OWL Formal Ontology for the Social Structure in the OperA Meta-model

THESIS

Submitted in the partial fulfillment of
The requirements for the degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE
TRACK INFORMATION ARCHETECTURE

By

Weili Tuo
Born in Chongqing, China

Web Information Systems
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, The Netherlands
http://wis.ewi.tudelft.nl

Author: Weili Tuo
Student ID: 4192990
Email: tuoweili19910606@gmail.com

Abstract:

The rise in various activities of organizations in society increases the demand for organizational model. Because the organizational model is used for managements and such activities will increase management demand for organizations. The organizational model (OM) defines the way in which activities such as task allocation, coordination and supervision are directed towards the achievement of organizational aims. OperA is an existing framework for organizational modeling. OperettA, which is based on the OperA approach, is a graphical environment for the analysis, management, and specification of an organizational model. There are two ways to create OperA organizational model: through editing XML documents and through OperettA. Within both ways, there are limitations and disadvantages, such as editing inconsistencies or semantic inconsistencies. In this thesis, we present the OWL OperA meta-model, which is redelivered from the OperA approach description into OWL2, and OWLperettA as a plug-in for OperettA to export the Social structure (SS) of an Organizational Model designed by OperettA into OWL2 format. The OWL OperA meta-model provides a more expressive way of defining or specifying the elements in an organizational model. OWLperettA, in combination with OperettA, presents a convenient environment to design the SS of an OM. Using reasoner and SPARQL query for checking the OWL2 ontology, we are able to overcome the limitations and disadvantages of OperA and OperettA described above for specifying and analyzing the Social structure of an organizational model.

Keywords: OperA, OperettA, OWL2, Organizational Modeling, the Social Structure, Ontology, meta-model, Semantic.

Graduation Committee

Chair: Prof. dr. ir. G.J. Houben, Faculty EEMCS, TU Delft
University supervisor: Dr. H.M. Aldewereld, Faculty TPM, TU Delft
Committee Member: Dr.M.V,Dignum, Faculty TPM, TU Delft
Committee Member: Dr.M.B.van Riemsdijk, Faculty EEMCS, TU Delft

Preface

This thesis has been produced as my final piece of work for my study at Delft University of Technology as a master Information Architect (IA) student in Computer Science. I performed my thesis within the Information and Communication Technology section. During the time at TUD, I had take courses from the domains of Business and IT. In order to support my future work in information, I decided to choose a topic related to the information. The experience of performing the challenges in this thesis makes me improve a lot in dealing with research problems.

I would like to deeply thank my daily supervisor Dr.Huib Aldewereld by this opportunity. Without the help and guidance of Huib, the thesis would not be possible to be finished. He provided all kinds of comments to me while I was performing this thesis. He pushed me to the right direction when I was confused and help me a lot by frequent meetings. I would like to thank my professor Geer Jan Houben. He allowed me to perform my thesis in ICT section and make me understand how to think in research. He also gave me advices for evaluating in creatively and powerful ways.

I would specially thank Dr.Virginia Dignum, who introduce me to the topic of this thesis and help me solve the problems I met. Furthermore, my thanks also go to Dr.Birna van Riemsdijk for providing me with feedbacks, guidance and critiques. I have learnt a lot from all of them.

Weili Tuo
Delft. The Netherlands
May 5th.2014

Contents

目录

1. Introduction

According to (Epstein, 2006)agent-based computational modeling is changing the face of social science. The purpose of any society is to allow its members to coexist in a shared environment and pursue their respective goals in the presence or in co-operation with others. Organizational society as (V. Dignum, Meyer, Dignum, & Weigand, 2003) described is an environment consists of different agents and interactions between different agents. These agents were revealed by (Ferber, Gutknecht, & Michel, 2004) as independent actors interacting together to coordinate their behavior and often cooperate to achieve some collective goal. An organization can be defined as a specific solution created by more or less autonomous actors to achieve common objectives. So, organizational modeling has been advocated to specify systems which can represent the regulating structure explicitly and independent from the acting components. OperA described by (M. Dignum, 2003) has been developed to provide an expressive way for defining open organizations distinguishing explicitly between the organizational aims, and the agents who act in it.

