Vrije Universiteit Brussel

Faculty of Science and Bio-Engineering Sciences
Department of Computer Science
Web & Information Systems Engineering Laboratory

Towards Collaborative Management of Semantic Business Rules: An Intelligent Environment Case

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Computer Science

Ioannis Arampatzis

Promotor: prof. dr. Olga De Troyer
Advisor: dr. Christophe Debruyne

2013-2014

Abstract

Nowadays we are facing an increasing challenge of smart environment automation, which is the use of information technology to control the available smart objects that comprise it. During the last decade several models of intelligent environments have been proposed. In order to operate in a dynamic environment, e.g. a smart home, devices must operate in a context-aware manner. In this thesis we adopt an ontology-oriented approach as a modeling technique for context-aware applications. We propose a method for the collaborative construction of Semantic Decision Tables (SDT) to manage semantic business rules over ontologies. More precisely, we adopt and extend the GOSPL tool for ontology engineering to facilitate the collaborative construction of SDTs. We furthermore implement an important component for this method, which is the mobile SDT editor. This component works as a rule configurator in an Android environment giving users the ability to create and use personalized semantics. The latter are used for the evolution of the collaborative construction of SDTs. To illustrate the proposed method we present a use case scenario for a smart home environment. Our method and tools facilitate on the one hand the collaborative constructions of rules inside GOSPL communities, and on the other hand, the extension of these rules by the SDT editor users for personalized semantics.

Acknowledgements

I would like to express my gratitude, first of all, towards my promoter prof. dr. Olga De Troyer for giving me the opportunity to work on my thesis under her supervision at the WISE Lab.

I would like furthermore to express my gratitude towards dr. Christophe Debruyne for being my advisor for this thesis, spending time in many conversations and providing answers to my numerous questions. His feedback was significant for this thesis to be completed.

I would like also to acknowledge the help and guidance of prof. dr. Robert Meersman and dr. Yan Tang Demey for accepting me at the STAR Lab environment, and for introducing me to the pertinent domain. They both gave me useful pointers and a starting point for my thesis.

Thank you.
Ioannis Arampatzis, Brussels, June 2014.

List of Figures

List of Definitions

Contents

Chapter 1

Introduction

1.1 Thesis Scope and Motivation

Information systems are developed to serve specific purposes. The Universe of Discourse (UoD) − specific to that purpose − is captured by a conceptual schema [HM08]. A conceptual schema of an information system is the result of the interactions between the business domain experts, the designers, and the users of that system. Thus, each conceptual schema is specific for every organization, regarding its application context. If two or more such information systems need to interoperate semantically, semantic technologies, and more specifically, ontologies are needed. An ontology is defined as *"an explicit specification of a conceptualization"* [Gru93b].

Conceptual schemas are used for the creation of a specific information system whereas ontologies are intended for reuse and interoperability between different information systems [SMJ02]. By using ontologies, *semantic interoperability* between different organizations' information systems is achieved.

Definition 1 (Semantic Interoperability [DLCM10]) *Semantic interoperability is defined as the ability of two or more autonomously developed and maintained information systems to communicate data and to interpret the information in the data that has been communicated in a meaningful manner, that is by means of an ontology that is shared by the involved information systems.*

Semantic interoperability is enabled by annotating the information systems of different organizations with ontologies. A *shared vocabulary*, expressed through the agreements that are made in the ontologies, between these organizations lets them exchange information. This is called as a *shared conceptualization* and is the result of the agreements between the stakeholders of the organizations with respect to their domain knowledge. The process of reaching the necessary agreements −

and all supporting activities, methods and tools involved − will be known as ontology engineering [GPFLC03].

The objective of this thesis is to investigate how we can easily manage business rules (especially decision rules) over ontologies. There exist several decision support technologies, such as decision trees, decision tables, Bayesian nets, balanced score cards, and decision models. A Semantic Decision Table (SDT) [TM07b] is a decision table properly annotated and formalized using ontologies. The implicit decision rules and meta-rules, such as the ones of describing dependencies among conditions, among actions, between conditions and actions, and across decision tables, are properly modeled as application-specific ontological models.

An SDT is business-oriented, meaning that it is suitable for being used by non-technical business people. SDTs and its supporting methods and tools have the possibility of formalizing business requirements into other formalisms, such as XML-based Semantic Decision Rule Language (SDRule-L) [TM09], or Description Logic as reported in [DT12] and even OWL as reported in [TM11].

SDTs furthermore provide means to capture decision makers' concepts, as well as their knowledge sharing in a collaborative environment [Tan10]. To this end, and as there was never introduced a tool for that, in this thesis we will investigate how we can use SDTs to collaboratively model business rules over ontologies. For this purpose, we have set up the following research questions:

- **Question 1** *How can we support the collaborative construction of SDTs?*
- **Question 2** *How can we extend SDT commitments to support user-defined rules (i.e. different layers of semantic rules)?*
- **Question 3** *How can the combination of both levels of SDTs evolve the SDT construction method?*

With respect to the research questions the following objectives are defined:

- **Objective 1** Provide a state-of-the-art review on similar projects on semantic decision making and the methods that will be used in our approach, to acquire the necessary knowledge in order to answer the aforementioned questions (Chapter 2).
- **Objective 2** Following the state-of-the-art review, investigate and propose a method for the collaborative construction of SDTs (Chapter 3). This objective will answer **Question 1**.
- **Objective 3** Display how the method - explained in Chapter 3 - is implemented (Chapter 4), and how this method can be extended, by means of tools, to support user-defined rules (Chapter 5). This objective will answer **Question 2**.
- **Objective 4** Demonstrate a generic method for the collaborative construction of SDT, in combination with the user-defined rules, and investigate how the latter can evolve the SDT con-

struction method (Chapter 5). This objective will answer **Question 3**.

1.2 Structural overview of the Thesis

The structure of this thesis is organized as follows:

Chapter 2 provides a state-of-the-art analysis on similar projects. Furthermore, a short description of the semantic web technologies is given, together with the engineering process on how to come to ontologies.

Chapter 3 presents the method that is adopted for the collaborative construction of the SDT. We show how we extend SDTs (Section 2.3.3) with glosses and social processes in order to achieve that.

Chapter 4 demonstrates the implementation of the adopted method, both for the collaborative construction and how end users commit to SDTs.

Chapter 5 demonstrates some generic examples of the creation of SDTs inside a community (collaborative construction of SDTs), and how reasoning works on local level (SDT Editor). To this end, the method on how the local level can evolve the collaborative construction of SDTs is provided.

Chapter 6 provides a summary of this thesis, as well as the limitations and the future work.

Chapter 2

Background and State-of-the-Art

2.1 Ontology and Ontology Languages

In this section we present an overview of what ontology is and how it can be used. In philosophy, ontology is a systematic account of existence [Gru93a]. The definition that is mostly cited in the computer science community is the one provided by Gruber [Gru93b], where ontology is defined as *"an explicit specification of a conceptualization."* Afterwards, the words *"formal"* and *"shared"* were added by the scientific communities as these aspects are necessary for ontologies to become usable in computer science [GG95]. Ontologies play a major role in the Semantic Web [BLHL01], and lead to the interoperation of information systems in a meaningful way. But interoperability is not the only benefit that we get from ontologies. Reusability of a shared understanding and a common vocabulary in a community are two of the most important benefits of ontologies. Figure 2.1 depicts the Semantic Web Stack[1], which shows the way to the Semantic Web. In this thesis the parts of the Semantic Web Stack that will be focused on are the following:

- **Resource Description Framework (RDF)** [LS99] model for data interchange.
- **RDF Schema (RDFS)** [BG04] for the development of taxonomies.
- **Web Ontology Language 2 (OWL 2)** [MPSP+09] for the creation of the ontologies.
- **Semantic Web Rule Language (SWRL)** [HPSB+04] for the description of business rules.
- **SPARQL Protocol and RDF Query Language** [PS08] which is a declarative query language for RDF.

[1] Available at `http://en.wikipedia.org/wiki/Semantic_Web_Stack`

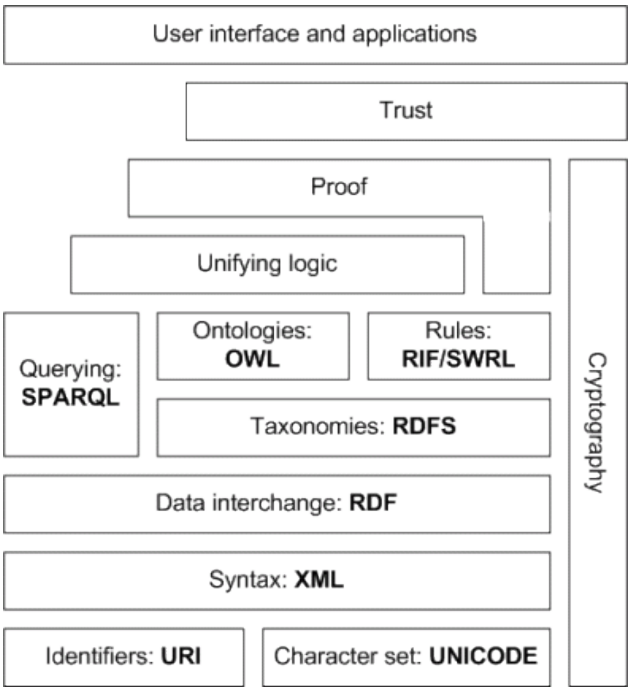

Figure 2.1: Semantic Web Stack [GVdMB08]

2.1.1 Resource Description Framework

The Resource Description Framework [LS99] was introduced in order to describe information and to define relationships between resources on the Web. Information is expressed in terms of *triples* $\langle subject, property, object \rangle$, where *property* stands for the relationship between the *subject* and the *object*. Furthermore, RDF helps agents to understand (albeit to a limited extent for software agents, as we will explain later on), share and exchange information. An issue regarding the nature of the Web is the need to avoid the name clashes that occur when referring to different Web sources. RDF uses namespaces, like in eXtensible Markup Language (XML), that are defined by referring to a Unified Resource Identifier (URI). In Figure 2.1 one can notice that URIs are the base of the Semantic Web Stack, which work as an identifier to the different sources of information that one may want to use. The limitation of RDF is that it is not very expressive (hierarchies and range properties cannot be expressed, for instance), thus it leads to reasoning limitations.

2.1.2 RDF Schema

RDF Schemas [BG04] (RDFS) are used to define the vocabulary and the relationships between concepts. In RDFS everything is expressed in terms of *classes*, *properties* and *values*. The difference with the XML Schema is that RDFS defines the vocabulary of the RDF and not the way it is structured.

In general, RDFS is too weak to describe the different resources in sufficient detail. The problem of RDFS is that even though it allows one to create classes and property hierarchies, and define the domains and ranges of properties, it does not support complex constructs useful for a wide range of applications such as cardinalities for properties, properties of properties (transitivity, symmetry, etc.), and the declaration of class disjointness, which bring us to the Web Ontology Language 2 (OWL 2).

2.1.3 Web Ontology Language 2

The Web Ontology Language 2 (OWL 2) [MPSP+09] is a language that defines the intended meaning of the modeling elements and the data values used in the different sources. It provides additional vocabulary along with formal semantics, and is used to formalize a conceptualization of a Universe of Discourse (UoD). OWL 2 supersedes the OWL standard of 2004 [AvH09], with which OWL 2 is compatible.

The second version of OWL, which became a W3C recommendation[2] in December 2012, is more efficient for reasoning tasks than the first version. It comes in three flavors: OWL 2 EL, OWL 2 QL and OWL 2 RL. The new functionality that was added in the second version of OWL with respect to the expressivity includes keys, property chains, richer datatypes, qualified cardinality restrictions, and enhanced annotation capabilities [GHM+08].

2.1.4 Semantic Web Rule Language

The Semantic Web Rule Language [HPSB+04] (SWRL) is based on a combination of OWL DL and Lite with the Rule Markup Language (RuleML). The rules have the form of a head (consequent) and a body (antecedent), and are expressed in terms of OWL concepts. As it stated in [HPSB+04], the intended meaning of a rule can be read as that whenever the conditions specified in the body are true, then the conditions specified in the head must also be true.

2.1.5 SPARQL Protocol and RDF Query Language

SPARQL Protocol and RDF Query Language [PS08] is an RDF query language, which means that it is able to manipulate data that are stored in RDF format. It is a W3C recommendation since 2008, and allows querying "triple" patterns (RDF). SPARQL allow users to write unambiguous queries, and is expected to make the Semantic Web machine-readable, which means that the (meta)data will be understood by computer systems (agents).

[2]OWL 2 W3C recommendation, http://www.w3.org/TR/owl-overview/

2.2 Related Work on Semantic Decision Making

In this section we describe the most relevant projects, approaches and services that have been developed.

During the last decade several projects have taken place regarding the modeling of intelligent environments. There are several topics that have been researched, but in order to operate in a dynamic environment, like a smart home, devices and agents must operate in a context-aware manner [SAW94]. When we use the word context we assume the definition from Dey et al. [DAS01], where *"context is any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects."* From the same paper we adopt the second category of context-aware functions, where a service is automatically executed. In that case, an application triggers an action on behalf of the user, according to the context changes.

To build context-aware applications one could say that there are two requirements. The first one is the context model and the second one is the middleware level support stated in Sugumaran and Gulla [SG11]. Regarding middleware we mean the data that we get from the physical layer (sensors) and the database. The most common middleware platform is OSGi[3]. Regarding the context model, there are three approaches to modeling techniques proposed in [GWPZ04]: the application-oriented, the model-oriented and the ontology-oriented. The application oriented and model-oriented approaches do not support knowledge sharing or context reasoning, as opposed to the latter approach. By the use of ontology languages such as RDF(S) and OWL the specification of concepts, the relationships and the reasoning over data is supported. Reasoning supports the creation and the management of specified rules by the users, in order to control their smart environment.

The most relevant project with our work is SESAME-S, described in Fensel et al. [FTK+13], which stands for SEmantic SmArt MEtering – Services for energy efficient Houses. One should note that when we refer to the SESAME-S project, we do not refer to the SESAME framework, described in Broekstra et al. [BKvH02], which queries and analyzes RDF data. In SESAME-S the authors adopt an ontology-based modeling approach and rules are translated into semantic policies. The authors propose an architecture (Figure 2.2) with a central Universal Control Box (UCB) that brings together an ontology and a rule infrastructure for context and rule-based reasoning, which is similar as the one in Ricquebourg et al. [RDM+07]. In the SESAME ontology, specified in OWL and N3 [TFP10], a hierarchy of concepts is provided.

The main concepts are the automation ontology, the meter data ontology, and the pricing ontology.

[3]OSGi Service Platform, `http://www.osgi.org`

The main purpose of the project is to reduce the energy consumption of the user. The end users are being offered a collection of pre-defined rules by *"power users"*, in order to avoid incorrect decisions. These pre-defined rules can be used by the end users in a drag-and-drop manner in the user interface that is provided to them. The reasoning engine that the authors used is Pellet [SPG+07] and is based on SWRL using the Jess Rule Engine [FH03]. The user interface is either a mobile or a web-based application. Regarding "power users", these are people who have experience with the models of the devices and the activities. The companies (vendors) that provide the sensors and the actuators could be included in that group.

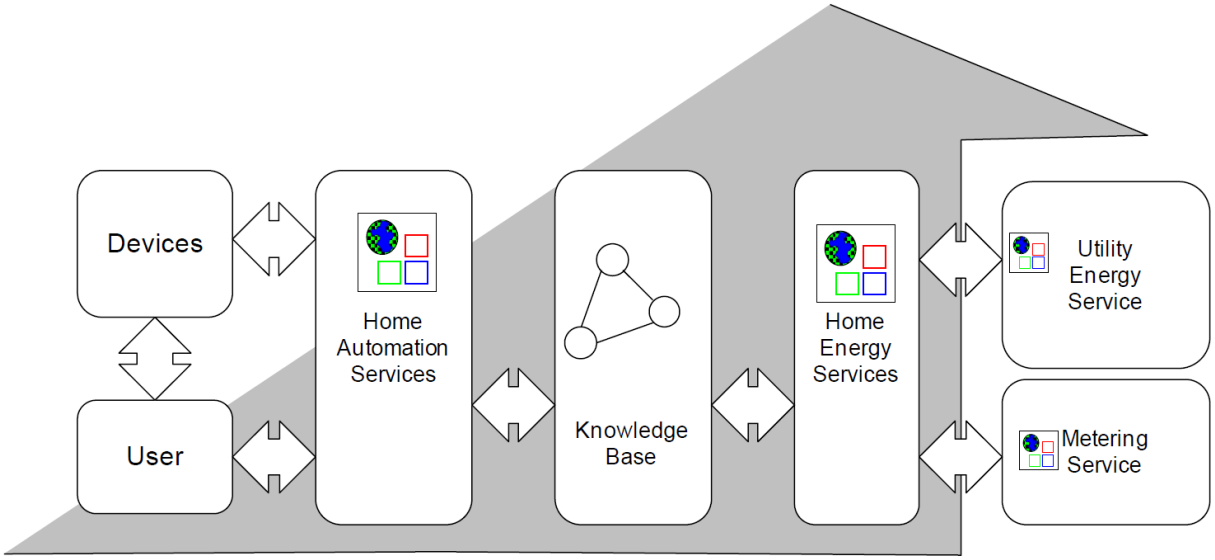

Figure 2.2: SESAME-S Service Architecture [TFP10]

Chen et al. [CFJ04] describe the Context Broker Architecture (CoBrA), which is an architecture for intelligent environments. In the center of this architecture (Figure 2.3), the broker agent manages and maintains the shared context model. The major components of the broker are the CoBra Ontology (COBRA-ONT), defined in OWL, which extends the SOUPA Ontology described in Chen et al. [CPFJ04], and the hybrid context reasoning engine, in which different engines are used for several types of reasoning. The EasyMeeting, proposed in Chen et al. [CP+04], is a prototype of a smart meeting room implemented on top of the CoBrA architecture, which uses the Jena[4] rule engine for ontology reasoning. The imported sensing information is translated into Jess[5] rules and the results from the reasoning procedure are stored back in the knowledge base in RDF format. The part that can be defined by the users is the privacy settings. The SOUPA policy ontology is used for that by the use of the Racer[6] reasoner.

