0.2$, $w_2 = 0.3 w_2 = 0.3$, $w_3 = 0.5 w_3 = 0.5$ and $w_4 = 0.5 w_4 = 0.5$ – then the SDT parser will return a message containing “it is not possible because the number of $w_1 + w_2 + w_3 + w_4 w_1 + w_2 + w_3 + w_4$ must be 1”.
- **Context ontology**. The previous version of C-FOAM is not based on any context ontologies. On the one hand, it provides the freedom to the ways on how concepts are defined in the domain ontologies. But on the other hand, many of the relations (also called (co-)roles in this report) in the annotation sets of lexons cannot be interpreted. And, concepts are defined randomly and many of them are useless, which also gives difficulties to the matching engine. In this version, we create the domain ontologies based on a context ontology. The relations between concepts are *restricted* and *interpreted* by the matching engine. The concepts in the domain ontologies are also created and connected using limited relations. By doing so, C-FOAM can easily discover concepts. Note that Onto-DIY supports ontology versioning. When a new relation is added, the ontology versioning server will suggest to be treated as the domain canonical relations and the ontology engineers will have to interpret and implement them in DIY-CDR.
- **A Scenario for Single Concept Discovering**. We have illustrated two situations. The first situation is to discover single concept, which was not considered in C-FOAM of the previous version.
- **LexMA**. In this report, we have designed a graph matching algorithm called LexMA for C-FOAM. LexMA transforms the graph matching problem into a lexon matching problem. We have also used GRASIM (Tang, 2010) as a graph matching algorithm in C-FOAM. Compared to GRASIM, the performance of LexMA is higher.
- **DIY-CDR**. DIY-CDR is an application of C-FOAM. It uses C-FOAM for recommending and discovering components. The end users of DIY-CDR can be technicians, Pro-Am and non-technical people. DIY-CDR is used after DIY requirements are analyzed and before the end user starts building his own solutions.

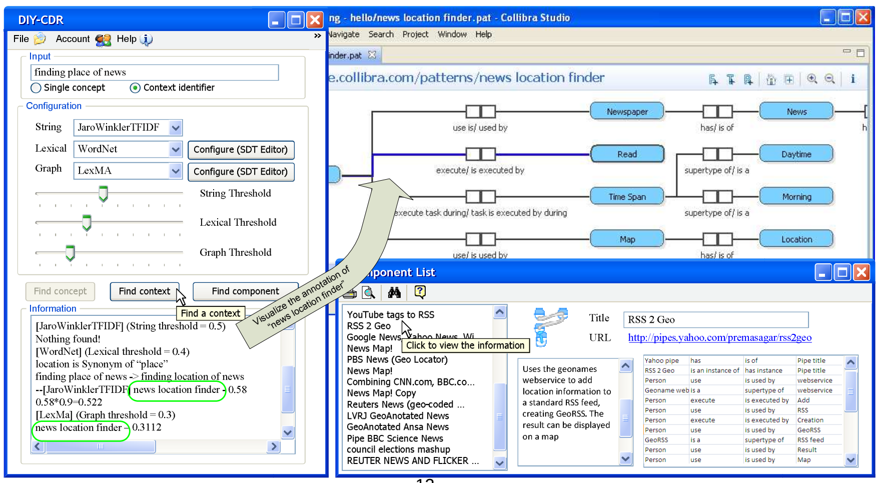
Note that we have illustrated examples of recommending *software* components in this report. We can also use DIY-CDR to recommend *hardware* components as long as they are properly annotated with the domain ontologies.
Note also that, by following the same procedures and principles, DIY-CDR can as well be used to *find similar DIY solutions* and *group* them together if necessary.

In this section, we have discussed how to use C-FOAM for DIY-CDR. C-FOAM contains one string matching algorithm, one lexical matching algorithm and one graph matching algorithm. We want to emphasize that C-FOAM and DIY-CDR are not restricted to certain matching algorithms, although we use JaroWinklerTFIDF, WordNet and LexMA to demonstrate C-FOAM in this report. We have also demonstrated how to use SDT to tune the algorithms in C-FOAM.

In the next section, we will illustrate the implementation issues of DIY-CDR.

3.4.4 Implementation and Results

The prototype contains user friendly interfaces for technicians (or Pro-Am) and non-technical end users. Figure 23 shows the interface design and screenshots of DIY-CDR. It is implemented in Java J2EE using Eclipse SDK[11].