As described above, we are able to concisely indicate that OperA is a meta-model or framework for design organizational model. OperettA, described in (Aldewereld & Dignum, 2011), is a tool to support the design, analysis and development of organizational model (OM) using the OperA through as EMF based representation of the OperA meta-model. According to (Okouya & Dignum, 2008), we will see that some problems will always occur when designers design the OM through editing document and OperettA. These problems will be described in the Section 1.1.

(M. Dignum, 2003) has introduce the Logic for Contract Representation (LCR), to describe interactions between different agents or social states in organizational model. However, LCR is logic and it make OperA framework hard to reason. In addition, because OperettA is based on (Budinsky, 2004), It does not contain an expressive semantics too, which means OperettA also cannot reason about consistency. These limitations in the OperA meta-model cause the problems of OperettA for the designer. By perfecting the development of the OperA framework, designers can design the OM easier and more accurately through OperettA, so we propose to convert OperA meta-model into a more expressive ontology.

Indicated by the above demand, we suggest several potential solutions for them. (Klyne & Carroll, 2006), McGuinneSS (McGuinness & Van Harmelen, 2004), W3C (Group, 2009) are suggested potential solutions for our conversion of the OperA framework to ontology. As (Okouya et al., 2008) has described that, RDF, OWL, OWL2 are all powerful language for describing ontology. Comparing RDF, OWL and OWL2, They are all RDF based and can express semantic meaning of data. OWL has more vocabularies than RDF to describe ontology, and OWL2, as an extension of OWL, further complete the syntax of OWL.

	EMF	OWL2
Features	✓ Standardized ✓ Structured ✓ Too supported (eclipse) ✓ Expressiveness ✓ Inheritance ✓ Constraint definition ✓ Maintainability ✓ Automatic code generated (eclipse)	✓ Standardized ✓ Structured ✓ Tool supported (Protégé) ✓ Expressiveness ✓ Inheritance ✓ Maintainability ✓ There exist a reasoner for classification and consistence of the ontology ✓ Semantic Query language can be used to query the ontology and a set of Instance data stored in a data repository ✓ Semantics ✓ Better Connectability ✓ Stronger Constraint definition

Table1.1 Comparison between EMF and OWL for designing OperA meta-mdoel

Table1.1 has shown that, comparing to EMF, OWL2 can cover most of EMF's features. In addition, OWL2 has more features that EMF doesn't have. We can see that, except for the above features, OWL2 don't use UNA as well. However, the OWL2 can be inferred by their semantic expressions to identify the elements. After comparing features of EMF and OWL2 in representing the OperA meta-model, we can introduce some advantages of OWL2 in the following:

- **Semantics:** In OWL2, all the elements in the ontology can be assigned semantic meaning. This can be used to check class consistency, allow for ontology query or any other thing. It is the most important feature that OWL2 can support in contrast to the EMF model design. With the semantic features, a lot of implicated relationships, or meaning of text can be contained in the OWL2 ontology design. This is also the primary cause for our solution in Section1.2.
- **Stronger constraint definition:** OWL2 provides a rich set of primitives and allows for consistency checking by using additional axioms and several OWL2 class and property expressions.
- **Connectability:** it means that OWL2 is easier to connect to different systems. There are multiple systems that are able to develop "similar OperettA" software based on OWL ontology. (Knublauch, Fergerson, Noy, & Musen, 2004) provides a lot of methods to realize OWL related implementation it.

Of these advantages, we will try to evaluate only the first two. As shown by the Table above, we think that OWL2 may be more powerful than EMF to design the OperA meta-mode, because it has more features like the powerful query language, connectability, semantics, and stronger constraint definition. Semantics, which is the core feature of OWL2, gives more meaning to elements and relationships in ontology. Stronger constraint definitions define elements constraints in ontology more detailed. These two features will make the meta-model more expressive. The connectability means better connection between different systems, which means more compatibility and reusability. All the advantages are caused by the differences between the

EMF and OWL2. Some of these improvements will be evaluated through the case studies (examples) which were already built for the EMF based meta-model.

After analyzing of Table1.1 above in order to perfect the OperA framework, we select to use OWL2 as the basis of converting the OperA meta-model. We propose to convert the OperA meta-model represented through EMF into OWL2 based ontology. We also intent to show the advantages of our converted model compare to the OperA meta-model designed in EMF. Below Figure1.1 show our objective of project.