[4]Apache Jena, `http://jena.apache.org`
[5]Jess rule engine, `http://herzberg.ca.sandia.gov`
[6]Racer reasoner, `http://www.franz.com/agraph/racer`

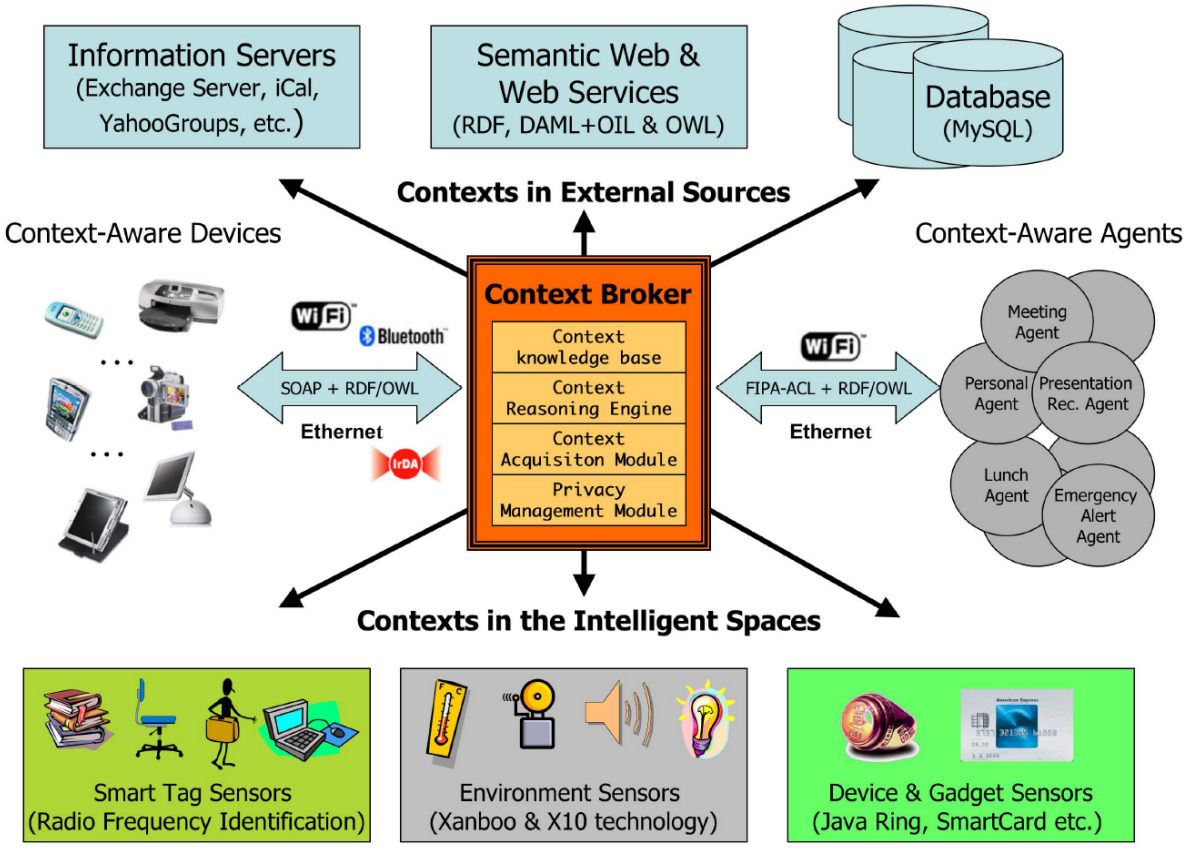

Figure 2.3: Context Broker Architecture [CFJ04]

Another ontology-based approach for context-aware applications is the Service-Oriented Context-Aware Middleware (SOCAM) described in Gu et al. [GWPZ04]. SOCAM's architecture (Figure 2.4) consists of context providers, either external or internal, the context interpreter, which consists of the context reasoning engines and the knowledge base, and the context aware services, which use the different levels of context. Furthermore in Wang et al. [WZGP04] a context ontology is proposed, named CONON, which consists of an upper and a specific (home domain) ontology. The reasoning is done by the Jena Framework. In the implemented prototype the authors state that there are two ways of reasoning, the ontology and the user-defined reasoning. However in their papers they do not state if there is an interface for the users to define their own rules. Last but not least, there are no references for community aspects, i.e. for the users to share their defined rules.

Similar to the ontology-based context-aware architectures, semantically enabled domotic systems have been developed, such as the Domotic OSGi Gateway (DOG) proposed in Bonino et al. [BCC08]. The word domotics comes from DOMus (latin for home) infOrmaTICS, and is another word for the digital home. The ontology that is proposed, called DogOnt, provides a conceptualization of

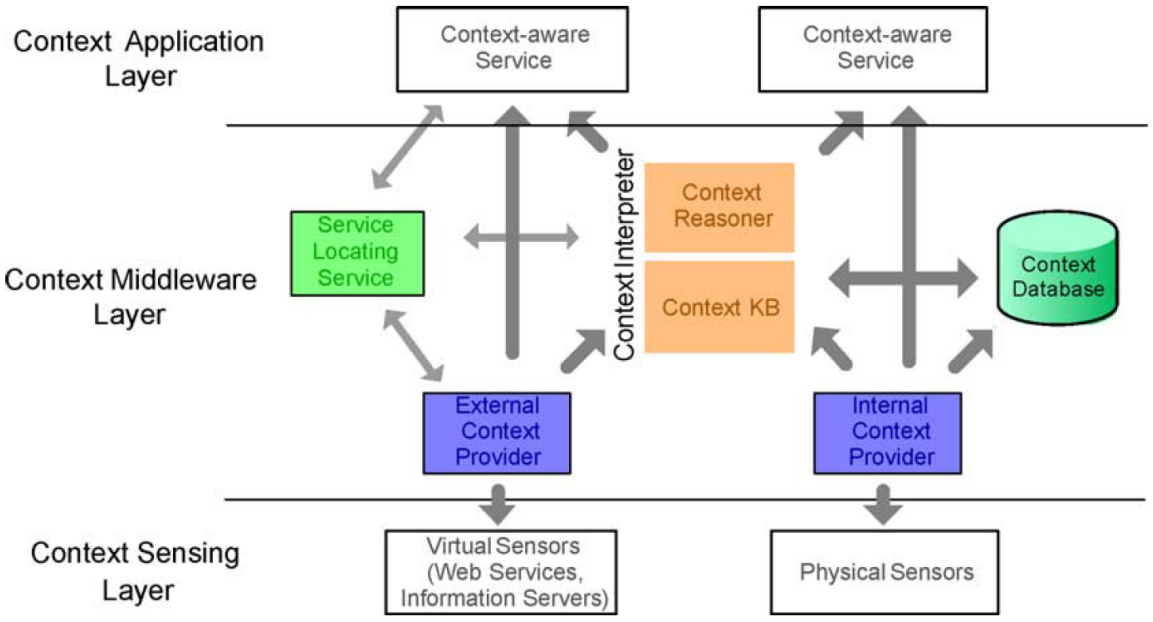

Figure 2.4: SOCAM Architecture [GPZ05]

devices inside a domotic environment. In DogOnt each device is represented as an object having a set of functionalities and states, which makes the procedure of adding new devices together with the states and functionality nontrivial. An API is built on top of the DOG architecture that is used to send commands to the devices, to search for the devices' states and to receive notifications when the states change. Last but not least, classification reasoning is used to automatically recognize device classes together with the functional descriptions as it is stated in Bonino and Corno [BC08].

Another related system of Ambient Intelligence (AmI) is intelliDomo, presented in Valiente-Rocha and Tello [VRT10]. It is based on ontologies for the control of domotic systems, such as DOG. In general, AmI refers to the (smart) environments that are responsive to the presence of people. The architecture that is proposed is shown in Figure 2.5. IntelliDomo is able to control the system automatically and in real-time. It is based on an ontology called OntoDomo. The user-defined rules can be modified via the DomoRules tool, which offers the user a simple SWRL rules creating application. This tool is developed in Java and supports the creation of rules through the SWRL-Factory API. That way the users will be able to produce their own rules and affect the system's behavior. Last but not least, the users are able to define some global situations, such as "cold" for temperature. These preferences are stored per profile in the ontology and cannot be shared between users.

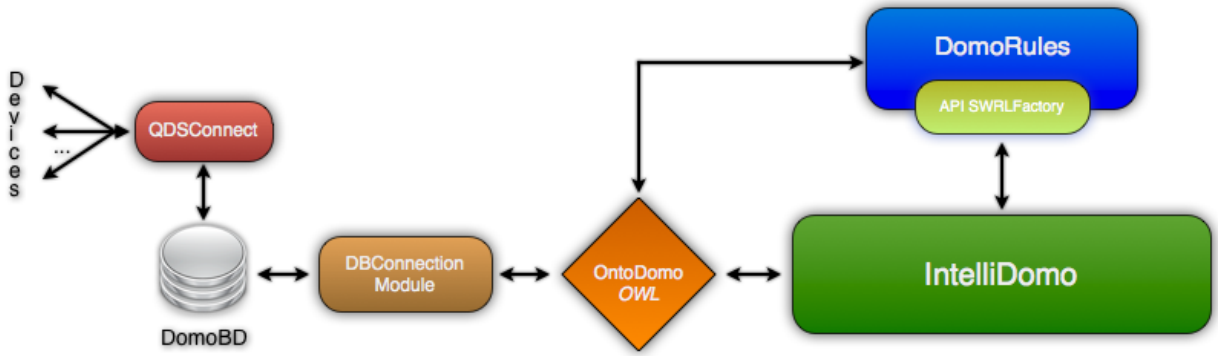

Figure 2.5: intelliDomo Architecture [VRT10]

2.2.1 Summary

This section includes a summary of the related work. Figure 2.1 displays a comparison table of all the projects that were analyzed in the related work section. An explanation of the left hand-side term is provided as follows:

- **User-defined rules** are the statements defined by the users to define or constraint the aspects of their smart environment, through a user interface. In all cases of the compared projects, one can note that SESAME-S and EasyMeeting are partially committed. That's because in SESAME-S, the rules are created by the so-called "power users", people who have experience with the models of the devices and the activities. In the EasyMeeting case, users can only define the privacy settings. For example, if they want to share their location or their status (e.g. being in a meeting). The only application that supports the user-defined rules is the intelliDomo application, where the users via an interface (DomoRules tool) can specify their rules, but still not in a user friendly manner.

- We use the term **"Extendable Rules"** when users can extend the existing rules by their own. This can only be done in the SESAME-S and in EasyMeeting projects, but still partially. For the former it is again up to the "power users" and for the latter just for the privacy settings. In all other cases the rules are not extendable.

- **Community aspect** is used in order to specify if the user-defined rules can be shared between communities of users. This happens only in the SESAME-S project where successful energy efficiency policies are shared. The concept of getting knowledge or any kind of input from the community does not exist.

- **Rule Language** is the language that is used to express rules as well as logic. To the known cases SWRL was used, which has the full power of OWL DL, but decidability cannot be

	SESAME-S	EasyMeeting (CoBrA)	CONON Prototype	DOG	intellidomo
User-defined Rules	Partially	Partially	No	No	Yes
Extendable Rules	Partially	Partially	No	No	No
Community Aspect	Yes	No	No	No	No
Rule Language	SWRL	Not known	Not known	SWRL	SWRL
Reasoning Engine	Pellet	Racer	Jena	DOG API	SWRLFactory API
Expressivity	SROIQ(D)	SRIQ(D-)	Not known	Not known	Not known
Natural Language	No	No	No	No	Partially
Ontology Name	SESAME	COBRA-Ont	CONON	DogOnt	OntoDomo
Aim	Context-Aware Services	Context-Aware Services	Context-Aware Services	Domotic Systems	Ambient Intelligence
Application	Energy Efficiency	Smart Meeting Room	Smart Home	Domotic Environment	Domotic Environment

Table 2.1: Related Work Summary

guaranteed.

- **Expressivity**[7] is the breadth of ideas that can be represented and communicated in that language.
- We use the term **"Natural Language"** when the application offers the ability to the users to use their own natural language to describe their own interpretations. The use of natural language exists in the intelliDomo projects and the interpretations, like "hot" or "cold" for temperature are pre-defined, and cannot be extended.
- **Ontology Name** is the name of the ontology that is used in the applications. **Aim** and **Application** are used to describe the category and the purpose of the projects.

From the related work analysis we concluded that the management of the semantic business rules is mainly maintained by the domain experts. End users are not furthermore given the ability to

[7]Expressive power, `http://en.wikipedia.org/wiki/Expressive_power`

extend the existing systems by means of their own defined rules.

2.3 Method Baseline

2.3.1 Ontology Engineering

In ontology engineering a group of stakeholders – which we also call a *community* – has a need to establish semantic interoperability and therefore aim to construct an ontology. The challenge of ontology engineering is to reach the necessary agreements to construct the ontology. A definition of ontology engineering that is adopted for this thesis is provided.

Definition 2 (Ontology Engineering [GPFLC03]) *Ontology engineering is the set of activities that concern the ontology development process, the ontology life cycle, and the methodologies, tools and languages for building ontologies.*

Since the requirements of a community evolve over time, so will the ontology. This makes the ontology-engineering process nontrivial. In the end, the agreed ontology will be used in the context of applications.

The ontology-engineering method that we adopt in this thesis is GOSPL, which stands for Grounding Ontologies with Social Processes and natural Language, proposed by Debruyne and Meersman [DM12]. GOSPL is based on the DOGMA ontology-engineering framework, which stands for Developing Ontology-Grounded Methods for Applications, proposed by Jarrar and Meersman [JM08].

2.3.2 Developing Ontology-Grounded Methods for Applications

In DOGMA, an ontology is decomposed into an ontology base and a commitment layer. This decomposition is called the principle of double articulation [SMJ02], as it is depicted in Figure 2.6.

The commitment layer is a set of application axiomatizations and mappings. The mapping from an application to the lexon base is called ontological commitment, which refers to a conceptualization. The axiomatization of an application consists of two parts. The first part is the set of lexons from the lexon base, and the second part is the set of rules that are used to constraint these lexons.

Definition 3 (Lexon [DM11]) *A lexon is described as a 5-tuple of the form* $\langle \gamma, Term_1, Role, InvRole, Term_2 \rangle$*, where:*

- $\gamma \in \Gamma$ *is the context identifier*
- $Term_1$ *and* $Term_2$ *are linguistic term-labels* $\in T$
- $Role$ *and* $InvRole$ *are pair role-labels, constituting a binary relation* R*, e.g.* `HasType/isTypeOf`

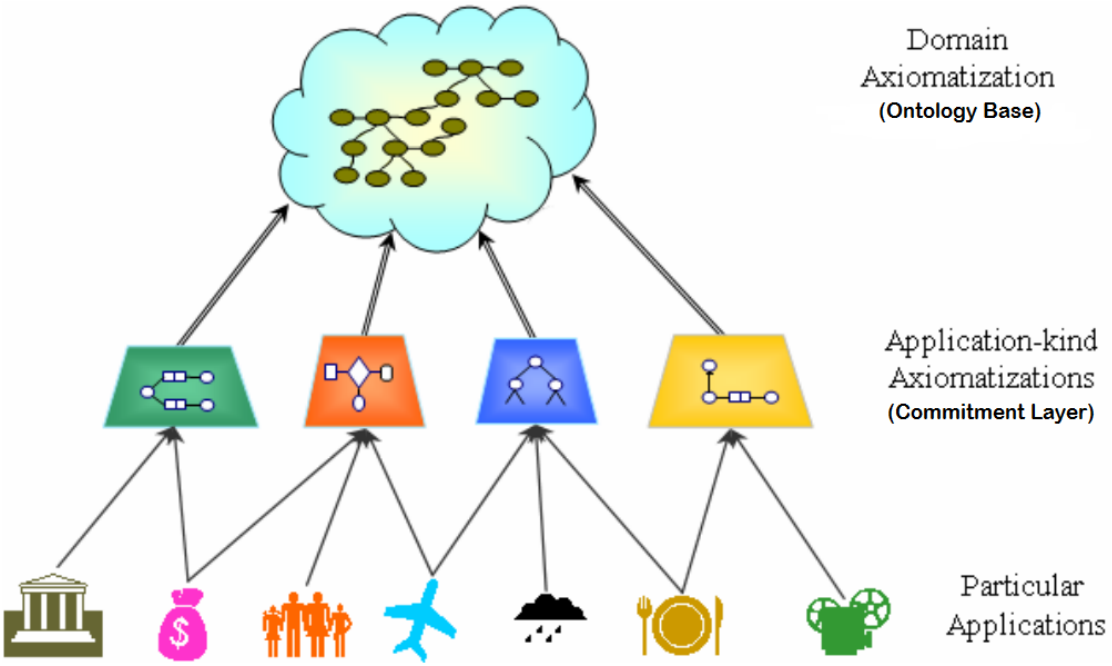

Figure 2.6: Ontology double articulation [JM08]

In the DOGMA approach it was shown that the use of fact-oriented techniques, such as Object-Role Modeling (ORM) described in Halpin and Morgan [HM08] is useful for the creation of ontologies [Spy05]. ORM is a method for modeling an information system at the conceptual level that can be easily understood by people. The information that is inserted is based on fact types described in natural language, where labels are used to describe certain concepts. These labels are described by the glosses in the DOGMA approach.