[JaroWinklerTFIDF] (String threshold = 0.5)
Nothing found!
[WordNet] (Lexical threshold = 0.4)
location is Synonym of "place"
finding place of news -> finding location of news
--[JaroWinklerTFIDF] news location finder - 0.58
0.58*0.9=0.522
[LexMa] (Graph threshold = 0.3)
news location finder – 0.3112

YouTube tags to RSS
RSS 2 Geo
Google News, Yahoo News, Wi
News Map!
PBS News (Geo Locator)
News Map!
Combining CNN.com, BBC.co...
News Map! Copy
Reuters News (geo-coded ...
LVRJ GeoAnotated News
GeoAnotated Ansa News
Pipe BBC Science News
council elections mashup
REUTER NEWS AND FLICKER ...

Title: RSS 2 Geo
URL: http://pipes.yahoo.com/premasagar/rss2geo

Uses the geonames webservice to add location information to a standard RSS feed, creating GeoRSS. The result can be displayed on a map

Yahoo pipe	has	is of	Pipe title
RSS 2 Geo	is an instance of	has instance	Pipe title
Person	use	is used by	webservice
Geoname web	is a	supertype of	webservice
Person	execute	is executed by	Add
Person	use	is used by	RSS
Person	execute	is executed by	Creation
Person	use	is used by	GeoRSS
GeoRSS	is a	supertype of	RSS feed
Person	use	is used by	Result
Person	use	is used by	Map

Figure 23 DIY-CDR user interfaces[12] (see Table 4 for a clearer view on the lexons in the ontology and Table 5 for a component)

In the left window frame (Figure 23), a user can search components linked to either a concept or a context. In the configuration group, he can select three algorithms: one for string matching[13], one for lexical matching and one for graph matching. He can call the SDT editor (Figure 24) to tune the parameters for the lexical matching and graph matching algorithms. We use three thresholds, each of which is to filter out the (internal) outputs that have low matching scores. In the information group, the text editor shows the processing information.

When a concept or a context is found, the user can use the Collibra studio[14] to browse the concept definition or the context definition in the ontology. The screenshot of Collibra studio is shown on the right-upper side of Figure 23. These definitions are modeled in ORM/ORM2 (Halpin, 2001), can be stored and published in OWL/RDF(s)[15]. For example, the context "news location finder", for the lexons of which we refer to Table 4, is published in OWL (see what follows).

```
…
<owl:Ontology rdf:about=""/>
<owl:Class rdf:about="#Daytime">
      <rdfs:subClassOf
rdf:resource="#TimeSpan"/>
 </owl:Class>
 <owl:Class rdf:about="#Location"/>
 <owl:Class rdf:about="#Map"/>
<owl:ObjectProperty rdf:about="#execute">
      <rdfs:domain rdf:resource="#Person"/>
    <owl:inverseOf
rdf:resource="#is_executed_by"/>
 </owl:ObjectProperty>
<owl:ObjectProperty
rdf:about="#execute_Task_during">
 <rdfs:domain rdf:resource="#Person"/>
```

[11] http://www.eclipse.org/
[12] Check http://www.starlab.vub.ac.be/website/DIY-CDR_demo for its demonstration
[13] Concerning the string matching algorithms, we have reused the implementation from the SecondString project **Error! Reference source not found.**.
[14] Collibra is a spinoff company from VUB STARLab. Collibra Studio, which is one of its main products, allows the creation of business semantics models that provide unambiguous definitions, identifications and mappings towards existing data sources.
[15] http://www.w3.org/TR/owl-ref/

```
 <owl:Class rdf:about="#Morning">
 <rdfs:subClassOf
rdf:resource="#Daytime"/>
</owl:Class>
 <owl:Class rdf:about="#News"/>
 <owl:Class rdf:about="#Newspaper">
 <rdfs:subClassOf
rdf:resource="&owl;Thing"/>
 </owl:Class>
 <owl:Class rdf:about="#Person"/>
 <owl:Class rdf:about="#Read">
<rdfs:subClassOf rdf:resource="#Task"/>
 </owl:Class>
 <owl:Class rdf:about="#Task"/>
 <owl:Class rdf:about="#TimeSpan"/>
<owl:Class rdf:about="&owl;Thing"/>

<owl:ObjectProperty
rdf:about="#Task_is_executed_by_during">
       <rdfs:domain
rdf:resource="#TimeSpan"/>
 </owl:ObjectProperty>
```