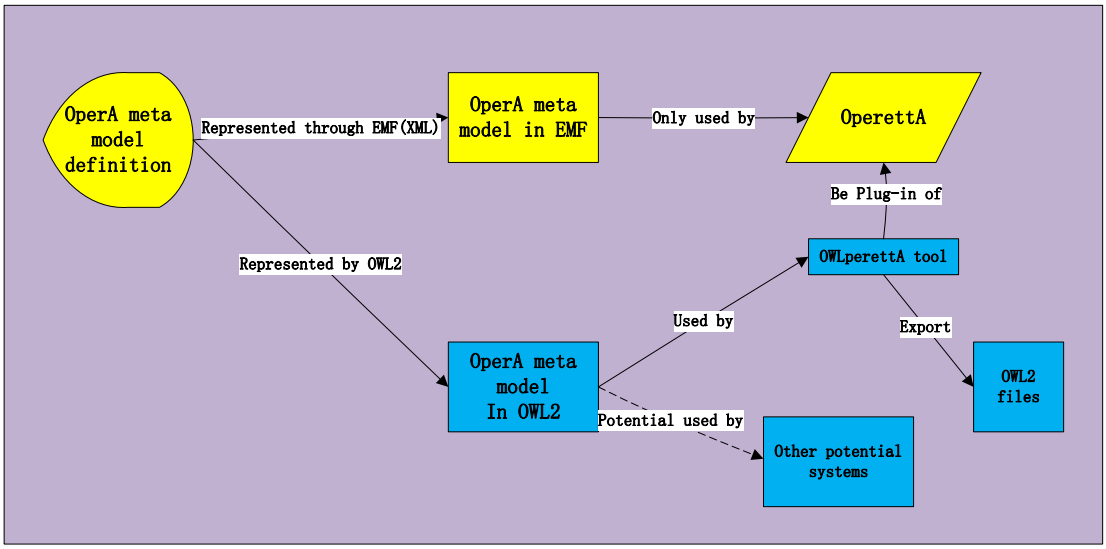


Figure1.1 Project Objectives

Overall, the project will produce the OperA meta-model which was represented in EMF, in OWL2. Through the project, an ontology called OWL OperA meta-model will be reproduced after converting the original model OperA meta-model represents through EMF, which we will call the OperA meta-model in the following Chapters. After we design the OWL OperA meta-model, we should be able to design cases with this model. However, editing OWL2 documents manually is difficult and error prone. We propose to design a tool to automatically create and export cases OWL2 to as well. With these OWL2-based cases, we can do the evaluation effective.

The introduction is structured as the follows. First, the research approach is presented containing description of the research questions and the solution approaches are listed. Secondly, we offer a methodology of our design and the process of achieving objectives. In addition, we describe our scope of project, and contributions made in this thesis.

1.1 Problem Statement

As mentioned above, there are a number of problems with the EMF based meta-model of OperettA. (Okouya & Dignum, 2008) introduces the main problems as follows:
The OperA meta-model lacks semantics. In a lot of situations with case, there is a high potential occurs errors in the editing process. It will be hard or complex to check the consistency of a case

model by the designers. All the inconsistencies checking have to be done manually. Artificial lookup will not work effectively if errors occur, which means that it can be hard to modify or upgrade an OperA case model. This problem can be divided into 2 sub-challenges which will be explained clearly in the following. Some kinds of error are not detectable in OperettA and editing. Our solution will try to solve this.

1.2 Challenges & Research Questions

When we design an organizational model through the OperA meta-model without OperettA, there is a high potential occurs errors in the editing process and it will be difficult for a designer to check the consistency or find the errors they have made while editing the documents. The model is based on XML hard to find out possible errors that may occurred in the designing phase. This challenge leads us to the first research question described as below.

1. Research question 1

How can the designers check the consistency of the case model easily instead of manually checking?

- Approach

We will use the Protégé to check the inconsistencies of the model and try to find all errors made in the designing. If some errors happen in the designing of a case model, it will be difficult to check the errors. As (Horridge, 2009) and (Walter, Parreiras, & Staab, 2012) pointed out, an important part of OWL2 is semantic features expression and stronger constraints. Constraint definition can describe the conceptual description better because it contains semantics it and more powerful expressions. Reasoner (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007) has been developed by using of the semantic expression and stronger constraints to realize automatically judgment and derivation. With the reasoner, the process of designing a case model in OperA will be highly improved by automatically checking the consistency of the ontology.