Definition 4 (Gloss [DM11]) *A gloss is a natural language description providing an "explanation" for a term or a lexon adequate within a given context.*

Since humans in the ontology-engineering process exchange informal definitions between themselves, i.e., glosses, in order to reach agreements and all agreements that lead to ontology evolution necessarily take place within that community, community should become an integral part of the ontology itself. All interactions leading to agreements are thus parameterized with the community. In order to promote communities to first class citizens, Debruyne and Meersman ([DM11], [DM12]) proposed to adopt and extend the DOGMA ontology-engineering framework.

By emphasizing the role of community and the exchange of glosses, the ontologies impart a hybrid aspect, which they call hybrid ontologies [MD10].

Definition 5 (DOGMA Ontology Description [DM11]) *A DOGMA Ontology Description Ω is an ordered triple $\langle \Lambda, ci, K \rangle$, where:*

- *Λ is a lexon base, i.e. a finite set of lexons*
- *$ci : \Gamma \times T \rightarrow C$ is a mapping function that maps a context identifier and a term uniquely to a concept $c \in C$, where C is a set of concepts*
- *K is a finite set of ontological commitments*

Definition 6 (Ontological Commitment [DM11]) *An ontological commitment is an ordered triple $\langle \sigma, \alpha, c \rangle$ where:*

- *$\sigma \subset \Lambda$ is a selection of lexons from the DOGMA ontology description*
- *$\alpha : \Sigma \rightarrow T$ is a mapping called an annotation from the set of application (information system, database) symbols to terms occurring in that selection*
- *c is a predicate over $T \cup R$ of that same selection expressed in a suitable first-order language*

2.3.3 Grounding Ontologies with Social Processes and Natural Language

Grounding Ontologies with Social Processes and Natural Language (GOSPL) [DM12] is a collaborative method for *hybrid* ontology engineering. Hybrid are the ontologies where:

- Communities are first-class citizens
- All agreements are parameterized with the community
- All interactions in a community are supported by a glossary

In GOSPL, an important instrument for reaching agreements within (a community of) stakeholders is glosses (Definition 4). Glosses help to remove ambiguities from the ontology and help to discover the relations between concepts. Ontologies are resources to be shared among both human stakeholders and the machines that need to use those to enable semantic interoperability. As the ontologies are the result of community agreements – supported by those glosses – ontologies impart a hybrid aspect. This lead to the development of so-called hybrid ontologies, which are ontologies in which all ontology-evolution operators are grounded with community agreements and supported by a set of glosses. The definition of the hybrid ontology description is as follows:

Definition 7 (Hybrid Ontology Description [DM11]) *A Hybrid Ontology Description is an ordered pair $H\Omega = \langle \Omega, G \rangle$, where:*

- Ω *is a DOGMA ontology description where the contexts in* Γ *are labelled communities*
- *G is a glossary. A glossary G is an ordered triple* $\Omega = \langle g_1, g_2, EQ_G \rangle$ *where:*
  - g_1 *is a function of the form* $g_1 : \Gamma \times T \rightarrow Gloss$*, the Term Glossary;*
  - g_2 *is a function of the form* $g_2 : \Lambda \rightarrow Gloss$*, the Lexon Glossary*
  - $Gloss$ *is a set of human-interpretable objects*
  - EQ_G *is a finite set of pairs of glosses considered to be describing the same concept*

The GOSPL method includes social processes between the stakeholders who are part of a community, in order to give the ability to the community to alter the hybrid ontology and to lead towards a closer approximation of its scope. In Figure 2.7 the different processes of the GOSPL method are summarized.

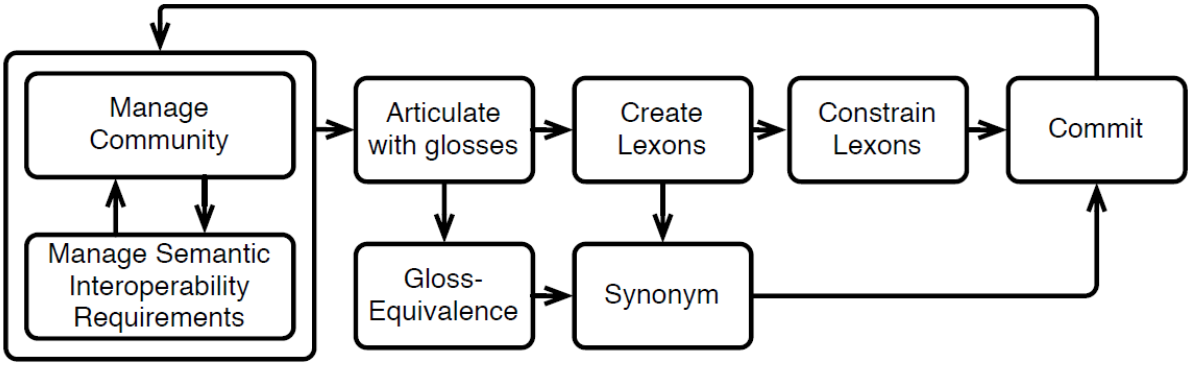

Figure 2.7: The GOSPL method [DM12]

The first action that needs to take place is the Semantic Interoperability Requirements (SIR). This is done by the users of the community who have a need to exchange information for a specific purpose. The definition of SIR is as follows:

Definition 8 (Semantic Interoperability Requirement [DM12]) *A semantic interoperability requirement for a community* $\gamma \in \Gamma$ *consists of an ordered tuple* $\langle KT, GO \rangle$*, where*

- $KT \subset \Gamma \times T$ *is a non-empty set of* key terms
- *GO is a non-empty set of goals for which descriptions of those key terms are needed*

We now list the existing processes. For their formalization, we refer to [Deb13]. We assume that the names of the processes are intuitive enough for the reader to understand.

- Request to add key to term

- Request to remove key term
- Request to add goal
- Request to remove goal

After this step, community users can start aligning the ideas of a community. In GOSPL terms are described informally before a formal description can be added.

Definition 9 (Articulation [DM11]) *Articulation means to describe a term referring to a concept with a natural language definition.*

Even though – at the start of a new community – no terms or lexons exist, users are already able to propose glosses to be used for certain terms. Communities preferably start with the key terms defined in their semantic interoperability requirements. When lexons do exist, the community is able to annotate both the terms in those lexons as the lexons themselves with a gloss. The social processes for managing glosses are:

- Request to add term-gloss
- Request to remove term-gloss
- Request to change term-gloss
- Request to add lexon-gloss
- Request to remove lexon-gloss
- Request to change lexon-gloss

Ideally, communities first agree on the informal definitions of concepts and, then, formalize the concepts using lexons and constraints. The next step is thus the creation of the lexons. From the definition of a lexon, users have to specify all five elements of the tuple $\langle community, Term1, Role, InvRole, Term2 \rangle$. The social processes in this phase are the following:

- Request to add lexon
- Request to remove lexon
- Change supertype of term

After this step, community users are able to constraint the lexons, like to create total and unique identifiers. The social processes in this phase are the following:

- Request to add constraint
- Request to remove constraint

Before we reach the commitment phase, there are two more steps that need to take place. During these steps the stakeholders try to find *gloss-equivalences* and *synonyms*. When two glosses from different communities describe the same concept one can state that these two glosses are *gloss-equivalent*. When two terms of different communities are the same one can state that these terms are *synonyms*. The two definitions are provided below.

Definition 10 (Gloss-equivalence [DM11]) *Given communities $\gamma_1, \gamma_2 \in \Gamma$ and terms $t_1, t_2 \in T$, the two glosses $g(\gamma_1, t_1)$, $g(\gamma_1, t_1)$ are said to be gloss-equivalent EQ_G if the two communities agree that the glosses used to describe the terms refer to the same concept (at gloss-level).*

Definition 11 (Synonym [DM11]) *Two community-term pairs $(\gamma_1, t_1), (\gamma_2, t_2) \in \Gamma \times T$ are said to be synonyms if terms t_1 and t_2, respectively in communities γ_1 and γ_2, refer to the same term (at term-level).*

One has to note the gloss-equivalences are on the glossary level, whereas synonyms are on the level of the formal description of concepts [DTM13]. When two glosses are the equivalent, it does not imply that the terms, described by these glosses, are synonyms. The gloss-equivalence concept can be used as a tool to help communities to find synonyms by double checking that the gloss-equivalence was not misleading, and that both terms refer to the same concept [DTM13].

The social processes in this phase are the following:

- Request to add gloss-equivalence
- Request to remove gloss-equivalence
- Request to add synonym
- Request to remove synonym

In the end, once (a part of) the hybrid ontology reaches a good approximation, the stakeholders can annotate the ontology to their information systems. On top of this method, a tool is designed that has the same name. This tool is presented in Debruyne and Meersman [DM12].

2.3.4 Semantic Decision Tables

There exist many decision support technologies such as decision tables, decision trees, Bayesian nets, balanced score cards, decision models and last but not least Semantic Decision Tables (SDT) [TM07b]. The main advantage of a decision table is that it is easy to be used and to be observed. This can be achieved due to the compact presentation of a decision table. Furthermore it is easy to learn how decision tables work and the process to construct them is trivial. A definition of a decision table is provided below.

Definition 12 (Decision Table [TM07a]) *A Decision Table is a triple $\langle C, A, R \rangle$, where:*

- *C is a set of conditions, A is a set of actions and R is a set of rules*
- *Each condition c_i, where $c_i \in C$, is defined as $\langle n_i, v_i \rangle$, where $n_i \in N$ is a condition stub label, and $v_i \in V$ is a condition entry value*
- *Each action a_i, where $a_i \in A$, is defined as $\langle m_i, u_i \rangle$, where $m_i \in M$ is a action stub label, and $u_i \in U$ is a action entry value*

- *Each rule r_j where $r_j \subset R$ is defined as a function $r_j : (setof)V^N \rightarrow A$*

Decision tables consist of three parts: conditions, actions and rules. Conditions comprise a condition stub and a condition entry. Similarly to conditions, actions comprise an action stub and an action entry. Rules are a combination of conditions and actions. The difference between decision tables and SDTs is that the latter are properly annotated with an ontology. This makes SDT to overcome the lack of semantic support that exists in decision tables.

Table 2.2: An example of a decision table

Condition	1	2	3	4
Temperature	>25	<=25	>25	<=25
Relative humidity	<40	<40	>=40	>=40
Action	1	2	3	4
Water the plant	*			

In Table 2.2 an example of a decision table is provided. This example is to decide on whether to water the plant or not. In this example two conditions take part. The first one has the condition stub label "Temperature" and the second one "Relative humidity." The condition entry values consist of the sets (>25, <=25) and (<40, >=40) respectively. The action stub label is the "Water the plant." The "*" next to the action stub label is the action entry value and defines whether an action is to be performed or not (when no action entry is provided). Rules comprise of the numbered table columns (*decision columns*).

In addition to DTs, SDTs contain three parts; a decision table, a set of lexons and a set of commitments.

Definition 13 (Semantic Decision Table [TM07a]) *A Semantic Decision Table is a quadruple $\langle \Gamma, C_l, A_l, R_l \rangle$, where:*

- *$\gamma \in \Gamma$ is a context identifier pointing to the original decision table and its settings*
- *C_l is the set of condition lexons and A_l is the set of action lexons*
- *Both C_l and A_l are defined inside the same context identifier γ*
- *R_l, which is generated through a formal ontological commitment, is a rule set that includes commitment axioms.*

An SDT is based on the DOGMA approach (Section2.6) and follows its layered structure: the lexon base layer, which contains the condition and the action lexons, and the commitment layer that defines the constraints through which applications may use these lexons. We continue with the definitions of the SDT lexon base and the SDT commitment base.

Definition 14 (SDT lexon base [TM07b]) *An SDT lexon base Σ is a finite set of lexons* $l = \langle \gamma, Term_1, Role, InvRole, Term_2 \rangle$

Definition 15 (SDT commitment [TM07b]) *An SDT commitment K is a tuple $\langle \zeta, \tau \rangle$ where:*

- $\zeta \in \Lambda \cup \{$*is a, part of, instance of*$\}$
- Λ *is a finite set of constraint types, subtypes and operators*
- *"is a", "part of" and "instance of" are role labels that can be interpreted respectively as taxonomical, meronymy and instance relationships*
- *given an SDT lexon base Σ and a set of term-instantiated lexons $inst(\Sigma)$, τ is a finite sequence of elements from $\Sigma \cup inst(\Sigma)$*

From the DT example illustrated in Table 2.2, if we would like to create its SDT, the lexons that should be extracted from the lexon base are the following:

- $\langle \gamma, Sensor, with, of, SensorType \rangle$
- $\langle \gamma, Sensor, with, of, ObservationValue \rangle$
- $\langle \gamma, Actuator, performs, perfomed\ by, Action \rangle$
- $\langle \gamma, Action, with, of, ActionName \rangle$

Following lexons, the commitments[8] that should be extracted from the commitment base are the following:

- $P1 = [Sensor, with, of, SensorType] : UNIQ(P1(with))$, which means that a $Sensor$ has at most one $SensorType$
- $P2 = [Sensor, with, of, ObservationValue] : MAND(P2(with))$, which means that a $Sensor$ has at least one $ObservationValue$

In this thesis, we assume that all lexons that exist in the lexon base exist also in the ontology. As a consequence all context identifiers, referred as γ in the extracted lexons of the aforementioned example, are communities. The reason for that is because these lexons are results of community agreements in GOSPL. As both SDTs and GOSPL are based on the DOGMA approach, a method for the collaborative construction of SDT by extending the GOSPL tool is proposed in Section 3.1.

[8] Commitments are explicated by the use of the notion of the semantic path, which helps us to express them in a language that can be interpreted

2.4 Conclusion

We provided some background information on the ontology-engineering process. Furthermore, from the state-of-the-art analysis on (context-aware) semantic decision making we concluded that the management of the rules is mainly maintained by the domain experts. End users are not given the ability to extend the existing systems by means of their own defined rules. In addition, there is no community aspect in the related projects. We argue that end users can be part of a community, and collaborate for the process of the semantic decision-making.

In the next Chapter, we aim to engage end users in this process by proposing a method for the collaborative construction of SDTs by extending GOSPL. The latter will be enriched with new social processes that will facilitate stakeholders to commit to SDTs. Furthermore, via an SDT editor, users will be able to define their own rules. These rules will support the evolution of the semantic decision making between communities of stakeholders.

Chapter 3

Collaborative Construction of Semantic Decision Tables

3.1 Introduction

In this chapter we propose a method for the collaborative construction of SDTs. It is stated in [Tan10] that SDTs provide means to capture decision makers' concepts, as well as their knowledge sharing in a collaborative environment, but there was never introduced a tool to support that. To this end, we describe how we adopt and extend the GOSPL method (Section 2.3.3) for ontology engineering, to support the collaborative construction of SDTs. GOSPL is a hybrid ontology-engineering method, with hybrid meaning that all concepts are always to be interpreted in a given context — namely the community in which the agreements took place —, and not only by their own formal structure [DM11], where a community agrees on those formal structures. These formal structures are the result of community agreements. A community consists of different stakeholders that share a common goal.

Both GOSPL and SDT are built on top of the DOGMA framework for ontology engineering. In DOGMA, the construction of ontologies is based on two layers: the lexon base – containing a vast set of plausible relations – and the commitment layer containing a selection of the strictly separate conceptualizations from any specific interpretations; a notion called *double articulation* (Section 2.3.2).

3.2 Extending SDT with glosses and social processes

Since the creation of SDTs is similar to the creation of ontologies, one can foresee how glosses and communities can play an important role in the creation of SDTs as well. We thus use glosses for the collaborative creation of SDTs. Before we continue, we define the collaborative (hybrid) SDTs as:

Definition 16 (Hybrid SDT) *A Hybrid SDT is a tuple $\langle SDT, H\Omega \rangle$, where:*

- *SDT is a semantic decision table*
- *$H\Omega$ is a hybrid ontology $\langle \Omega, EG \rangle$ with Ω a Dogma Ontology Description and EG an extended glossary*
- *EG is an ordered 6-tuple $E\Omega = \langle g_1, g_2, g_s, g_c, g_a, EQ_G \rangle$ where:*
  - *g_1 is a function of the form $g_1 : \Gamma \times T \rightarrow Gloss$, the Term Glossary*
  - *g_2 is a function of the form $g_2 : \Lambda \rightarrow Gloss$, the Lexon Glossary*
  - *g_s is a function of the form $g_s : S \rightarrow Gloss$, the SDT Glossary*
  - *g_c is a function of the form $g_c : C \rightarrow Gloss$, the Condition Glossary*
  - *g_a is a function of the form $g_a : A \rightarrow Gloss$, the Action Glossary*
  - *$Gloss$ is a set of human-interpretable objects*
  - *EQ_G is a finite set of pairs of glosses considered to be describing the same concept*

3.2.1 A Method for constructing hybrid Semantic Decision Tables

GOSPL prescribed several social processes for the creation of, for instance, lexons, glosses and constraints. For hybrid SDTs, we see the management of conditions (stub and entry) and actions (stub and entry) as a set of social processes of a community. At least one condition and one action are required in order to construct a hybrid STD, which is furthermore a separate social process inside a community. To this end, we extend GOSPL with the following social processes to support the creation of hybrid SDTs:

Conditions Management

- Request to add/remove/change gloss of condition stub
- Request to add/remove annotation from condition stub
- Request to add/remove condition entry

Actions Management

- Request to add/remove/change gloss of action stub
- Request to add/remove annotation from action stub

SDT Management

- Request to add/remove/change gloss of SDT
- Request to add/remove condition from SDT
- Request to add/remove action from SDT
- Request to add/remove decision rules from SDT

One can notice that there are three categories for the creation of a hybrid SDT. The first one is for the management of the conditions, the second for the management of the actions, and the last one for the management of the hybrid SDTs. All three categories start with the "request to add gloss." Glosses are responsible for describing the conditions, actions, and SDTs by their (implicit) meaning. Stakeholders' opinions or statements are part of the discussion that leads to an agreement regarding the gloss. To this end, an example of the use of glosses is provided.