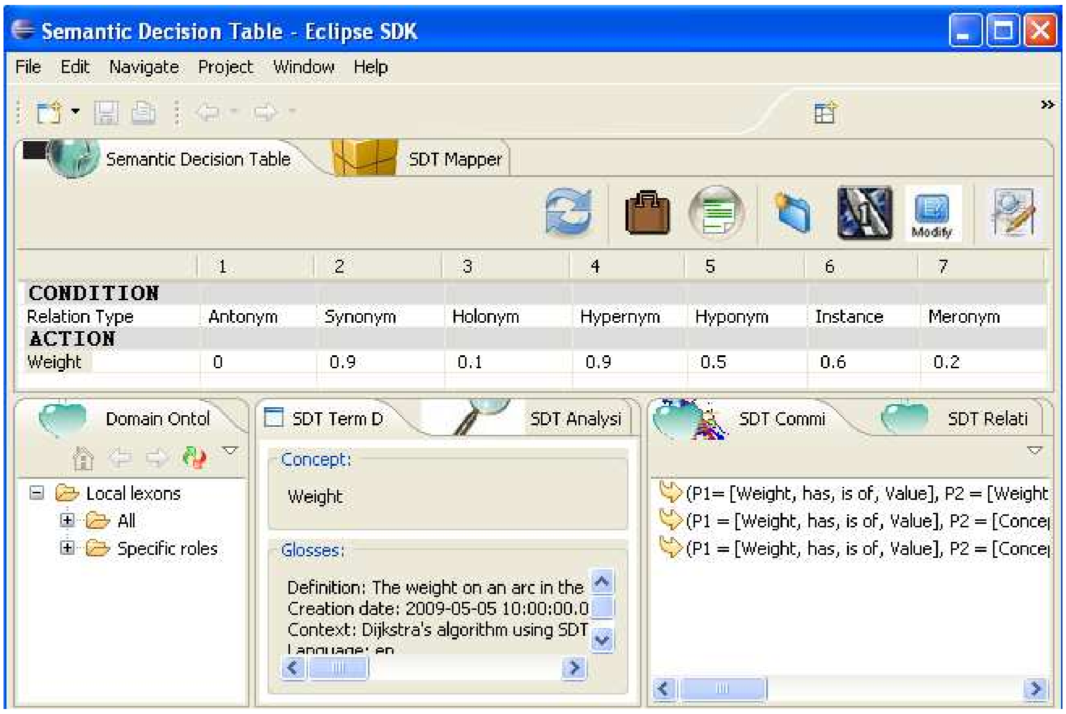
```
 <owl:inverseOf
rdf:resource="#Task_is_executed_by_during"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#has">
   <rdfs:domain rdf:resource="#Map"/>
    <rdfs:domain rdf:resource="#News"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty
rdf:about="#is_executed_by">
       <rdfs:domain rdf:resource="#Read"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#is_of">
       <rdfs:domain rdf:resource="#Location"/>
       <owl:inverseOf rdf:resource="#has"/>
 </owl:ObjectProperty>
 <owl:ObjectProperty rdf:about="#is_used_by">
       <rdfs:domain rdf:resource="#Map"/>
       <rdfs:domain
rdf:resource="#Newspaper"/>
 </owl:ObjectProperty>
   <owl:ObjectProperty rdf:about="#use">
 <rdfs:domain rdf:resource="#Person"/>
       <owl:inverseOf
rdf:resource="#is_used_by"/>
 </owl:ObjectProperty>
…
```

Readers of this article may reuse existing ontology engines, such as ΩΩ-RIDL (Trog et al., 2007), Jena[16], SPARQL[17] and Racer (Haarslev and Möller, 2003), to query and reason our ontologies.

The right-down window frame (Figure 23) shows the list of recommended components. When one component is selected, its image, title, URL, description and lexon table (the annotation) will be shown.

Note that DIY-CDR also supports non-technical users. He can enter an input in the text field (in the left window frame in Figure 23) and click on the button “find component”. The list of suggested components will be shown on the right-down window frame (Figure 23).

[16] http://jena.sourceforge.net/
[17] http://www.w3.org/TR/rdf-sparql-query/



	1	2	3	4	5	6	7
CONDITION							
Relation Type	Antonym	Synonym	Holonym	Hypernym	Hyponym	Instance	Meronym
ACTION							
Weight	0	0.9	0.1	0.9	0.5	0.6	0.2

Concept:
Weight

Glosses:
Definition: The weight on an arc in the
Creation date: 2009-05-05 10:00:00.0
Context: Dijkstra's algorithm using SDT
Language: en

(P1= [Weight, has, is of, Value], P2 = [Weight
(P1 = [Weight, has, is of, Value], P2 = [Conce
(P1 = [Weight, has, is of, Value], P2 = [Conce

Figure 24 a screenshot of the SDT editor

Figure 24 shows an SDT plug-in framework that contains the plug-ins of a decision table editor (see the upper window in Figure 24), an SDT engine for analyzing the correctness and completeness of an SDT (in the hidden tab of the middle view in Figure 24), a domain ontology viewer (see the window in the left-down corner of Figure 24), a glossary viewer/editor and an SDT (see the middle view in Figure 24) and an SDT commitment editor/DECOL editor (see the window in the right-down corner of Figure 24).