We use a case study to prove that OWL2 can help checking the inconsistency of a model. For this we use the conference scenario, detailed in Chapter6, which was already modeled in current version of OperettA. We will design OperA case model using the same case description. Finally, the outcome should show that whether it can automatically check inconsistency in the case model which is guided by the OWL OperA meta-model through reasoner.

For a model, if some semantic errors happens, that not mean the model is incorrect. As (Aldewereld & Dignum, 2011) has mentioned that, OperettA is able to be used to create OM which is based on OperA meta-model. So, we are capable of using OperettA to design OM. However, for the OperettA, it uses the EMF for the development, which means that semantic errors are not able to be checked. Semantic error defines that in some kind of situation, the description of elements in model is not suitable for semantic meaning of the syntax or concepts. Without Unique Name Assumption (UNA) is a kind of semantic consistency for OperettA, which means that when we entry the name for different individuals in OM, it is not possible for us to

use the name to identify the individuals (same name can be assigned to different individuals). However, as empirical fact, it's better for a model to make individuals different with different name which can result in without vague. If we can find out the individuals in the OM with the same name, we are able to locate these individuals and have the possibility to make them different and easy to be identified visually. There is another one semantic consistency may happen while designing the OperA OM. It is unacceptable that we when design an OperA case model without OperettA, we make an objective has sub objective of itself. However, this kind of situation is possible in syntax of OperettA. These introduction leads to following research question 2.

2. Research question2

How can we check the semantics consistency of a model exported by the tool developed by us?

- Approach

By the introduction of (Prud'Hommeaux & Seaborne, 2008), SPARQL query provide query statement for designers to query specific information from a RDF based format document. As (Klyne & Carroll, 2006) has described that, RDF is an expressive language and basis of OWL2. So, if we try to query information of OWL2, we can use the SPAQRL query. As (Sirin & Parsia, 2007) has presented that, because of the large vocabulary of OWL2 compare to RDF, we should do some modification for the SPARQL query to query DL syntax. With the help of this query, we can access the ontology and query the specific information in the ontology. Therefore, this approach by using SPARQL query can provide us method to access data in the ontology and query all specific information out.

We are able to use case study to prove the validation of the approach. The conference scenario mentioned above is used again. For evaluate semantic inconsistency example 1, firstly, we add some same names for different individuals in the conference case description. Then, we try to design this OM case by OperettA. In the following, we use the tool we propose to develop to export this case into OWL2 file which follows the OWL OperA meta-model. The tool will be detailed explained in Chapter3. After we get the OWL2 file which represents the conference case with some same names for different individuals in OM, we input the file into Protégé, and using the (Rodriguez-Muro, Lubyte, & Calvanese, 2008) which will be detailed described in Chapter2 to query the individuals with the same name. Results should show that different individuals of same name will be queried and located by SPARQL query. With regards to semantic inconsistency example 1, we add an objective has its sub-objective attribute of its own in the OWL OperA case model. Then, we use the reasoner to export inferred ontology. On the top of this ontology, combining with SPARQL query, we can get this objective which has sub-objective of itself. The detailed evaluation will be explained in Chapter 6.

As the information introduced in the above paragraph, we need to get a conference case in OWL2 for the SPARQL query. But the OWL2 is complex and hard to edit the case in the OperA OWL meta-model format by txt, so there is a demand to support creation of OWL OperA OM conveniently.

3. Research question3

How can we automatically get a case model in the format of the OperA OWL meta-model?

- Approach

We have designed a tool to convert a case model with OperA model into an OperA OWL representation (in accordance with the OWL OperA meta-model). After using this tool, we get an OWL model of organizational model which can be used by protégé. With the help of this tool, we can do the transformation easier.

With the mapping from the OperA meta-model to the OWL OperA meta-model, we will develop the tool based on OperettA to take full use of the functions in OperettA resource. Combined with Jena (McBride, 2001), we can implement this tool.

On the basic of these solutions to these challenges, there should be two expected detailed functionalities added to the model:

1. It has the ability to do the inconsistency checking of the ontology automatically.
2. It is able to check the semantic inconsistencies of the ontology automatically.