If we recall the example in Table 2.2, imagine that we want to create an SDT to water our plant when it is hot, and the humidity is high. For that purpose we have to define an SDT with the name *"water the plant."* As we already have in mind when we want to water our plant, we define the SDT gloss as *"the purpose of this SDT is to water the plant when it is hot and the relative humidity is low."* Now stakeholders of the community know that we want to create an SDT that will water the plant when it is hot and humid. If there are not any objections and the community agrees on the SDT gloss, the conditions and the actions could be defined.

The community is now responsible to agree on what *"hot"* means, which leads to the connection of *"hot"* with the temperature. The condition stub label will thus be *"temperature is hot"*, with the gloss *"temperature is hot means being at a temperature that is higher than normal or desirable."* One objection here from a stakeholder could be that it is hot not only *"..being at a temperature.."* but also when *"..exhibiting a temperature.."*. This leads the community to agree on the gloss *"temperature is hot means being at or exhibiting a temperature that is higher than normal or desirable."* Now that the community has agreed on the meaning of *"hot"*, the condition entry value is responsible for defining the *"normal or desirable temperature."* For the purpose of this example let us assume that it is a temperature of 20 Celsius Degrees. Similarly, for the second condition regarding humidity, the condition stub label can be defined as *"low relative humidity"* with the gloss *"relative humidity, expressed as a percentage, measures the current absolute humidity relative to the maximum for that temperature. When the relative humidity drops below a percentage, the air feels dry to skin."* Again, the community has to agree on that percentage. From the aforementioned example, one can notice the advantages of using glosses, which work as a driver for the agreements in the community.

Glosses help not only for defining the stub labels and the entry values of the conditions and the actions, but also for the annotations with the lexons from the ontology, as described in the example of Table 3.1.

We see the method for the collaborative construction of SDTs as a sequence of social interactions, where two phases take place. The first one is the creation of the conditions and the actions, and the second one is the creation of the hybrid SDT by the use of the agreed conditions and actions of the first phase, as it is depicted in Figure 3.1. During the first phase, the condition and action management can happen at the same. In the end, the agreed conditions and actions will be used accordingly – depending on the purpose of the SDT – during the second phase, the SDT management, where the first step of the hybrid SDT construction is to add the agreed conditions and actions from the previous phase.

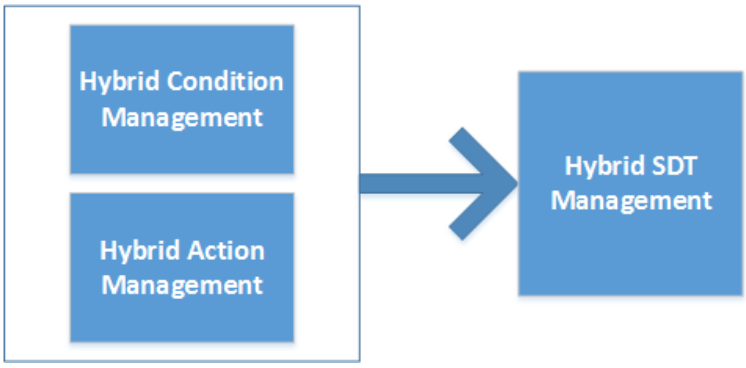

Figure 3.1: Method for the hybrid SDT construction (abstract)

In the following sections the processes of this method will be prescribed. But before we continue an example of how we use the proposed method to construct a hybrid SDT, by means of conditions and actions, is provided. The SDT that is used as an example is illustrated in Table 3.1.

Table 3.1: Hire Professional Driver - SDT

Condition	1	2
Driver with professional experience	Yes	No
Action	1	2
Hire	*	

In order to create the condition for the pertinent SDT, some social processes need to take place. The first one is the creation of the condition stub label. To this end, a gloss should be articulated to the pertinent condition stub label. In this case, the gloss is defined as "A Person who owns a driving license and has professional driving experience." Once the stakeholders of the community, and in particular of the *Company Community*, agree to that gloss, the process to add the annotations take

place. From the aforementioned gloss, stakeholders are able to track the lexons that are needed for the annotation of this condition stub label. Namely, these would be the following:

- $\langle Company\ Community, Person, owns, owned\ by, License\ Type \rangle$
- $\langle Company\ Community, License\ Type, is\ of, has, Drivers\ License \rangle$
- $\langle Company\ Community, Person, with, of, Driving\ Experience \rangle$
- $\langle Company\ Community, Driving\ Experience, is\ a, is, Professional\ Experience \rangle$

The procedure to annotate the condition and action stub labels is non-trivial. In the aforementioned example we see how glosses can help stakeholders to annotate the stub labels with the pertinent lexons. Once the annotation process is complete (after the agreements inside the community), the process to add the condition entry values to the condition should take place. Regarding the action, the same procedure applies, with the difference that no action entry values can be added, as this is part of the decision rule creation process, described in Section 3.2.4. The complete conditions and actions are now ready to be added to an SDT. In order to create an SDT, a gloss should again be articulated to the SDT. In this case, the gloss for the SDT would be "SDT to hire a driver with professional experience."

3.2.2 Hybrid Conditions

We see the creation of a condition in two steps. Firstly, the creation of a condition stub and secondly the creation of a condition entry. To start with, to create a condition stub a gloss should be articulated to that condition stub. Articulation is the process of describing with a natural language description (Definition 9). In the example of Table 3.1 we showed how to use glosses for the creation of a condition. The reason for using glosses is twofold. With the use of glosses the stakeholders of a community can reach agreements, as natural language descriptions are interpretable by humans, thus they can discuss and reason about the concepts that are introduced. We argue that condition and action stub labels are not enough to describe them. By using the aforementioned example with the "Driver with professional experience" gloss, one would mean a "Person who has experience in sport racing", which is a totally different concept from the one that was introduced in the example. This is one of the main reasons why glosses are of major importance. Secondly, we see that the use of glosses can help the evolution of hybrid SDTs between different communities, as it is, for example, described in GOSPL with the *gloss-equivalences* and *synonyms* (Definitions 10 & 11). This part is beyond the scope of this thesis, but could be considered as future work.

Once the condition stubs are articulated with glosses, the annotations with the lexons of the community take place. This is the part where the stubs are mapped to a (set of) lexon(s) that exist in the community's lexon base. The target is to build the connection between the condition stubs and

the concepts in the ontology.

The last part of this process is the creation of the condition entry, which is a (range) value that is assigned to a condition stub. By the time the stakeholders of a community agree on a value for the condition entry, a condition is considered complete. The procedure of the creation of a condition is depicted in Figure 3.2

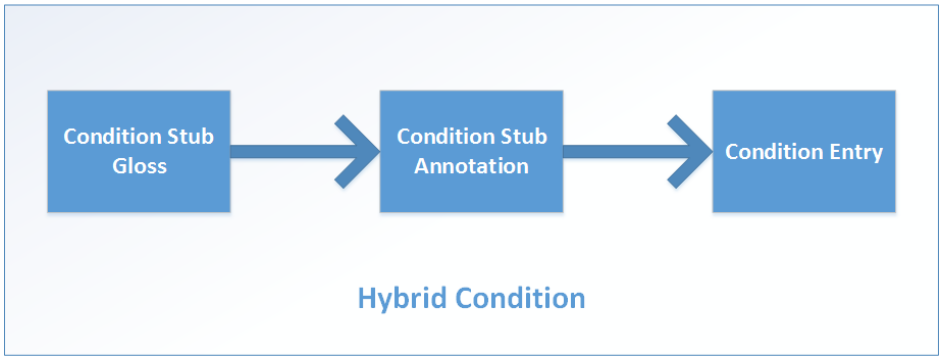

Figure 3.2: Method for the hybrid condition construction

3.2.3 Hybrid Actions

Actions are the same as conditions. The main difference here is that even if actions do have an entry, the instantiation is not part of this process. Action entries will be used in the decision rules process (Section 3.2.4). Same with the conditions, actions are articulated with glosses. Once this part is complete, the annotations take place, as it is displayed in Figure 3.3.

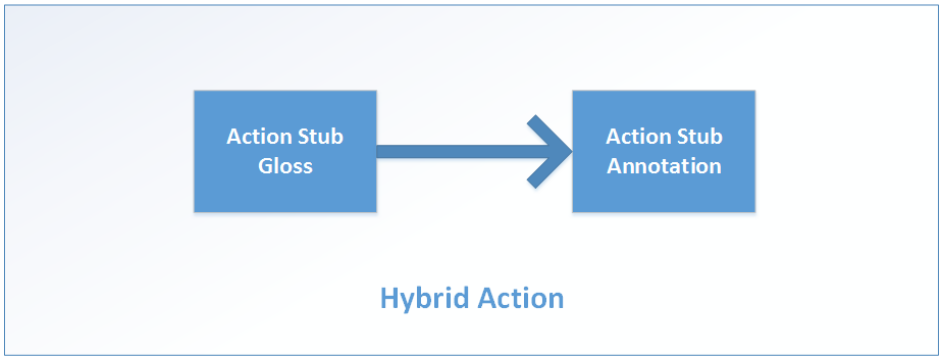

Figure 3.3: Method for the hybrid action construction

3.2.4 Hybrid SDT

The last part of this method is of course the creation of the SDT. This process begins with the articulation of the SDT with a gloss. The stakeholders of the community afterwards can add the conditions and the actions to the SDT. In this step one should note that the gloss of the SDT helps the stakeholders to interact and agree on the desired conditions and actions. If we recall the example of Table 3.1, the gloss of the SDT is "an SDT to hire a driver with professional experience." One could clearly notice that the gloss of this SDT defines the conditions and the actions that need to be included to the SDT, which in our example is the "Driver with professional experience" for the conditions, and "Hire" for the action. Furthermore, the SDT gloss is the part that allows the stakeholders of different communities to adopt an SDT across different communities.

Regarding the conditions and the actions, the stakeholders can choose from the condition and the action base. The selected conditions and actions are mapped to the SDT. The last part that is left is the creation of the decision rules. A decision table is created and the only part that is missing is the action entries. By the time the stakeholders add the action entries to the table, the decision rules are complete and the method for the creation of a hybrid SDT is over. The procedure is depicted in Figure 3.4.

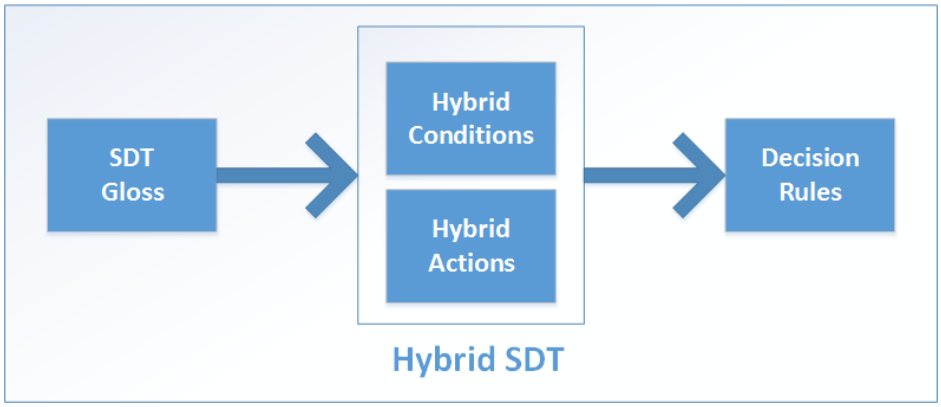

Figure 3.4: Method for the hybrid SDT construction

3.3 Conclusion

In this chapter the social processes that are needed to create a hybrid SDT are prescribed. The proposed method, together with the sequence of the social processes that the community stakeholders need to follow is prescribed. During these social processes, glosses play a major role. We see the hybrid SDTs as an extension of a community commitment. This commitment includes a selection of a (set of) lexons, the constraints that apply to these lexons, and the rules that are

the outcome of the agreed SDTs. An example of how to create a hybrid SDT is prescribed in the following chapter (Section 4.1.2).

Chapter 4

Implementation

4.1 The collaborative SDT engineering platform

The collaborative SDT engineering platform is built as an extension of GOSPL (Section 2.3.3). GOSPL - and therefore also the prototype of this thesis -is developed as a J2EE project, running in a JBoss Application Server[1]. Regarding the client, it is developed by a combination of JavaServer Pages[2] and JQuery. In this chapter the SDT commitment and glosses together with the social processes for the SDT creation will be prescribed.

4.1.1 The SDT commitment and glossary

Once a stakeholder of a community is logged in, the main menu inside the community is displayed. The second tab is called "glossary." This tab lists the glosses of a community, which are used to articulate the terms and the lexons that are part of the community commitment. As it is displayed in Figure 4.1, this tab is extended in order to display not only the aforementioned glosses, but also the glosses for the conditions, the actions and the SDTs. That way the stakeholders can have a general view regarding all the glosses that are part of the community commitment.

The third tab is called "SDT." In this tab an overview of all the SDTs together with the conditions and the actions is displayed. There are three main parts. The first part is the table with all the SDT commitments of a community. This table, provided in Figure 4.2, displays the name of the SDT with the conditions and the actions that take part in it.

[1] Available at `http://www.jboss.org`

[2] Available at `http://www.oracle.com/technetwork/java/javaee/jsp/index.html`

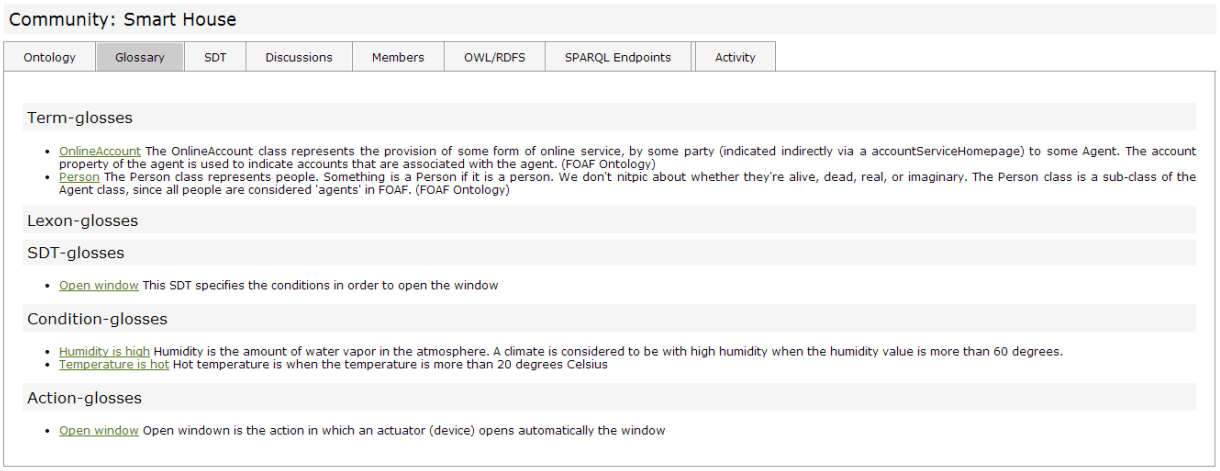

Figure 4.1: Extended glossary of the Smart House community

Figure 4.2: SDTs overview table inside the SDT tab

Secondly there are two more tables, one of the conditions and one of the actions. In those two tables the (condition and action) stubs are displayed, as it is depicted in the following screenshot (Figure 4.3 and Figure 4.4 respectively). In these tables more information about the conditions and the actions is displayed, like the annotations and the condition entries for the conditions.

From the "SDT" tab, the ability to access the XML file of the SDT commitment is given (Figure 4.5). Apart from the display of the XML file, the link on which the XML file can be accessed is available.