We have also used GRASIM (Tang, 2010) as a graph matching algorithm in C-FOAM. Compared to GRASIM, LexMA consumes much less. The environment settings of our test are as follows: Intel Core 2 Duo CPU, P9400 @2.40GHZ, 2.37 GHz, 2.45 GB of RAM, Windows XP v 2002, Service Pack 3, JVM 16 M. We use the data set in (Tang, 2010) to run LexMA. The biggest lexon set in G_1G_1 contains 84 lexons; and G_1G_1 has 39 lexons on average. The biggest lexon set in G_2 G_2 has 47 lexons; and G_2G_2 has 15 lexons on average.

The maximum cost of running LexMA is 3765 milliseconds; and the average cost is 562 milliseconds. The maximum cost of running GRASIM is 23657 milliseconds; and the average cost is 13213 milliseconds. On average, LexMA is about 24 times faster than GRASIM.

In this section, we have illustrated the implementation issues of DIY-CDR. And we have run a test for LexMA, which is the graph matching algorithm used in DIY-CDR.

4 Semantic Web Services

As detailed in [Passin04], for the Semantic Web, semantic indicates that the meaning of data on the Web can be discovered not just by people, but also by computers. In contrast, most meaning on the Web today is inferred by people who read web pages and the labels of hyperlinks, and by other people who write specialized software to work with the data. Semantic Web stands for a vision in which computers as well as people can find, read, understand, and use data over the World Wide Web to accomplish useful goals for users. [Auer07] as an example tries to extract the meaning of web pages to create a database from knowledge, then a search engine will be able to parse this generated database and reply to user's query. Other simplified web services are already available in [XMethods] for example.

Semantic Web is introduced as there are similarities with multimedia context representation and multimedia analyzers. In fact, we are once again facing in World Wide Web knowledge which can be considered as an agglomeration databases, ontologies… and web services have to deal with data at different level of granularity.

A Web service is a software component identified by a URI which is accessible via standard network protocols such as but not limited to Simple Object Access Protocol (SOAP) over HTTP.
Figure 24 introduces to main Web Service specifications and tools:

- · Discovering using [UDDI] to discover web services location and activities.
- · Description using [WSDL] to describe a web service and how to interact with it.
- · Packaging using SOAP to package interaction with the Web Service -
- · Transport using HTTP or TCP/IP to carry the data envelope across the internet.

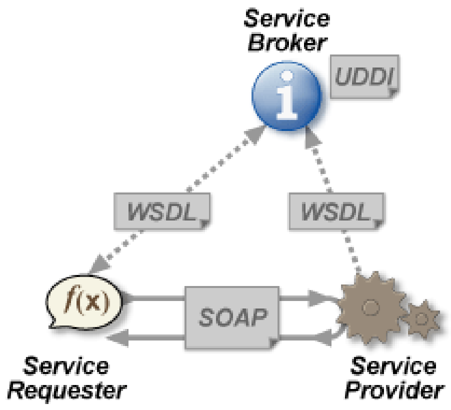


Figure 24: Synoptic of Web Service Architecture [Juszczyk05]

Semantic web provide solutions as detailed in [Herrmann07] to identify pertinent services and to chain them accordingly to their inputs and outputs through specifications like [WSDL-S, OWL-S, WS-BPEL …]. A recent overview on semantic Web Service (SWS) is presented in [Wu08]. The main goal of both OWL-S and WSDL-S is to establish a framework within which service descriptions are made and shared.

4.1 WSDL-S

The Web Service Description Language (WSDL) is an XML format document which describes the Web Service. WSDL-S stands for Web Service Description Language – Semantic. It extends WSDL in order to use semantic capabilities of OWL to provide semantically enriched meanings of service descriptions as detailed in [Herrmann07].