In addition, there should be an added functionality for the OperettA tool:

3. We can have a tool to convert an OperA case model into an OWL OperA case model. OperA case model means the SS of a case OM built on OperA meta-model. OWL OperA case model represents the SS of a case OM built on OWL OperA case model

These added functionalities will be detailed described and evaluated in Chapter 5 and Chapter 6. If we prove that these properties are valid, we can answer our research by these added functionalities. In order to answer these research questions, we use the methodology introduced below as the guidance of our processes.

1.3 Methodology

This project follows the design methodology explained below:

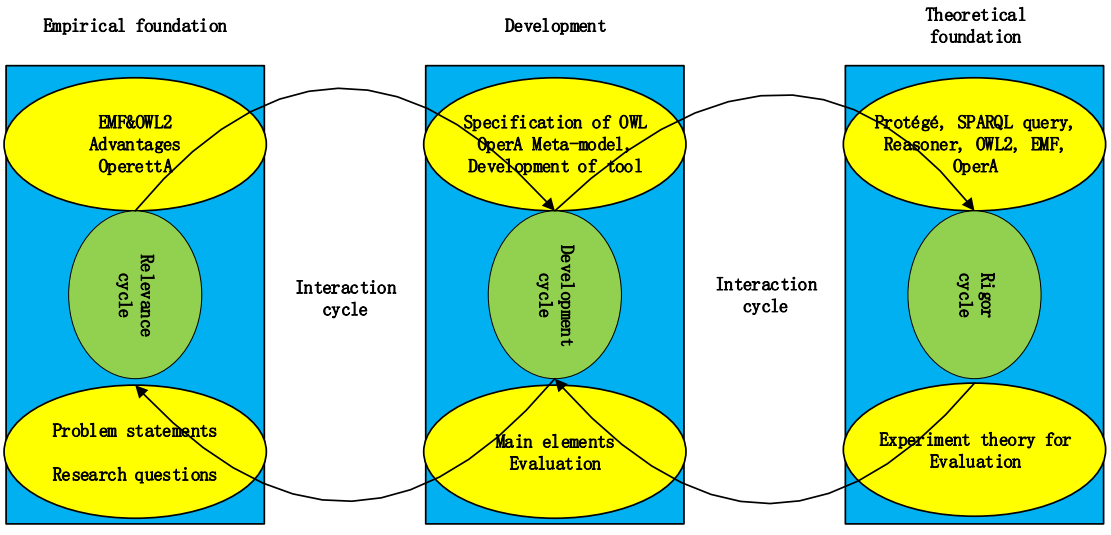


Figure1.2 Design Methodology

In this methodology, there are three core parts to the project: the relevant cycle, the development cycle and the rigor cycle. These cycles will interaction with each other development. Eventually, we can get the outcome through the develop cycle:

- ✓ Relevance cycle: Empirical foundation

In this cycle, we analysis the OperA meta-model, and OperettA, and form the challenges and research questions. According to the experience of OperettA which is based on the OperA meta-model, we discover the challenges that the existing for OperettA. The process can be called Empirical foundation. This part provides the foundations for the development, and compares the results of the evaluation the development to confirm our contributions.

- ✓ Design cycle: Development

In this cycle, the development intends to design the meta-model of OperA in OWL and the tool for automatically exporting OWL OperA case model. The design of the case study and evaluation of this case such that we can judge whether the OWL OperA meta-model is correct and can solve the research questions. This part gets directions from the empirical foundation and the theoretical basis to make the development feasible.

- ✓ Rigor cycle: Theoretical foundation

In this cycle, we search for the related work and knowledge which are need for this project which are called background theories. We should find our theoretical ways or methods to do the experiments which are used to support the evaluation. In this part, we get the requirements from the development part and find the theoretical foundation for the development.

1.4 Scope

As OperA is a complicated framework consists of several parts: The Social structure (SS), the Interaction Structure (IS), the Communication Structure (CS), and the Norm Structure (NS) are the core sub-frameworks of OperA. We scope our research in this project into a particular part of these described above. In OperA, each part of them is functionally independent module and elements in them are described independent. In order to convert all the OperA, we should do them separately, which will improve the efficiency. So, the scope of our thesis is not limiting our results.

Therefore, an OWL-Based ontology for only OperA meta-model, named the called Social Structure (SS) will be designed and evaluated. We call this the ontology OWL OperA meta-model.