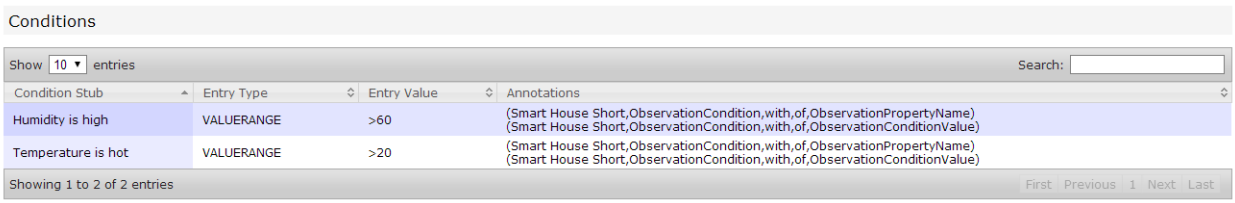

Figure 4.3: SDT conditions overview table inside the SDT tab

Figure 4.4: SDT actions overview table inside the SDT tab

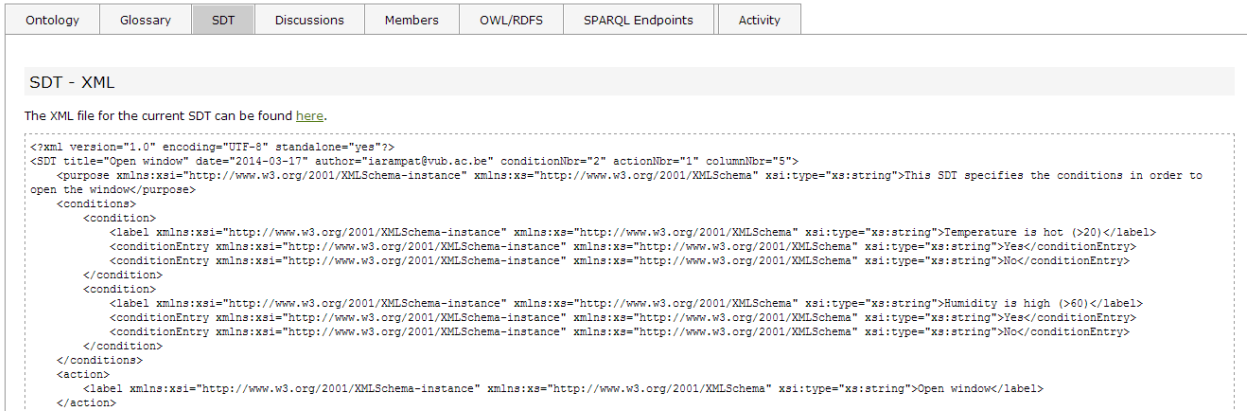

Figure 4.5: XML implementation of the hybrid SDT

4.1.2 Social Processes for hybrid SDT in GOSPL

Because GOSPL is discussion-oriented, the evolution of the ontology is dependent on the agreements between the stakeholders. The same applies to the SDTs. The method on how to create SDTs is analyzed in Section 5.1. Several social processes take place in this method. To start with, the conditions need to be added to the condition base. To this end, stakeholders need to agree on the gloss for the condition stub. If we recall the example of Table 3.1 with the condition stub label "Driver with professional experience", this is the social process in which stakeholders interact and decide on the gloss of the condition. Such an interaction is illustrated in Figure 4.6. In general, the procedure that has to be followed is firstly to add the glosses, continuing with the annotations and in the end adding the condition entries.

All these social processes can be accessed via the "discussion" tab, of the main menu. The processes for the conditions are displayed in Figure 4.7. The two other social processes that are also of great importance in this part are the "request to add annotation to condition stub" and the "request to add condition entry." Regarding the first one, a stakeholder should choose a condition stub together with a lexon from the lexon base. Stakeholders are not limited to only one lexon per condition stub, and this procedure can take place multiple times. Thus, a condition stub can have multiple annotated lexons. A screenshot of this social process is displayed in Figure 4.8.

Figure 4.6: Social processes for the condition gloss

The social process "request to add condition entry" includes also the selection of the condition stub in the first place, and then requests from the stakeholders to select the entry type. This is a selection between a value, a value range and a Boolean value. A screenshot of the "request to add condition entry" is provided in Figure 4.9.

In parallel with the conditions, stakeholders can also add the actions for the SDTs. The procedure is the same. The social processes regarding the actions are displayed in figure 4.10.

The last part before the creation of the SDTs is the agreement of the decision rules. A screenshot with the social processes regarding the completion of an SDT is provided in Figure 4.11. Stakeholders have first to agree on the conditions and the actions that will take part in an SDT, and then add them. The last step is to "request to add decision rules." Again, the SDT is selected, and then the SDT is automatically generated. For this part the SDT API[3] is used. A screenshot of the latter social process is displayed in Figure 4.12.

By the time the decision rules are agreed, the generated XML file can be found under the "SDT" tab by selecting the desired SDT.

[3] Available at `http://sourceforge.net/projects/sdtapi/`

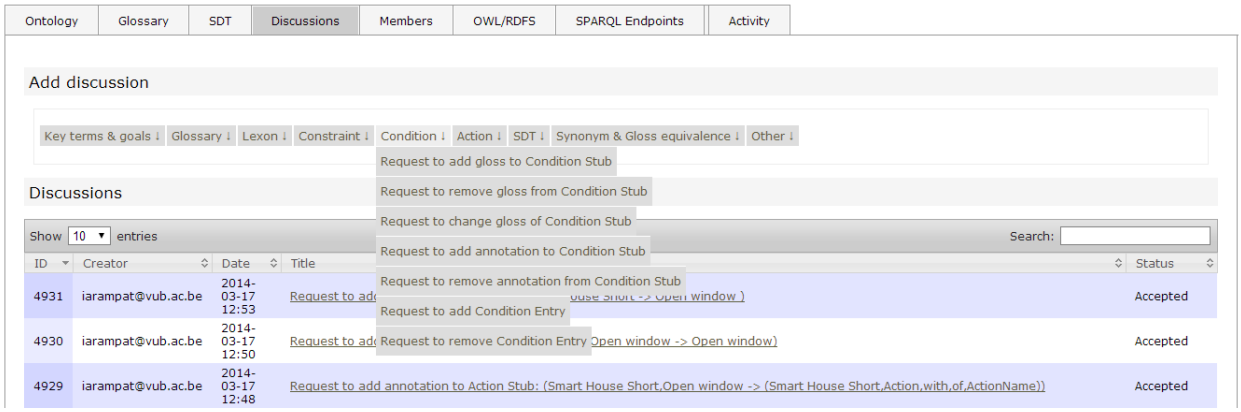

Figure 4.7: Social processes for the hybrid conditions

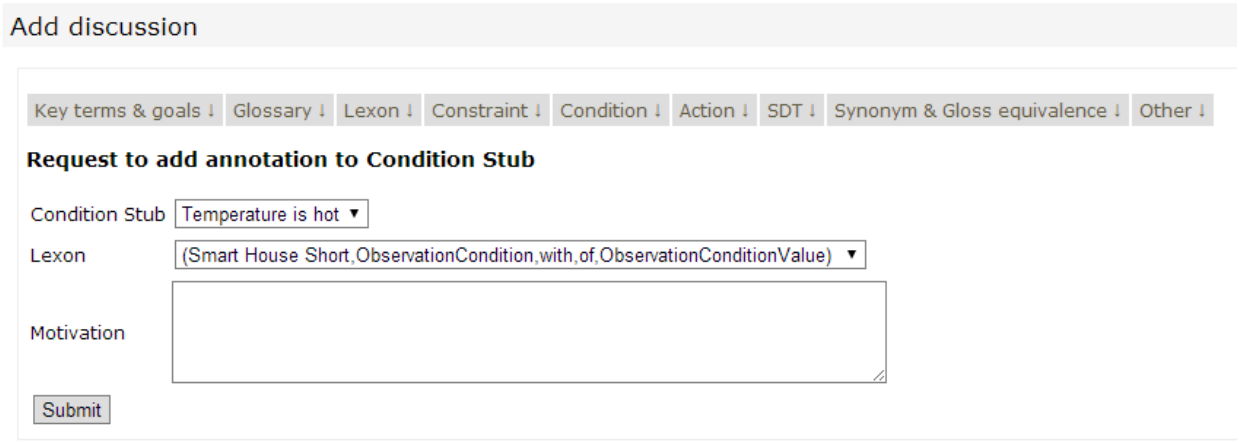

Figure 4.8: Social process - Request to add annotation to condition stub

4.2 SDT Editor

In this section we will prescribe the SDT editor, which is a component for the management of decision rules from its users. It is an Android application which gives the ability to edit the SDTs that are created in GOSPL communities. Firstly, an introduction to the Android OS and development is given.

Android is an Operating System (OS) not only for mobile phones and tablet devices, but also for desktops. The first release of the system was in 2008. It was developed by Google in the union of the Open handset alliance, i.e. a consortium of firms with the purpose of developing open standards for mobile devices. Android was then built as the first open source platform for mobile devices. Since Android is an open source project, developers have access to all the platform source code. They are free to change part of the base code without the need for a license agree-

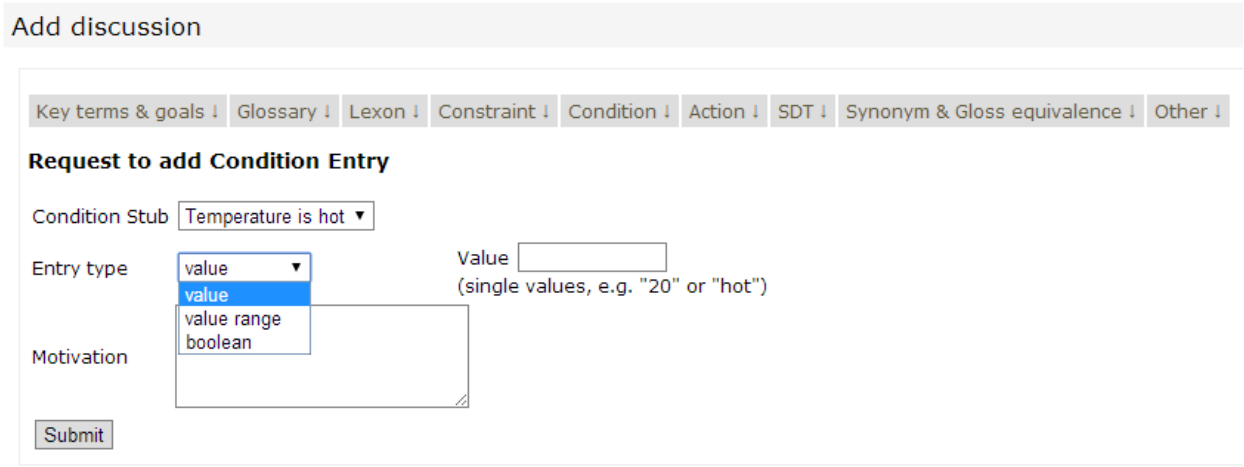

Figure 4.9: Social process - Request to add condition entry

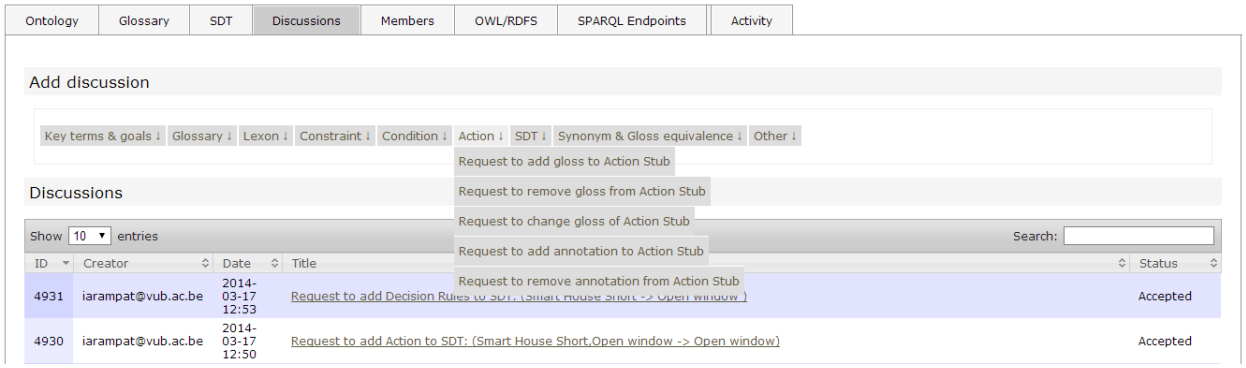

Figure 4.10: Social processes for the hybrid actions

ment. Furthermore, the Software Development Kit (SDK) offers developers the necessary tools to start building the Android applications of their preferences. Android's programming language is Java. However, Android does not use the Java Virtual Machine to run the generated code. In this case a different Virtual Machine is used, namely the "Dalvik Virtual Machine." Eclipse is the official Integrated Development Environment (IDE) that supports android applications' development. Yet an additional plugin - Android Development Tools (ADT) - has to be installed in order to extend the IDE functionalities. This plugin allows managing android projects and also includes the Android Virtual Device Manager that permits to create different device emulators for the different available versions of the API. The ADT extends the existing capabilities of Eclipse to let one quickly set up Android projects, create the User Interface Design for his applications, add packages and libraries based on the Android Framework API.

Our Android application, the SDT editor, consists of four activities. One can consider activities as subsections of the main application, given for specific purposes. These four activities are the

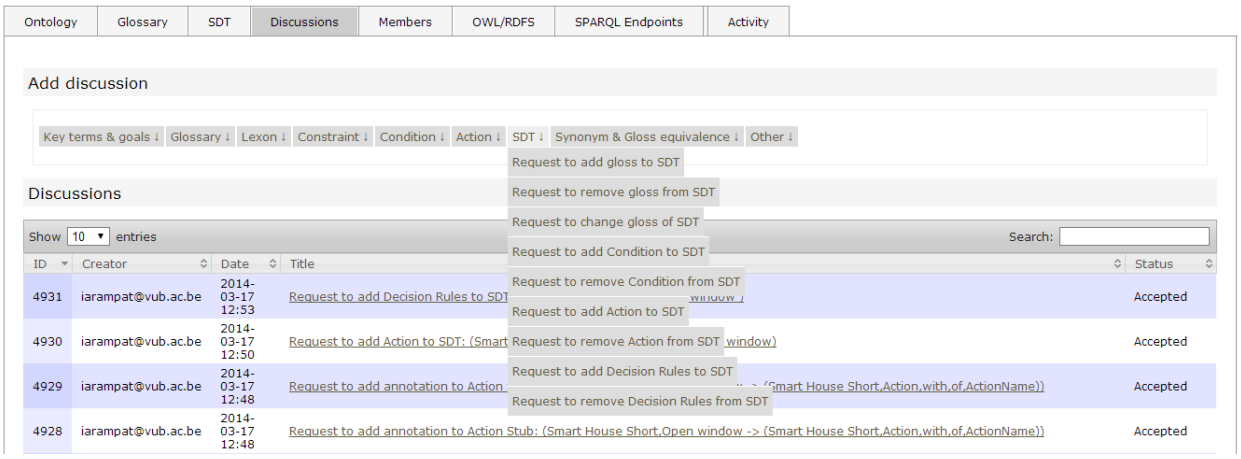

Figure 4.11: Social processes for the hybrid SDT

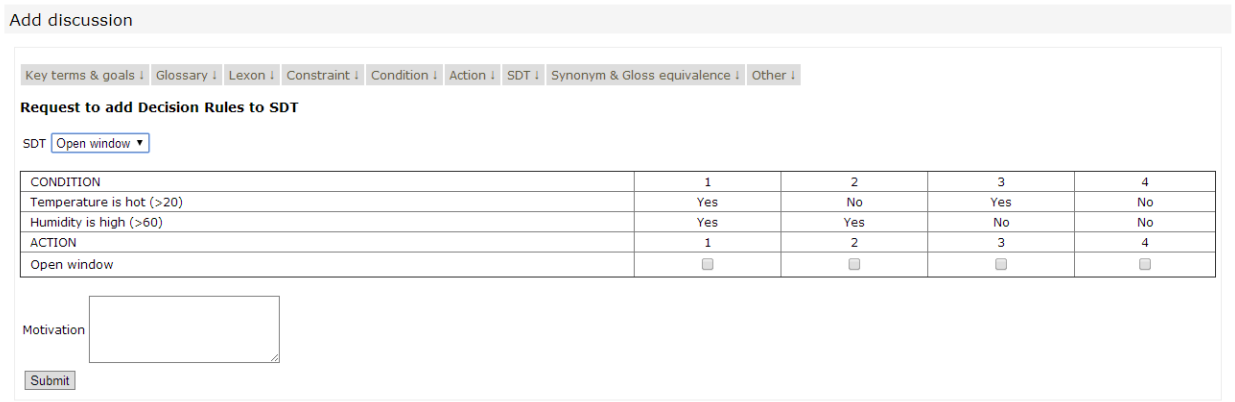

Figure 4.12: Social process - Request to add decision rules to SDT

following, as it is depicted in Figure 5.10:

- General Management
- Edit Decision Rules
- Edit Conditions
- Share SDT

The first activity, the **General Management**, is the activity responsible for the management of the SDT files. Users are able to insert the endpoint that points to the desired community and download the SDTs that are agreed in that community. These SDTs are the *SDT commitments* of a community. Furthermore, in this activity, users are able to view the SDTs and to delete them, if their needs are not met from these SDTs.

The second activity, the **Edit Decision Rules**, is the activity where users are able to edit the de-

cision rules of the SDT commitments. Firstly, a list with all the SDTs that are downloaded is displayed. Again, users are able to view the SDTs and then to edit the decision rules according to their needs. When they tap the button ”edit”, a table is displayed with the pre-defined decision rules selected. With the use of checkboxes, users are able to change these pre-defined values, and to save them into the same SDT file, which is stored locally. Furthermore, the user and the modified date are captured.

The third activity, the **Edit Conditions**, is responsible for the edit of the condition stubs and entries. It applies the same as with the *Edit Decision Rules* activity, regarding the list and the view of the SDTs. This time the button ”edit” redirects the users to a new list with the conditions that are expressed inside the selected SDT. Then, users are able to edit the desired condition just by taping the button ”edit” next to the condition. There, they can change the name of the condition stub, giving their own interpretation to the condition, and to change the value of the condition entry. One should note that the annotations of the condition remain the same, and users are not able to change them. A message of confirmation is displayed in order to reduce the possibility of a mistake from the side of the user. After the confirmation of the user the file is saved and stored locally.

The last activity, the **Share SDT**, is the activity where users are able to share their edited SDTs inside the community. For the purpose of this task, an SDT repository is used. The benefit from this architecture is twofold. This SDT repository can operate as an input to the GOSPL community, and as a way for users to share their edited decision rules and conditions. We argue that this is an exchange of knowledge, and can help the evolution of the SDT commitments. In GOSPL, a table with all the edited SDTs is created, where the stakeholders of a community can check and act accordingly (e.g. create a new social process with a request to edit decision rules, or request to edit condition stubs and entries).

4.3 SDT Simulator

In order to be able to test the generated SDT XML files, and to be able to simulate a smart home environment, a simulator application was implemented. The SDT simulator is a standalone application, which means that users have to upload a file locally, but this does not limit the application to get files from a public repository, where all the end users store their SDT XML files. Furthermore, the XML files are parsed using the SDT API, which uses the JAXB[4] parser.