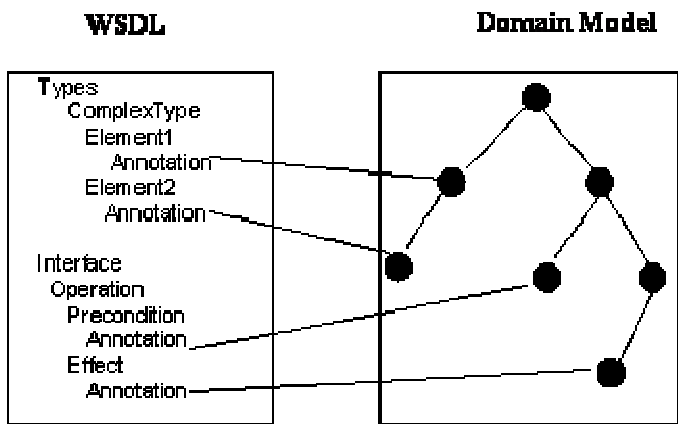


Figure 25: Association of semantics to WSDL elements [WSDL-S]

As detailed in [WSDL-S], Figure 25 shows how semantic annotations are associated with various elements of a WSDL document (including inputs, outputs and functional aspects like operations, preconditions and effects) by referencing the semantic concepts in an external domain semantic model. The domain model can consist of one or more ontologies.

4.2 OWL-S

OWL-S stands for Web Ontology Language for Services (or Ontology Web Language for Services). Following the layered approach to markup language development, OWL-S is built on the Ontology Web Language (OWL) Recommendation produced by the Web-Ontology Working Group at the World Wide Web Consortium.
As introduced in Figure 26, it is an ontology that enables automatic service discovery, invocation, composition and execution monitoring. Composition is based on pre- and postconditions.

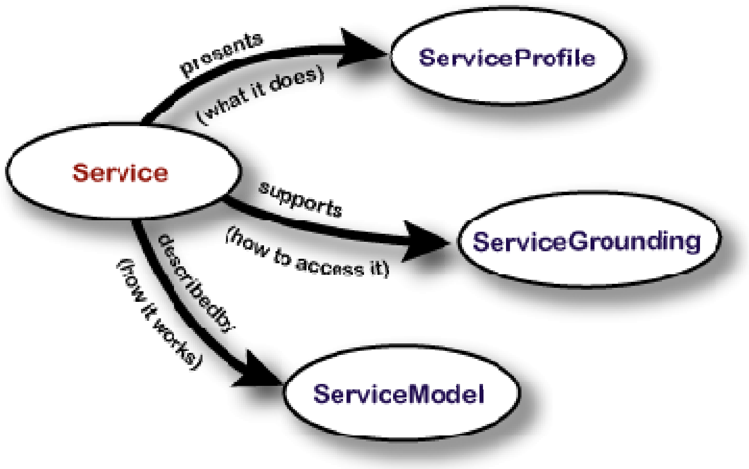


Figure 26: Top level of the service ontology [OWL-S]

5 Conclusion

We have discussed the need and method of modeling space semantics in the form of ontologies. After we have developed the domain ontologies separately and independently by different partners, we tried to manage them using ontology alignment and matching methods. The idea of DiYSEMESS (meaning evolution support system) has been raised and designed, yet not fully applied to real case scenario due to the fact that 1) the structure of our current ontologies are rather simple; 2) the DiYSEMESS portal is not easily used by domain experts; 3) we do not have much time for the experiments. Nevertheless, partners begin to understand the need to aligning their ontologies and insights in the domain for potential usages, such as components discovering across domain.

We have also presented an interesting tool called Ontology-based Do-It-Yourself Component Discoverer and Recommender (DIY-CDR). It has been designed and implemented by the Belgian sub-consortium. It is to be used in the Onto-DIY system, which allows users to create their own applications using their own evolving semantics. DIY-CDR delivers recommendations that match users' preferences, needs and hopes at the right moment, in the right place and on the right media. It supports both technical and non-technical end users. Although DIY-CDR is designed to support the Onto-DIY architecture, it can be used for the purpose of knowledge discovery in general.

DIY-CDR uses an ontology-based matching strategy for discovering and recommending components. DIY-CDR can as well be used to find similar DIY solutions and group them together if necessary. DIY-CDR is used right after a user has a rough idea on what to build and before starting building his DIY solutions. We have extended the C-FOAM matching strategy, which contains algorithms at the levels of string matching, lexical matching and graphical matching. We have designed a graph matching algorithm called LexMA and a simple lexical matching algorithm using WordNet. We use SDTs to set up the parameters for LexMA and the lexical matching algorithm.

DIY-CDR is not restricted to particular matching algorithms. We have illustrated examples of recommending software components in this article. We can also use DIY-CDR to recommend hardware components as long as they are properly annotated with the domain ontologies.

6 References

[Klein04] M. Klein, “Change Management for Distributed Ontologies”, PhD thesis, Vrije Universiteit Amsterdam, 2004.