1.5 Contributions

In this thesis, we made the following contributions:

1. We produce an OWL OperA meta-model by converting the SS of the OperA meta-model into OWL2.
2. We develop a plug-in of OperettA for exporting the SS of an OM designed by OperettA into

OWL2. Through this plug-in, we are able to get models on OWL OperA meta-model.

3. We do the evaluation of OWL OperA meta-model through case study. After the evaluation, we successfully prove that OWL OperA meta-model produce more advantages comparing to OperA meta-model.

1.6 Design Guideline

In order to convert OperA meta-model into OWL2, some more research steps need to be taken as well. The following activities will be taken to do the development:

1. Study the OperA meta-model (Social structure) and EMF modeling, use of protégé, OWL2 language;
2. Analysis and learn the Social structure of OperA, the do specification of SS for converting it into OWL2. Elements, relationship and activities analyzing of SS.
3. Design correct and complete mapping OWL OperA meta-model of Social structure in OWL2 by protégé;
4. Develop a simple tool as plug-in of OperettA based on the OWL OperA meta-model for creating case model.
5. Example collection and learning of the OperA Social structure and do the case study for the evaluation.

The thesis is organized as follows: Chapter2 presents the related works which we may use or discuss in our project. Chapter3 discusses about OWL OperA meta-model and specify features of this model. Chapter4 provides the specification of the tool OWLperettA, this tool is a plug-in of OperettA and is able to be used for exporting OperA model into OWL2 model. With the description of Chapter5, we can get to know the added functionalities which will occur after our meta-model conversion. Chapter6 show the evaluation of these added functionalities mentioned in previous Chapter. Finally, we summarize what we have contributed and give the conclusion for our thesis in Chapter7.

2. Background Information

Before we design the OWL OperA meta-model or develop the tool, we give some necessary theoretical foundations on the relevant topics used in this thesis. According to the science cycle of methodology described in Chapter 1, the OperA framework, the OperettA tool, OWL2 model, Protégé etc are the necessary theoretical foundations we are going to use this information run in the remainder of the thesis.

2.1 OperA

The OperA framework is an organizational modeling approach (M. Dignum, 2003). OperA combines the specification of organizational structures, requirements and objectives allowing participants to act freely on their own capabilities and demands. At an abstract level, the OperA model describes the objectives of the organization which are used to describe the desired states that the organizations want to be in.

OperA contains an organizational model (OM), social model (SM), and interaction model (IM). The OM specifies the means to realize the objectives of the organization. The OM is divided into of 4 structures as follow: the social structure (SS), the interaction structure (IS), the normative structure (NS) and the communication structure (CS). The social model identifies organizational roles to agents and describes contracts about role enactment. The interaction model specifies the interaction agreements between role-enacting agents and the organization. In our project, we focus on the OM.

The organizational model in OperA, as shown in Figure 2.1 below, it is a part of OperA meta-model. The four structures (SS, IS, NS, CS shown below) are the main elements. Roles and interactions that determined by the global goals, are described in the social. Interactions between the roles, determined by the global goals, are described in interaction structure. Organization norms describing the expected or desired behaviors, are defined in the normative structure. The ontology and communication languages are in the communication structure. In our project, we keep our scope in SS of OperA framework. Because we have limited time to do this project and the OperA is a large framework that cannot be converted completely in a short research time. In OperA, each part of them is functionally independent module and elements in them are described independent. In order to convert all the OperA, we should do them separately, which will improve the efficiency. So, the scope of our thesis is not limiting our results.