A screenshot of the application is provided in Figure 4.13. The use of the simulator is quite trivial. Users have to upload an SDT XML file, and then the conditions are automatically displayed in

[4]Available at https://jaxb.java.net/

the condition section. Users can select the conditions they want to apply. When a condition is selected, it applies that the condition is true. By pressing the "simulate" button, the application parses the decision rules, and displays the actions that apply, according to the selected conditions. Furthermore, a list of the annotated lexons is displayed at the bottom of the application window. Apart from the fact that users can check the application of the SDT they have already created, the SDT simulator gives also the ability to easily check the agreed SDTs of the community before end users apply them to their smart environment.

4.4 Conclusion

In this chapter we prescribed the three components that were implemented for the purposes of the method that is proposed. The extension of GOSPL is for the collaborative construction of SDTs. The stakeholders of a community can collaboratively create SDTs which will be used afterwards by the users of the SDT editor. The latter is an Android application that gives the ability to edit and share the edited SDTs. That way we facilitate users for using personalized semantics. Furthermore, when users share their edited SDTs, a mechanism that helps the evolution of the SDT commitments is enabled. Last but not least, the SDT simulator is the application for the simulation of smart environments.

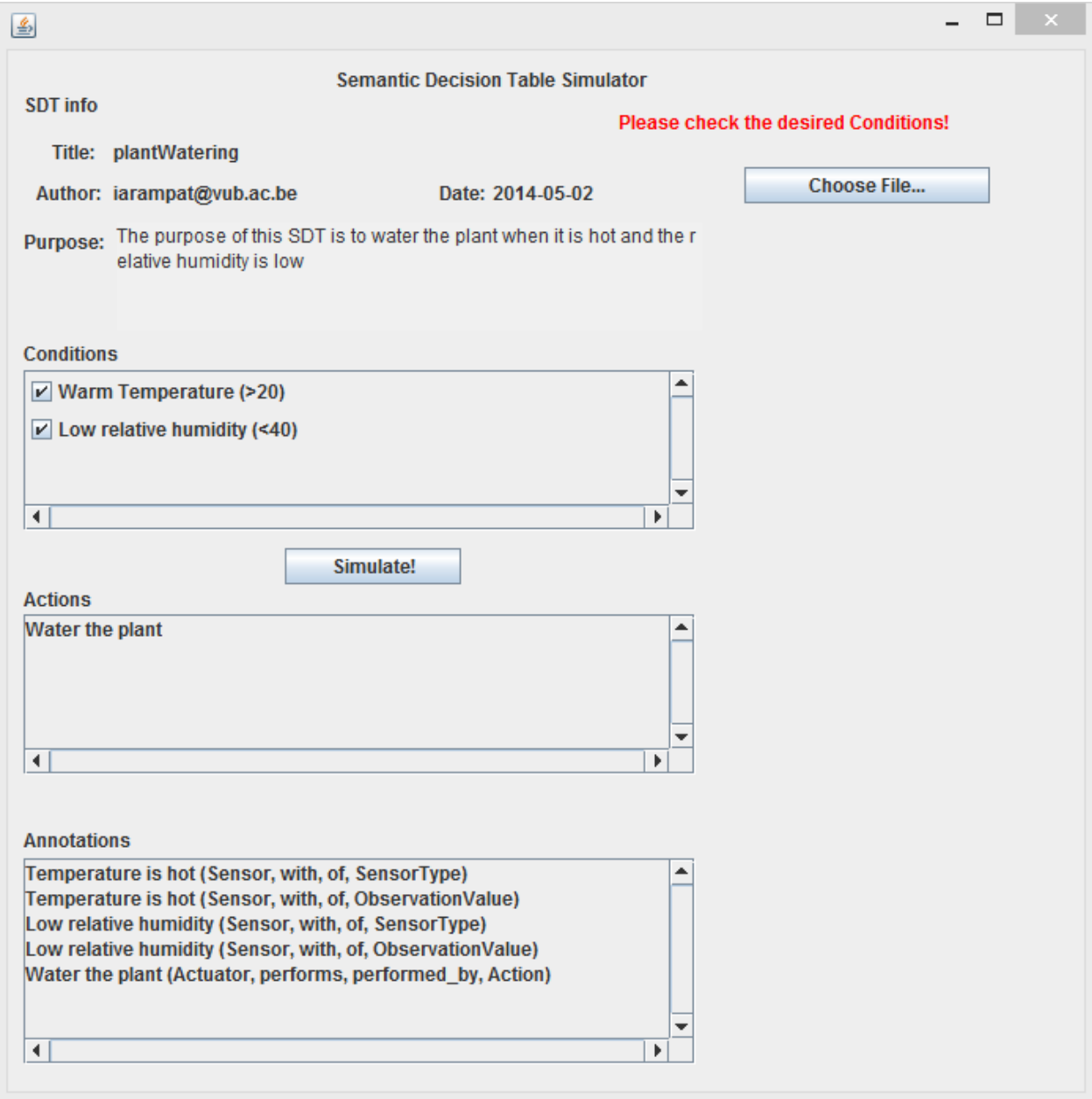

Figure 4.13: Screenshot of the SDT XML simulator

Chapter 5

Demonstration

In this section we prescribe a demonstration of our method and tools by means of a use case scenario. Imagine that in a secretary's office there are some plants to care. But the secretary sometimes is busy and out of the office. For that reason she wants to find a way to treat her plants automatically. Thus, she wants to have a smart watering system that could monitor the conditions of the environment and to water the plants when needed. The secretary has some sensors installed in her office. These sensors consist of a light sensor, a humidity sensor, and a temperature sensor. Apart from the sensors there are also some actuators inside the office. These actuators consist of a curtain opener and closer, a rail motor (to drag the flower next to the window) and a water actuator that is responsible for the watering of the plant. This environment is depicted in Figure 5.1.

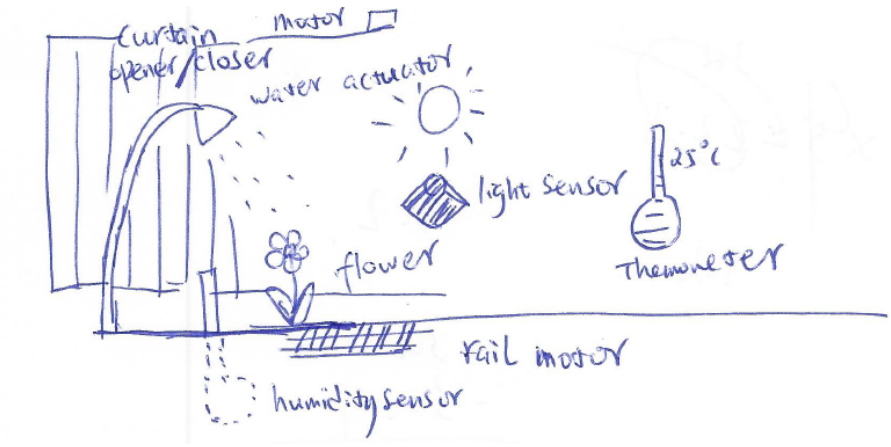

Figure 5.1: Intelligent environment use case [Tan14]

We will demonstrate how the secretary can create her own plan for the management of her smart watering system. In the beginning we prescribe how the creation of the plan takes place in

the community, and then how the secretary can specify her personal semantics, by the use of SDTs.

5.1 Collaborative construction of SDTs use case

To create and to edit an SDT, an agreement inside the community must take place. This is what we call SDT commitment. In order to create an SDT commitment several social processes must take place. To start with, the name of the SDT and the gloss of it must be declared. The agreed social process is depicted in Figure 5.2.

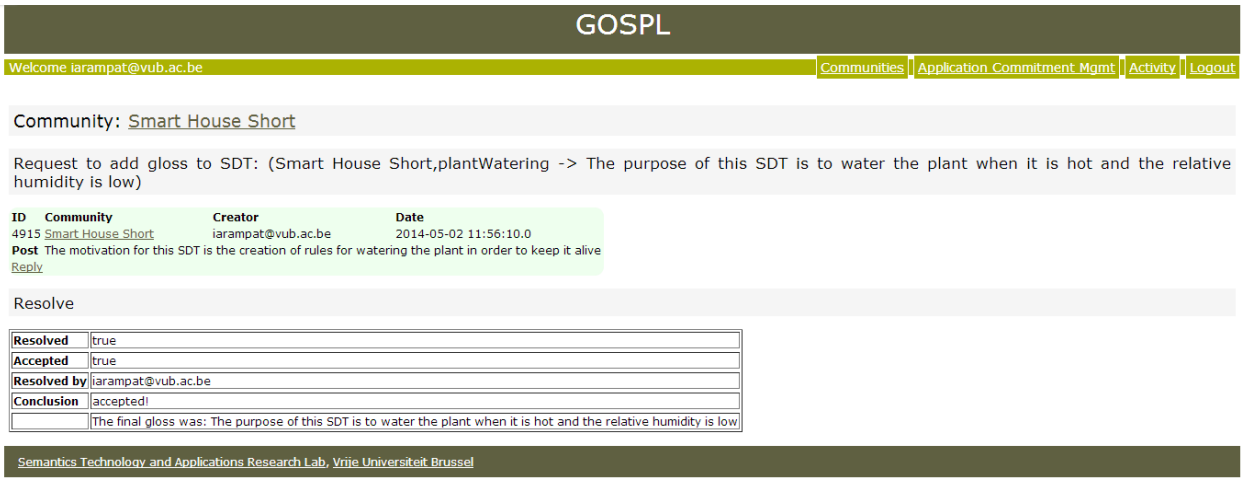

Figure 5.2: Social process for the SDT gloss

Once the aforementioned social process is accepted, the stakeholders of the community can start creating the conditions and the actions that will be included in the SDT. From the example above one can notice that the conditions that are needed in this use case are the ”hot temperature” and the ”low relative humidity.” Thus, the stakeholders can continue with the social processes for creating the conditions. In that case, the stakeholders have to agree on what ”hot” means for them, and how the ”low relative humidity” is defined. One should note that for the creation of the conditions we also use glosses. To this end, stakeholders of different communities can use these conditions by means of gloss equivalences and synonyms. Figure 5.3 displays the creation of the gloss for the condition ”Temperature is hot.”

In order to complete the creation of the conditions, the stakeholders must agree on the annotations and on the condition entry for each condition. For the purposes of this uses case, we assume that ”hot” is above 25 Celsius degrees and ”low relative humidity” is below 40%. Figure 5.4 depicts the form for the creation of the condition entry. Before the condition entries, the annotations have to

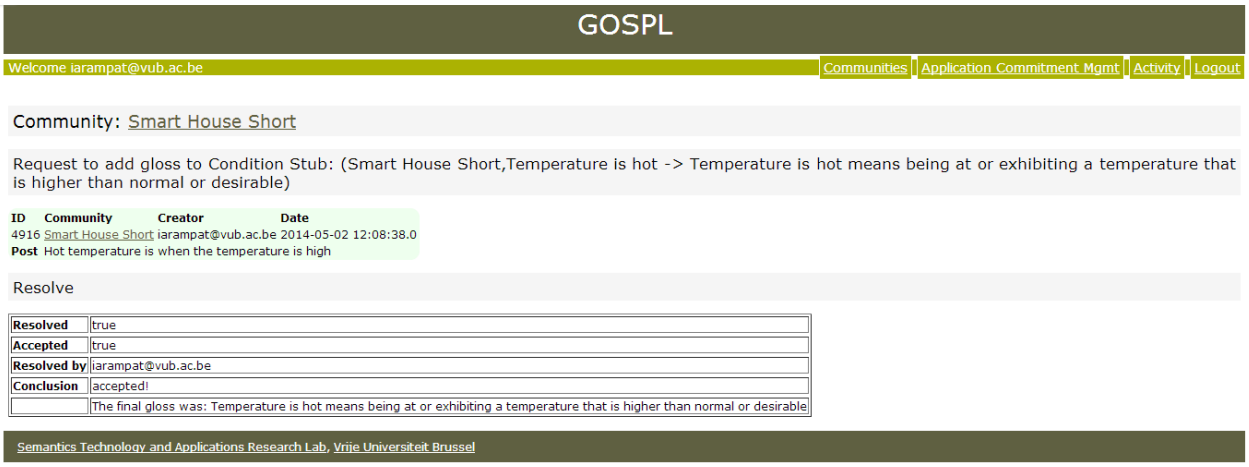

Figure 5.3: Social process for the "Temperature is hot" condition

be agreed between the stakeholders. When these steps are complete, the condition is displayed in the table with the conditions overview, under the "SDT" tab from the main menu, as it is displayed in Figure 5.5.

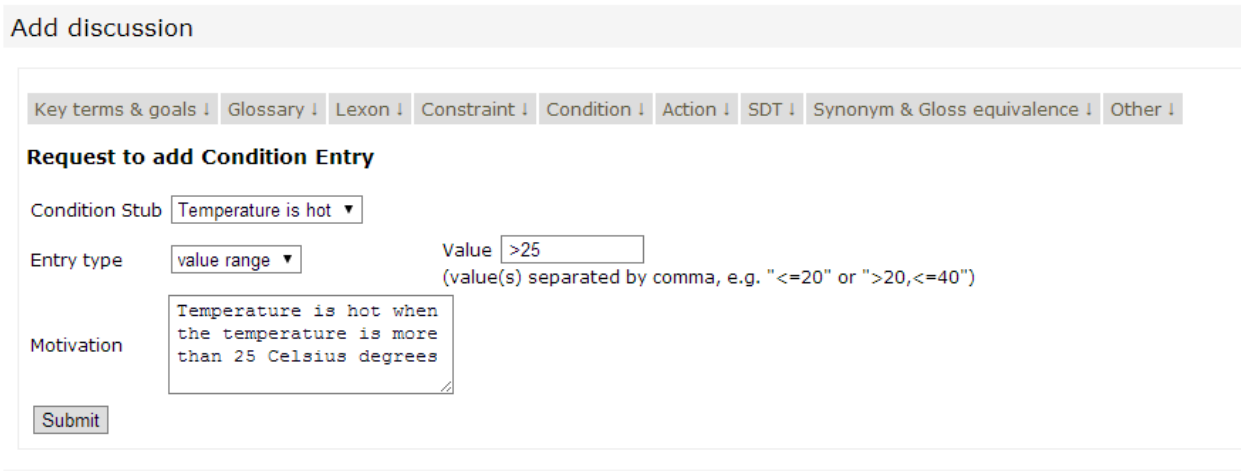

Figure 5.4: Form for the condition entry

When these steps are complete, the condition is displayed in the table with the conditions overview, under the "SDT" tab from the main menu, as it is displayed in Figure 5.5.

The same also applies to the actions. Stakeholders need first to agree on the gloss of the action, and then to add the annotations. One should note that action entries will be inserted in another part of the process. When conditions and actions are agreed, then they can be inserted to the SDT. The form on how to add the conditions and the actions is displayed in Figure 5.6. When the conditions and the actions are agreed and inserted to the SDT, these conditions and actions are visible in the

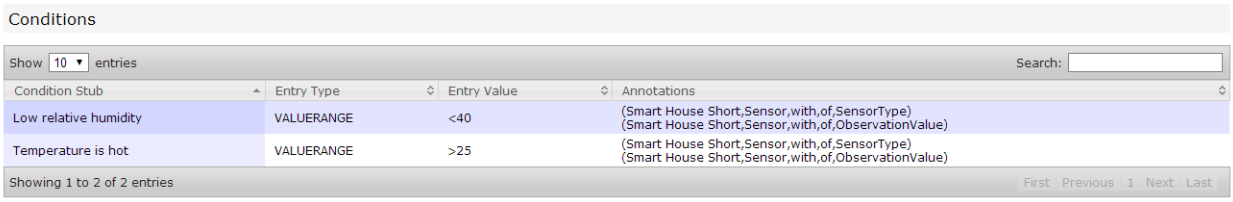

Figure 5.5: Overview of the community conditions

SDT overview table (Figure 5.7). The last part of the process is the creation of the decision rules. In this step, stakeholders should agree on the action entries, which finalize the step of the decision rules creation. Figure 5.8 displays how the stakeholders of the community can insert and interact for the agreement of the creation rules.

Figure 5.6: Form to add conditions and actions to an SDT

Figure 5.7: Overview of the SDTs

Last but not least, the generated XML file of the aforementioned SDT can be found when the stakeholders click on the SDT link under the SDT overview tab. The generated XML file is displayed as in Figure 5.9. The link that provides the XML file directly with its own link can be accessed by pressing the ”here” link. The XML code of the SDT used in this use case is presented in Appendix A.