[Lacot05] X. Lacot, Introduction à OWL, un langage XML d'ontologies Web, Juin 2005.

[Antoniou04] G. Antoniou, F. Van Harmelen, A semantic Web Primer, MIT Press, April 2004, ISBN 0-262- 01210-3

[Passin04] T. B. Passin, Explorer's Guide to the semantic WEB, Manning Publications Co., p.304, ISBN 978-1-932394-20-6, March 2004.

[Bray98] T. Bray, RDF and Metadata, xml.com, June 09, 1998,internet web page, http://www.xml.com/pub/a/98/06/rdf.html, retrieved on August 2009.

[Tauberer06] J. Tauberer, What Is RDF, xml.com, July 26, 2006, internet web page, http://www.xml.com/pub/a/2001/01/24/rdf.html, retrieved on August 2009.

[Brickley07] D. Brickley, L. Miller, FOAF Vocabulary Specification 0.91, Namespace Document 2 November 2007 - OpenID Edition, internet web page, http://xmlns.com/foaf/spec/, retrieved on August 2009.

[RDFS_Wiki] RDF Schema web page on Wikipedia, Wikipedia®, Internet web site, http://en.wikipedia.org/wiki/RDF_schema, retrieved on August 2009.

[OWL04] OWL Web Ontology Language Overview, W3C, W3C Recommendation 10 February 2004, internet web site, http://www.w3.org/TR/owl-features/, retrieved on July 2009.

[Isaac05] A. Isaac, Conception et utilisation d’ontologies pour l’indexation de documents audiovisuels, Thesis, Paris IV University, Sorbonne, 2005.

[Horridge04] M. Horridge, H. Knublauch, A. Rector, R. Stevens, C. Wroe, A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools - Edition 1.0, The University Of Manchester, August 27, 2004.

[Bechhofer04] S. Bechhofer, R. Volz, WonderWeb OWL Ontology Validator, internet web site, http://www.mygrid.org.uk/OWL/Validator, retrieved on July 2009.

[Smith07] J. R. Smith, The real Problem of Bridging the “Semantic Gap”, Multimedia Content Analysis and Mining (MCAM07), Lecture Notes in Computer Science, Vol. 4577, p.16-17, July 2007.

[Cassell01] J. Cassell, Embodied Conversational Agents: Representation and Intelligence in User Interfaces, AI Magazine, Vol. 22 No. 4, p.67-84, 2001.

[Delezoide06] B. Delezoide, Modèles d’indexation multimédia pour la description automatique de films de cinéma, Ph.D. Thesis, Université Pierre et Marie Curie, Paris, France, April 2006.

[Martin98] K. D. Martin, Y. E. Kim, Musical instrument identification: a pattern-recognition approach, in Proceedings of 136th Meeting of Acoustical Society of America (ASA'98), Vol. 104, Issue 3, p.1768-1768, Norfolk, Va, USA, September 1998.

[Bennett02] P. N. Bennett, S. T. Dumais, E. Horvitz, Probabilistic combination of text classifiers using reliability indicators: Models and results, in Proceedings of the 25th annual international ACM conference on Research and development in information retrieval (SIGIR), p. 207-214, Tampere, Finland, 2002.

[Arndt07] R. Arndt, R. Troncy, S. Staab, L. Hardman, Adding Formal Semantics to MPEG-7: Designing a Well-Founded Multimedia Ontology for the Web, Department of Computer Science, University of Koblenz. Technical Report. January 2007.

[Auer07] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. Ives, DBpedia: A Nucleus for a Web of Open Data, 6th International Semantic Web Conference (ISWC 2007), Busan, Korea, November 2007.

[XMethods] Web Service Provider, internet web site, http://xmethods.com/ve2/index.po, retrieved on August 2009.

[Herrmann07] M. Herrmann, M. A. Aslam, O. Dalferth, Applying Semantics (WSDL, WSDL-S, OWL) in Service Oriented Architectures (SOA), in Proceedings of the 10th International Protégé Conference, Budapest, Hungary, July 15-18, 2007.

[Wu08] C. Wu, V. Potdar, E. Chang, Latent Semantic Analysis – The Dynamics of Semantics Web Services Discovery, Advances in Web Semantics I: Ontologies, Web Services and Applied Semantic Web, 2008, in Chang, E. and Dillon, T. and Meersman, R. and Sycara, K. (ed), Advances in Web Semantics I, p. 346-373. Heidelberg, Germany: Springer.

[WSDL-S] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, K. Verma, Web Service Semantics - WSDL-S, Version 1.0, W3C Member Submission 7 November 2005, internet web page, http://www.w3.org/Submission/WSDL-S/, retrieved on August 2009.

[WSDL] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, internet web page, http://www.w3.org/TR/wsdl, retrieved on August 2009.

[UDDI] Official community gathering place and information resource for the UDDI OASIS Standard, © 1993-2009 OASIS, internet web page, http://uddi.xml.org/, retrieved on August 2009.

Castano, S., Ferrara, A., Hess, G.N. (2006): *Discovery-Driven Ontology Evolution*, Giovanni Tummarello, Paolo Bouquet, Oreste Signore (Eds.): SWAP 2006 - Semantic Web Applications and Perspectives, Proceedings of the 3rd Italian Semantic Web Workshop, Pisa, Italy, 18-20 December, 2006

de Moor, A., De Leenheer, P., Meersman, R. (2006), *DOGMA-MESS: A Meaning Evolution Support System for Interorganizational Ontology Engineering*, Proc. of 14th ICCS conference, Springer-Verlag, V. 4068, p.189-203, Aalborg, Denmark, 2006

De Leenheer, P. and Debruyne, C. (2008): *DOGMA-MESS: a tool for fact-oriented collaborative ontology evolution*, in proc. Of OTM 2008 workshops, Meersman, Tari and Herrero (Eds.), LNCS 5333, pp. 797-806, Srpinger-Verlag Berlin Heidelberg, 2008

Dey, A. K. (2000) Providing architectural support for building context aware applications, Doctoral dissertation, College of Computing, Georgia Institute of Technology

Flouris, G., Plexousakis, D., Antoniou, G. (2006): *A Classification of Ontology Changes. In: The Poster Session of Semantic Web Applications and Perspectives (SWAP)*, 3rd Italian Semantic Web Workshop, PISA, Italy

Haarslev, V., Möller, R. (2003) Racer: An owl reasoning agent for the semantic web, In Proc. of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with 2003 IEEE/WIC International Conference on Web Intelligence, Vol. 13 (2003), pp. 91-95

J. Heflin, J. Hendler, S. Luke. Coping with Changing Ontologies in a Distributed Environment. In Proceedings of the Workshop on Ontology Management of the 16th National Conference on Artificial Intelligence (AAAI-99), WS-99-13, AAAI Press, pp. 74-79, 1999

Halpin, T.A. (2001) Information Modelling and Relational Databases: From Conceptual Analysis to Logical Design, ISBN-13: 978-1-55860-672-2, ISBN-10: 1-55860-672-6, San Francisco, California, Morgan Kaufman Publishers
Henricksen, K., Indulska, J., and Rakotonirainy, A. (2003) Generating Context Management Infrastructure from High-Level Context Models. In Industrial Track Proc. of the 4th Inter. Conference on Mobile Data Management (MDM'03), Melbourne, Jan, pp. 1–6

Jaro, M.A. (1989) Advances in Record-linkage Methodology as Applied to Matching the 1985, Census of Tampa, Florida. The Journal of the American Statistical Association, vol. 84, 414—420

Jaro, M.A. (1995) Probabilistic Linkage of Large Public Health Data Files (disc: P687-689). Statistics in Medicine, vol. 14, 491--498

Jarrar, M., Demey, J., Meersman, R.: *On reusing conceptual data modeling for ontology engineering*. Journal on Data Semantics 1(1), 185–207 (2003)

Jones, S.K. (1972) A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation, vol. 28(1), 11---21

Khattak, A.M., Latif, K., Lee, S. and Lee, Y.K. (2009): *Ontology Evolution: A Survey and Future Challenges*, Communications in Computer and Information Science, Volume 62, pp. 68-75, ISBN 978-3-642-10579-1

Klein, M. (2004): *Change Management for Distributed Ontologies*. PhD Thesis, Department of Computer Science, Vrije University, Amsterdam (2004)

Klein, M., Noy, N.F.: A component-based framework for ontology evolution. In: Proceedings of the (IJCAI 2003) Workshop on Ontologies and Distributed Systems, CEUR-WS, vol. 71 (2003)

Nodine, M.H. and Fowler, J. (2005): *On the Impact of Ontological Commitment*, Whitestein Series in Software Agent Technologies and Autonomic Computing, Book Ontologies for Agents: Theory and Experiences, Birkhäuser Basel,ISBN 978-3-7643-7237-8, pp.19-42

Nonaka, I., Takeuchi, H.: *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, Oxford (1995)