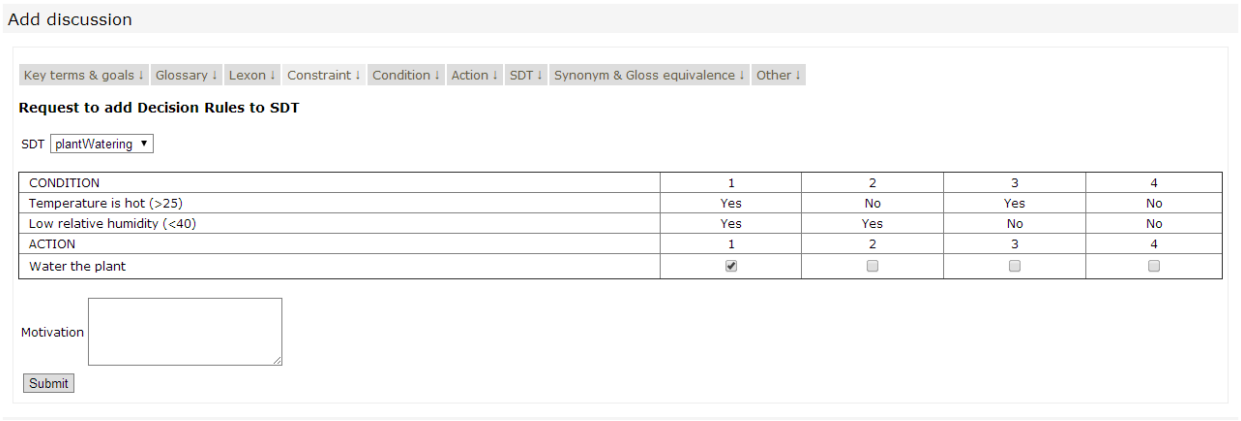

Figure 5.8: Form for the creation of the decision rules

Figure 5.9: XML file of the SDT

5.2 SDT editor use case

Users of the SDT editor are able to download and view the SDTs in their Android devices. Furthermore, they can edit the SDT XML files. More specifically, they can edit the decision rules and the conditions (condition stub and entry). The only prerequisite to store the files locally is the endpoint to the desired community. The first screen that users get when they open the application is displayed in Figure 5.10.

By tapping on the top button, the general management of the files is available (Figure 5.11). The button in the bottom is for the connection with the endpoint of the community. Users are then able to specify the endpoint and to download the SDT commitments (Figure 5.12). After the download, the SDT commitments are stored locally and users are able to specify their own semantics, by editing the condition/action stub labels and the condition entry values. Furthermore, the downloaded files are displayed in the general management activity.

Once the SDT commitments are downloaded, users are able to edit the decision rules and the

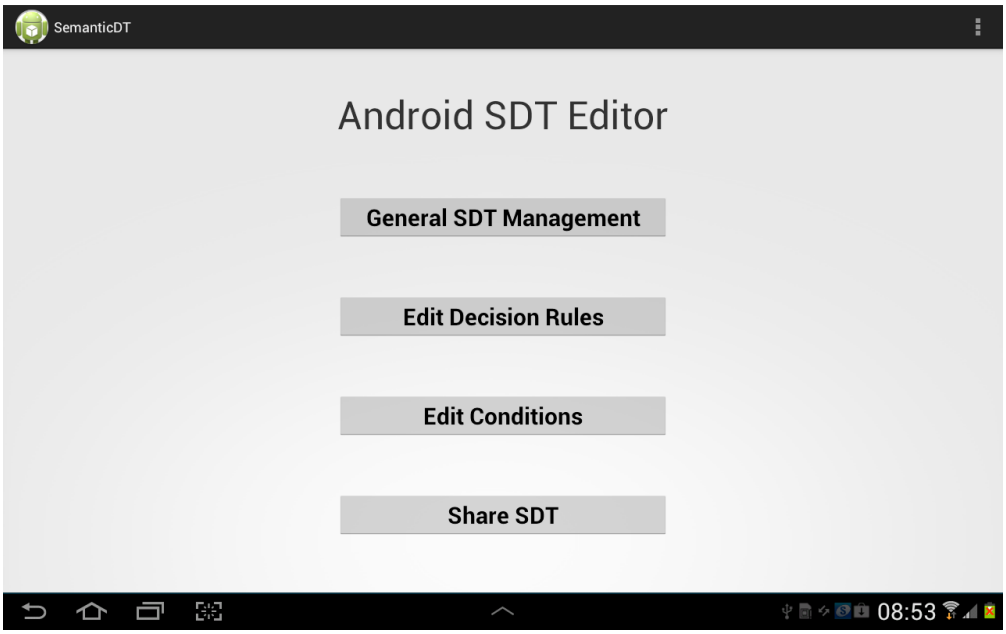

Figure 5.10: Main screen of the SDT editor

conditions. An example of the decision rules edit is displayed in Figure 5.13. When the button saved is tapped, the new decision rules are stored, together with the email of the user and the modification date.

The same applies for the edit of the conditions. In our use case scenario one can notice that the community agreed that ”hot temperature” is when the temperature is more than 25 Celsius degrees. In Figure 5.14 we select the condition that we want to change. In the next step we change this value range to the interpretation of the user, which is above 20 Celsius degrees. Furthermore, the name of the condition changes, as the user prefer to use the word ”warm temperature”, which ends up in another interpretation of the condition (Figure 5.15). To this end, an alert message is displayed to ask the user to confirm the new condition, preventing him from unwanted changes (Figure 5.16).

As we wanted to investigate a way to have these modified SDT files shared between users, we created an SDT repository and we give the ability to the users to share their personal semantics. This is done by tapping the last button from the main screen. The result is displayed in Figure 5.17. Then the user can upload the desired SDT file to the SDT repository. This action also enables the stakeholders of the community to access these files. That way the evolution of the collaborative construction of the SDTs is supported. The list with the modified SDTs is displayed in Figure 5.18. The stakeholders of the community can search and access the edited SDTs by clicking on the link in their names. The result of the aforementioned example is displayed in Figure 5.19

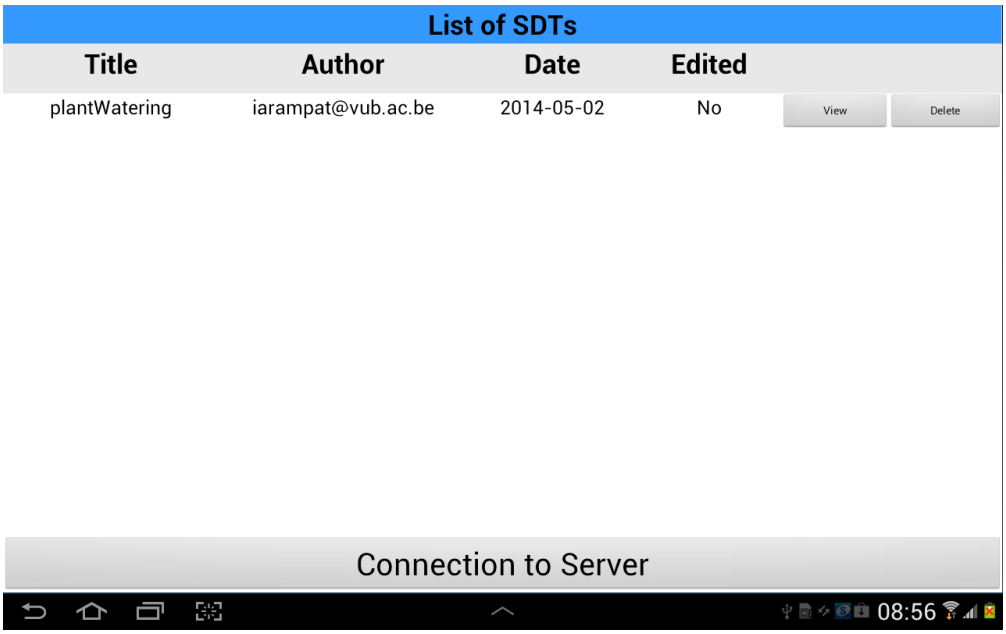

Figure 5.11: General management of the SDT files

5.3 Conclusion

In this section we demonstrate how to use the method for the collaborative construction of SDTs. We created an SDT commitment by using GOSPL. We used our SDT editor, by means of an Android application, to edit the SDT and to create personalized semantics.

In this demonstration we showed how the community can benefit from the users of the SDT editor. More specifically we prescribed how the SDT commitments can evolve with the use of the SDT repository by the users.

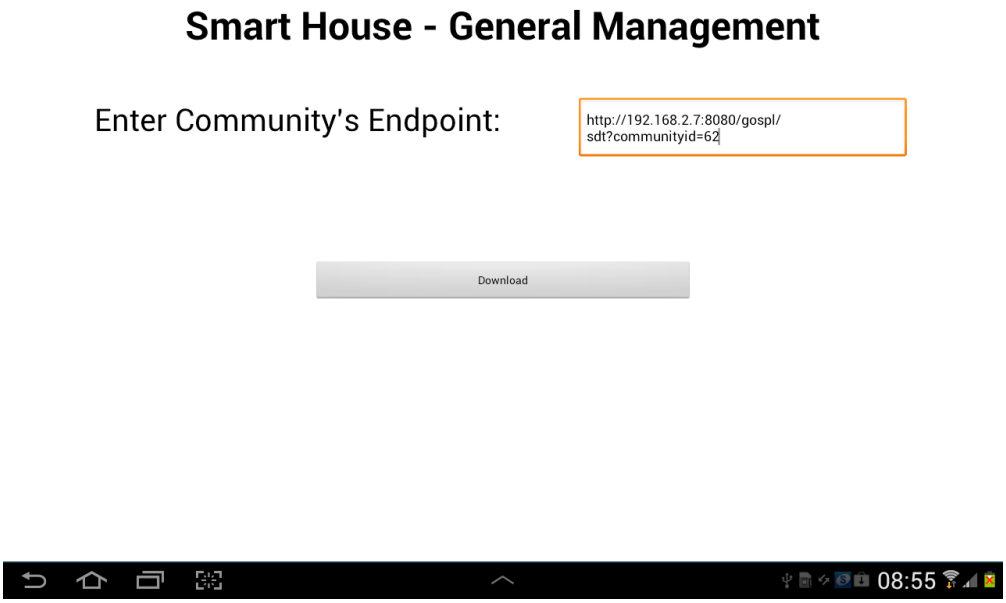

Figure 5.12: Download screen of SDT commitments

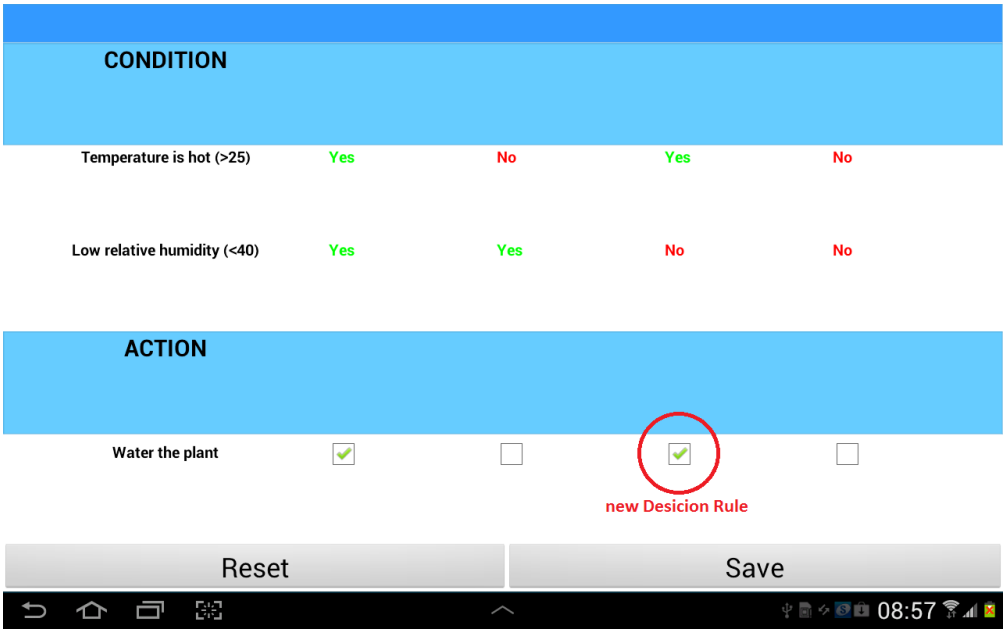

Figure 5.13: Edit screen of the decision rules

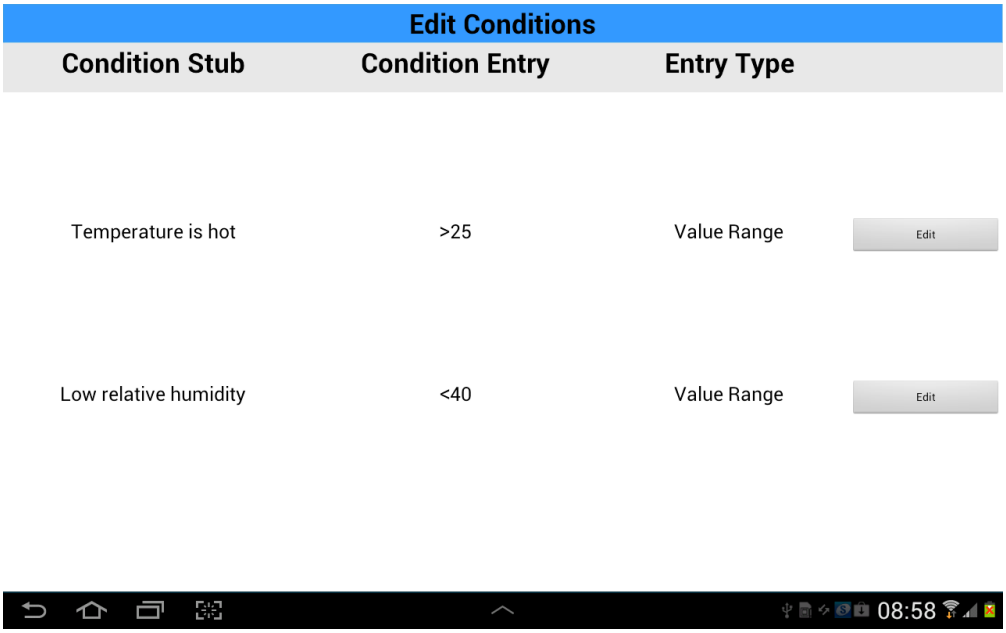

Figure 5.14: Edit screen of the conditions

Figure 5.15: Edited SDT after the ”Warm Temperature” insertion

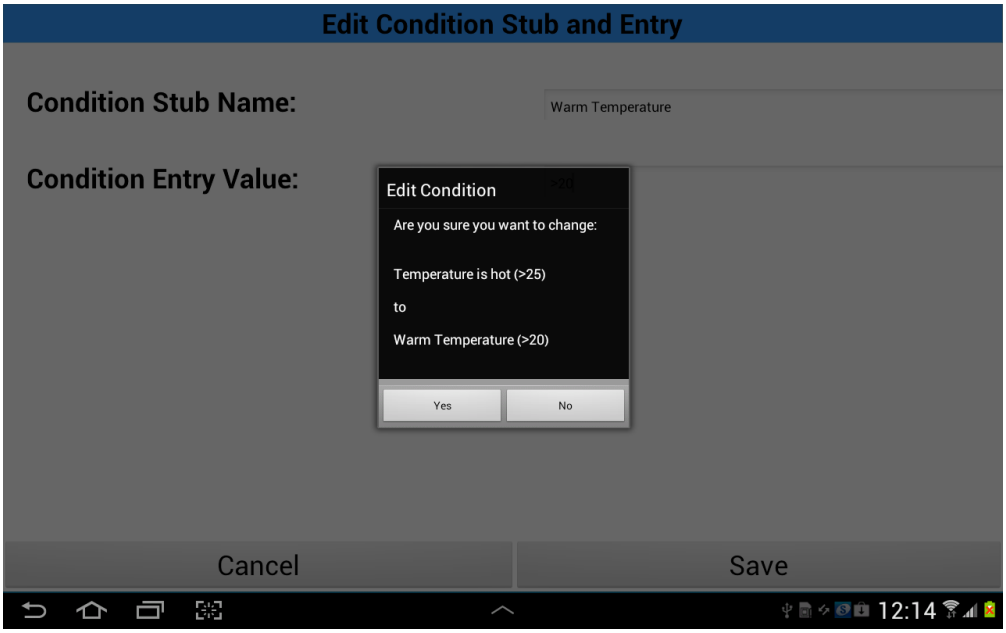

Figure 5.16: Alert message for the confirmation of the condition change

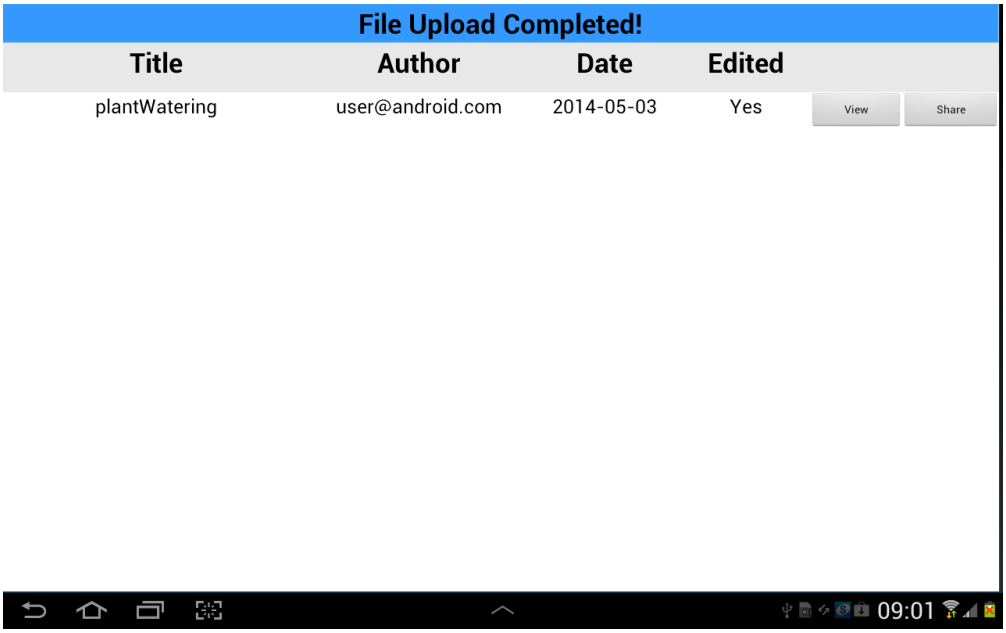

Figure 5.17: Screen to share the edited SDT

Figure 5.18: Shared SDTs list in GOSPL

Figure 5.19: Access of the edited SDT in GOSPL

Chapter 6

Conclusion

In this chapter we reflect on the research questions, mentioned in the beginning of this thesis, and on how we answer them. To this end, a summary of what we feel that are the most important parts is mentioned, together with the limitations we faced during this thesis.