Noy, N. F., Klein, M. (2004): Ontology Evolution: Not the Same as Schema Evolution. Knowledge and Information Systems, 6(4):428-440, 2004

Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A Framework for Ontology Evolution in Collaborative Environments. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 544–558. Springer, Heidelberg (2006)

Ou, S., Georgalas, N., Azmoodeh, M., Yang, K. and Sun, X. (2006) A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development, in "Model Driven Architecture – Foundations and Applications", LNCS Vol. 4066/2006, 188-197, DOI: 10.1007/11787044_15

Schilit, B. N., Adams, N. L., and Want, R. (1994) Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, US

Spyns, P., Tang, Y. and Meersman, R., *An Ontology Engineering Methodology for DOGMA*, Journal of Applied Ontology, special issue on "Ontological Foundations for Conceptual Modeling", Giancarlo Guizzardi and Terry Halpin (eds.), Volume 3, Issue 1-2, p.13-39 (2008)

Stojanovic, L., Madche, A., Motik, B., Stojanovic, N. (2002): *User-driven ontology evolution management*. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 285–300. Springer, Heidelberg

Strang, T. (2003) Service Interoperability in Ubiquitous Computing Environments, PhD dissertation, Ludwig-Maximilians Univ. Munich

Strang, T., Linnhoff-Popien, C. (2004) A Context Modeling Survey. Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp’04, Nottingham, Sep.

Tang, Y. (2010) Towards Evaluating GRASIM for Ontology-based Data Matching, in proc. of the 9th international conference on ontologies, databases, and applications for semantics (ODBASE'2010), Springer Verlaag, LNCS 6427, p. 1009 ff, Hersonissou, Crete, Greece

Tang, Y., Debruyne, C. and Criel, J. (2010) Onto-DIY: A Flexible and Idea Inspiring Ontology-based Do-It-Yourself Architecture for Managing Data Semantics and Semantic Data, in proc. of the 9th international conference on ontologies, databases, and applications for semantics (ODBASE'2010), LNCS 6427, p. 1036 ff. Crete, Greece, Oct 26~28

Tang, Y., Meersman, R., Ciuciu, I.G., Leenarts, E. and Pudney, K. (2010) Towards Evaluating Ontology Based Data Matching Strategies, in proc. of fourth IEEE Research Challenges in Information Science RCIS'10, Peri Loucopoulos and Jean Louis Cavarero (eds.), pp: 137 - 146, ISBN: 978-1-4244-4839-5 Nice, France, May 19 - 21

Tang, Y. (2010) Semantic Decision Tables - A New, Promising and Practical Way of Organizing Your Business Semantics with Existing Decision Making Tools , ISBN 978-3-8383-3791-3, LAP LAMBERT Academic Publishing AG & Co. KG, Saarbrücken, Germany

Tang, Y., and Meersman, R. (2009) SDRule Markup Language: Towards Modeling and Interchanging Ontological Commitments for Semantic Decision Making, Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches, IGI Publishing, ISBN: 1-60566-402-2, USA

Tang, Y., De Baer, P., Zhao, G., and Meersman, R. (2009) On Constructing, Grouping and Using Topical Ontology for Semantic Matching, the 5th international IFIP workshop on Semantic Web and Web Semantics (SWWS’09), proc. Of On the Move to Meaningful Internet Systems: OTM 2009 Workshops, Springer, LNCS 5872, ISBN -978-3-642-05289-7, pp 816-825, Vilamoura, Portugal, Nov. 1 ~ Nov. 6

Trog, D., Tang, Y., and Meersman, R. (2007) Towards Ontological Commitments with O-RIDL Markup Language, Proc. of International RuleML Symposium on Rule Interchange and Applications (RuleML'07), in, Adrian Paschke and Yevgen Biletskiy (eds.), Springer Verlag, LNCS 4824

Verheijen, G., Van Bekkum, J.: *NIAM, an information analysis method*. In: Proc. of the IFIP TC-8 Conference on Comparative Review of Information SystemMethodologies (CRIS 1982). North-Holland, Amsterdam (1982)

Wang, X.H., Gu, T., Zhang, D. Q., and Pung, H.K. (2004) Ontology based Context Modelling and Reasoning Using OWL, Context Modelling and Reasoning Workshop at PerCom’ 04

Winkler, W.E. (1999) The State of Record Linkage and Current Research Problems, Statistics of Income Division, Internal Revenue Service Publication R99/04, Available from http://www.census.gov/srd/www/byname.html

[ii] Note that in the original document, Ontology transformation and ontology access are called data transformation and data access. In order to align our terminology, we use such naming.