6.1 Research Questions and Objectives

In this thesis we asked ourselves the following questions:

- **Question 1** *How can we support the collaborative construction of SDTs?*
- **Question 2** *How can we extend SDT commitments to user-defined rules (i.e. different layers of semantic rules?)*
- **Question 3** *How can the combination of both levels of SDTs evolve the SDT construction method?*

In order to answer the aforementioned questions we set the following objectives:

- **Objective 1** *Provide an analysis and the state-of-the-art on the domain of management of business rules for intelligent environments (Chapter 2)*
- **Objective 2** *Develop a method for the collaborative construction of SDTs (Chapter 3)*
- **Objective 3** *Implement the tools that will support the stakeholders of the communities to create SDTs and give the ability to the users of the SDT editor to create their own rules by means of personalized semantics (Chapter 4)*
- **Objective 4** *Demonstration of these tools and investigation of a way to help the evolution of SDTs inside communities (Chapter 5)*

In the next section we prescribe the way on how we met these objectives.

6.2 Contributions

The main problem statement after the analysis of the related work on context-aware semantic decision making is that current architectures for the management of rules over intelligent environments lack on personalized semantics. To the best of our knowledge, there are no components that support different levels of semantics. In this thesis we provide two levels in that respect, and more specifically the level of the community (implemented in GOSPL) and the level of users (SDT editor). We argue that by the use of SDTs there are many advantages. To name some of them, SDTs are easy to use and to observe. This means that SDTs can be used by non-technical users, and not only by domain experts. Furthermore, by the use of natural language, these users can easily create their decision rules, by using their own interpretations. In the next sections we will describe how we came up to the aforementioned research questions, and how we met the requirements to answer them.

6.2.1 Background and State-of-the-Art

Firstly we introduced the background information regarding the semantic web technologies. Then, the analysis of the related work took place. During this analysis, we found out that the ontology-based architectures and its implementations lack of user-defined rules. In Table 2.1 an overview of the related work is provided. One can notice that the community aspect on all projects, apart from SESAME-S is missing. Furthermore, most of the projects lack of personalized semantics, by means of user-defined rules. All rules are created by the domain experts, and then are used by the end users. This means that they do not facilitate users to define their own rules, but only to use those that are pre-defined. This was the reason we tried to give room to the users to participate in the process of the creation of the rules, by introducing a method for the collaborative construction of SDTs.

6.2.2 Method for the collaborative construction of SDT

From the related work analysis we identified a gap between the users and the creation of the decision rules. This was the reason we used GOSPL for the collaborative construction of SDTs. GOSPL is a method and a tool for hybrid ontology engineering. This means that communities are first-class citizens and all agreements are parameterized within a community. By providing the method for the collaborative construction of SDTs we give the ability to the users to define their

semantics and to agree upon them. This method consists of a sequence of social processes inside communities. To this end, glosses facilitate users in describing the SDTs, the conditions, and the actions by their (implicit) meaning. The latter makes glosses work as a driver for the agreements inside the communities.

6.2.3 Tools

Apart from the method that is proposed in this thesis, three tools were implemented that provide support to it.

The first one is the extension of GOSPL for the construction of the hybrid SDTs. The stakeholders of a community can interact with each other via the social processes that were added in GOSPL as extensions for the creation of the SDTs. As the method proposes, stakeholders can create the conditions and the actions that will later on use in the SDTs. For each step an agreement inside the community must take place. In the end, when the SDT commitment is complete, a link to that is created in order to be used by the users of the SDT editor. The latter is the second tool that was implemented for the purposes of this thesis.

The SDT editor is an Android application that allows users to edit, if needed, the conditions and the decision rules of an SDT. Users of the SDT editor can download and store locally the SDT commitments by the use of the community's endpoint. When the files are stored locally, the ability to edit the conditions and the decision rules is facilitated. Furthermore, users can share their edited SDTs, not only with other users of the SDT editor, by the use of the SDT repository, but also with the stakeholders of the community. That way the evolution of the collaborative construction of SDTs is supported.

Last but not least, a simulator for the SDT files was created. Users can use this simulator to simulate the conditions and the actions of the SDTs they have created. It is a tool that promotes users to check the SDTs for possible logical error after the creation of their own decision rules.

6.2.4 Demonstration

To demonstrate the use of our method and tools, we applied a use case scenario. To start with, we used GOSPL for the creation of an SDT commitment. To create this SDT commitment, the method that is proposed in this thesis was used. Then we prescribed how users of the SDT editor can edit the SDT commitment. Once the edit was complete, we shared the edited SDT with other users and the community from which the SDT was downloaded in the first place. Last but not

least we prescribed how the stakeholders of the community can use the edited SDTs to derive new conditions and decision rules, if they think it is appropriate.

6.3 Limitations and Future Work

In this thesis we have presented a method and the tools for the collaborative construction of SDTs. Furthermore, we implemented the tools that support the creation of hybrid SDTs in GOSPL, and how users can edit the SDT commitments by defining their own defined rules. The tools that are presented are still research prototypes and are used only for demonstration purposes of the proposed method.

Regarding the SDT editor, login functionality will be implemented, which will help to the sharing of the edited SDTs. Furthermore, by the login functionality it will be available in GOSPL directly the generation of a discussion, by means of a social process, when an SDT editor user shares an edited SDT. Currently, this is not possible because in order to create a social process in GOSPL, one should be member of the community.
Another issue with the SDT editor is how we can guarantee the consistency of the edited SDTs. There exist methods for the validation and the verification of an SDT [Tan10], but this is left a future work.
Last but not least, current implementation of the SDT editor supports the access of SDTs up to four conditions. The reason for that is the limitation we have from the small screen of the Android device. A way for displaying more conditions, and in extension to that more columns in the SDT files, will be investigated, and this is part of our future work.

Regarding the GOSPL extension, we want to run evaluation tests to investigate whether the method that is proposed if favorable, and that we provide a friendly user interface to the stakeholders of the communities. To this end, improvements of the extended social processes and of the user interface that is provided for them could be introduced. This is another part of our future work. Last but not least, a way for the use of the gloss-equivalence and the synonym concepts between communities regarding SDTs will be investigated.

Appendices

Appendix A

Generated SDT XML file

In this appendix, we present an example of a generated SDT XML file, created in our use case scenario:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <SDT title="plantWatering" date="2014-05-02" author="iarampat@vub.ac.be" conditionNbr=
       "2" actionNbr="1" columnNbr="5">
3      <purpose xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://
           www.w3.org/2001/XMLSchema" xsi:type="xs:string">The purpose of this SDT is to
           water the plant when it is hot and the relative humidity is low</purpose>
4      <conditions>
5          <condition>
6              <label xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="
                   http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Temperature is
                   hot (&gt;25)</label>
7              <conditionEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Yes</
                   conditionEntry>
8              <conditionEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">No</
                   conditionEntry>
9          </condition>
10         <condition>
11             <label xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="
                   http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Low relative
                   humidity (&lt;40)</label>
12             <conditionEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Yes</
                   conditionEntry>
13             <conditionEntry xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">No</
```

```
                conditionEntry>
14          </condition>
15      </conditions>
16      <action>
17          <label xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://
                www.w3.org/2001/XMLSchema" xsi:type="xs:string">Water the plant</label>
18      </action>
19      <content>
20          <cell xposition="0" yposition="0" item="CONDITION"/>
21          <cell xposition="0" yposition="1" item="␣"/>
22          <cell xposition="0" yposition="2" item="␣"/>
23          <cell xposition="0" yposition="3" item="␣"/>
24          <cell xposition="0" yposition="4" item="␣"/>
25          <cell xposition="1" yposition="0" item="Temperature␣is␣hot␣(&amp;gt;25)"/>
26          <cell xposition="1" yposition="1" item="Yes"/>
27          <cell xposition="1" yposition="2" item="No"/>
28          <cell xposition="1" yposition="3" item="Yes"/>
29          <cell xposition="1" yposition="4" item="No"/>
30          <cell xposition="2" yposition="0" item="Low␣relative␣humidity␣(&amp;lt;40)"/>
31          <cell xposition="2" yposition="1" item="Yes"/>
32          <cell xposition="2" yposition="2" item="Yes"/>
33          <cell xposition="2" yposition="3" item="No"/>
34          <cell xposition="2" yposition="4" item="No"/>
35          <cell xposition="3" yposition="0" item="ACTION"/>
36          <cell xposition="3" yposition="1" item="␣"/>
37          <cell xposition="3" yposition="2" item="␣"/>
38          <cell xposition="3" yposition="3" item="␣"/>
39          <cell xposition="3" yposition="4" item="␣"/>
40          <cell xposition="4" yposition="0" item="Water␣the␣plant"/>
41          <cell xposition="4" yposition="1" item="*"/>
42          <cell xposition="4" yposition="2" item="␣"/>
43          <cell xposition="4" yposition="3" item="␣"/>
44          <cell xposition="4" yposition="4" item="␣"/>
45      </content>
46      <annotation>
47          <annotated relation="Sensor" role="with" corole="of">
48              <DecisionItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">
                    Temperature is hot</DecisionItem>
49              <Target conceptName="SensorType"/>
50          </annotated>
51          <annotated relation="Sensor" role="with" corole="of">
52              <DecisionItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">
                    Temperature is hot</DecisionItem>
53              <Target conceptName="ObservationValue"/>
54          </annotated>
```

```
55          <annotated relation="Sensor" role="with" corole="of">
56              <DecisionItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Low
                    relative humidity</DecisionItem>
57              <Target conceptName="SensorType"/>
58          </annotated>
59          <annotated relation="Sensor" role="with" corole="of">
60              <DecisionItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Low
                    relative humidity</DecisionItem>
61              <Target conceptName="ObservationValue"/>
62          </annotated>
63          <annotated relation="Actuator" role="performs" corole="performed_by">
64              <DecisionItem xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xs="http://www.w3.org/2001/XMLSchema" xsi:type="xs:string">Water
                    the plant</DecisionItem>
65              <Target conceptName="Action"/>
66          </annotated>
67      </annotation>
68  </SDT>
```

Bibliography

[AvH09] G. Antoniou and F. van Harmelen. Web Ontology Language: OWL. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 67–92, Berlin, Heidelberg, Germany, 2009. Springer-Verlag.

[BC08] D. Bonino and F. Corno. DogOnt - Ontology Modeling for Intelligent Domotic Environments. In *International Semantic Web Conference*, pages 790–803, 2008.

[BCC08] D. Bonino, E. Castellina, and F. Corno. The DOG Gateway: Enabling Ontology-Based Intelligent Domotic Environments. pages 1656–1664, 2008.

[BG04] D. Brickley and R. V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, 2004.

[BKvH02] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International Semantic Web Conference*, pages 54–68, 2002.

[BLHL01] T. Berners-Lee, J. Hendler, and Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[CFJ04] H. Chen, T. W. Finin, and A. Joshi. Semantic Web in the Context Broker Architecture. In *PerCom*, pages 277–286, 2004.

[CP+04] H. Chen, F. Perich, D. Chakraborty 0001, T. W. Finin, and A. Joshi. Intelligent Agents Meet Semantic Web in a Smart Meeting Room. In *AAMAS*, pages 854–861, 2004.

[CPFJ04] H. Chen, F. Perich, T. W. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *MobiQuitous*, pages 258–267, 2004.

[DAS01] A. K. Dey, G. D. Abowd, and D. Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. pages 97–166, 2001.

[Deb13] C. Debruyne. *Grounding Ontologies with Social Processes and Natural Language*. PhD thesis, Vrije Universiteit Brussel, 2013.

[DLCM10] P. De Leenheer, S. Christiaens, and R. Meersman. Business Semantics Management: A Case Study for Competency-Centric HRM. *Computers in Industry*, 61(8):760–775, 2010.

[DM11] C. Debruyne and R. Meersman. Semantic Interoperation of Information Systems by Evolving Ontologies through Formalized Social Processes. In J. Eder, M. Bieliková, and A.M. Tjoa, editors, *Proceedings of the 15th international conference on Advances in databases and information systems*, ADBIS '11, pages 444–459, Berlin, Heidelberg, Germany, 2011. Springer-Verlag.

[DM12] C. Debruyne and R. Meersman. GOSPL: A Method and Tool for Fact-Oriented Hybrid Ontology Engineering. In T. Morzy, T. Härder, and R. Wrembel, editors, *Proceedings of the 16th international conference on Advances in databases and information systems*, ADBIS '12, pages 153–166, Berlin, Heidelberg, Germany, 2012. Springer-Verlag.

[DT12] Y. T. Demey and T. K. Tran. Using SOIQ(D) to Formalize Semantics within a Semantic Decision Table. In *Rules on the Web: Research and Applications*, pages 224–239. Springer, 2012.

[DTM13] C. Debruyne, K. Tran, and R. Meersman. Grounding Ontologies with Social Processes and Natural Language. *Journal on Data Semantics*, 2(2-3):89–118, 2013.

[FH03] E. Friedman-Hill. *JESS in Action*. Manning Greenwich, CT, 2003.

[FTK+13] A. Fensel, S. Tomic, V. Kumar, M. Stefanovic, S. V. Aleshin, and D. O. Novikov. SESAME-S: Semantic Smart Home System for Energy Efficiency. pages 46–57, 2013.

[GG95] N. Guarino and P. Giaretta. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. J. I. Mars, editor, *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*, pages 25–32, Amsterdam, The Netherlands, 1995. IOS Press.

[GHM+08] B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler. OWL 2: The Next Step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[GPFLC03] M. Gomez-Perez, A. Fernandez-Lopez, and O. Corcho. *Ontological Engineering*. Springer-Verlag, Berlin, Heidelberg, Germany, 2003.

[GPZ05] T. Gu, H. K. Pung, and D. Zhang. A Service-Oriented Middleware for Building Context-Aware Services. pages 1–18, 2005.

[Gru93a] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[Gru93b] T. R. Gruber. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *International Journal of Human-Computer Studies*, 43(5-6):907–928, 1993.

[GVdMB08] A. Gerber, A. Van der Merwe, and A. Barnard. A Functional Semantic Web Architecture. In *The Semantic Web: Research and Applications*, pages 273–287. Springer, 2008.

[GWPZ04] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang. An Ontology-bBsed Context Model in Intelligent Environments. In *Proceedings of communication networks and distributed systems modeling and simulation conference*, volume 2004, pages 270–275, 2004.

[HM08] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2008.

[HPSB+04] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. *W3C Member submission*, 21:79, 2004.

[JM08] M. Jarrar and R. Meersman. *Ontology Engineering - The DOGMA Approach*, volume 4891, chapter 3, pages 7–34. Springer-Verlag, Berlin, Heidelberg, Germany, 2008.

[LS99] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, World Wide Web Consortium (W3C), 1999.

[MD10] R. Meersman and C. Debruyne. Hybrid Ontologies and Social Semantics. In F. K. Hussain and E. Chang, editors, *Proceedings of the 4th IEEE International Conference on Digital Ecosystems and Technologies*, DEST '10, pages 92–97, Washington, DC, USA, 2010. IEEE Computer Society.

[MPSP+09] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, and U. Sattler. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. *W3C recommendation*, 27:17, 2009.

[PS08] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, World Wide Web Consortium (W3C), 2008.

[RDM+07] V. Ricquebourg, D. Durand, D. Menga, B. Marhic, L. Delahoche, C. Logé, and A. Jolly-Desodt. Context Inferring in the Smart Home: An SWRL Approach. In *AINA Workshops (2)*, pages 290–295, 2007.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.

[SG11] V. Sugumaran and J. Gulla. *Applied Semantic Web Technologies*. CRC Press, 2011.

[SMJ02] P. Spyns, R. Meersman, and M. Jarrar. Data Modelling versus Ontology Engineering. *SIGMOD Record*, 31(4):12–17, 2002.

[SPG+07] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. pages 51–53, 2007.

[Spy05] P. Spyns. Object Role Modelling for Ontology Engineering in the DOGMA Framework. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 710–719. Springer, 2005.

[Tan10] Y. Tang. Semantic Decision Tables-A New, Promising and Practical Way of Organizing Your Business Semantics with Existing Decision Making Tools. *LAP LAMBERT Academic Publishing AG & Co. KG, Saarbrucken*, 2010.

[Tan14] Y. Tang. STARLab website, Master Thesis Description, 2012 (accessed June 3, 2014). http://www.starlab.vub.ac.be/website/node/791.

[TFP10] S. Tomic, A. Fensel, and T. Pellegrini. SESAME Demonstrator: Ontologies, Services and Policies for Energy Efficiency. In *I-SEMANTICS*, 2010.

[TM07a] Y. Tang and R. Meersman. On Constructing Semantic Decision Tables. In *Database and Expert Systems Applications*, pages 34–44. Springer, 2007.

[TM07b] Y. Tang and R. Meersman. Towards Building Semantic Decision Table with Domain Ontologies. *ICITM*, pages 14–22, 2007.

[TM09] Y. Tang and R. Meersman. 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, USA*, 2009.

[TM11] Y. Tang and R. Meersman. Towards Directly Applied Ontological Constraints in a Semantic Decision Table. In *Rule-Based Modeling and Computing on the Semantic Web*, pages 193–207. Springer, 2011.

[VRT10] P. A. Valiente-Rocha and A. L. Tello. Ontology-Based Expert System for Home Automation Controlling. In *IEA/AIE (1)*, pages 661–670, 2010.

[WZGP04] X. Wang, D. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *PerCom Workshops*, pages 18–22, 2004.