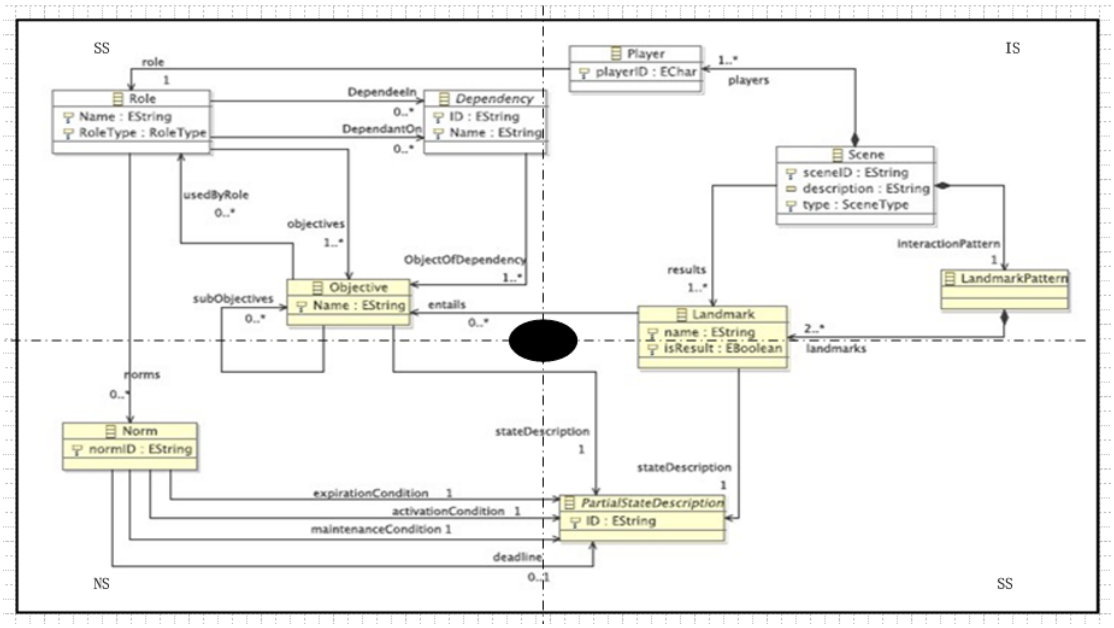


Figure 2.1 Part of OperA framework

There are several important elements that should be explained and described in the SS. These elements are Role, Objective, Dependency, and Rights. These elements will be described in chapter 3. The OperA is the guiding framework and methodology for creating OWL OperA meta-model we have mentioned in Chapter 1 and Chapter 4. It is also the basis for OWLperettA in Chapter 3.

2.2 OperettA

The OperettA tool was described in (Aldewereld & Dignum, 2011). OperettA is an IDE (Integrated Development Environment) developed to support the design, analysis and development of agent organizations using the OperA conceptual framework and methodology. It is intended to support software engineers and developers in both developing and documenting the various aspects of specifying and designing an organizational model.

The OperA conceptual framework is represented in EMF (EMF is explained later in Section 2.5) and resulted in the OperA meta-model. The OperA meta-model provide an expressive way for defining organizations and elements among the organizations. OperA meta-model supports the development of model driven software OperettA. OperettA enables the specification of organizational elements. OperettA also supports the specification of interactions among participants in the organizations.

OperettA is based on EMF, build using the OperA meta-model and containing visual editors in Eclipse framework. OperettA will be used during the evaluation as described in Chapter 5, and Chapter 6. In Chapter 3, we are going to introduce the OWLperettA plug-in embedded in OperettA used for designing OWL2-based OperA model.

2.3 OWL

2.3.1 OWL & OWL2 Language

OWL is a web ontology language described in (McGuinness & Van Harmelen, 2004). OWL is based on RDF. It can be used by applications to deal with the data in the ontology rather than just presenting them to humans. OWL is used to directly describe the meaning of terms in vocabularies and relationships among different terms. This representation is called an ontology. OWL has more facilities to express meaning and semantics than RDF as (McGuinness & Van Harmelen, 2004) described. OWL adds more vocabulary for describing properties and classes compared to the RDF. There are three significant sub-languages in OWL: OWL-lite, OWL-DL, OWL-Full. For the design of an ontology based on OWL, the three sub-languages can be selectively used for different situation and requirements.

According to (McGuinness & Van Harmelen, 2004), the three sub-languages can support different situations as following:
OWL-Lite : the simplest OWL sub-language that can be used for the specification of inheritance and constraints.
OWL-DL: compared to the OWL-Lite, OWL-DL is based on description logics which allows for automatic reasoning. Computational completeness and decidability are important features of OWL-DL.
OWL-Full: the most expressive language, but it lacks computational capabilities.

Overall, OWL-Lite has lower complexity than OWL-DL and OWL-Full. OWL-DL and OWL-Full both want to maximize expressiveness. However, OWL-DL supports computational completeness through description logics while OWL-Full supports free syntactic use without computational guarantees.

Basic OWL ontology consists of Individuals, Properties and Classes, and can be edited through the Protégé tool as mentioned in (Horridge, 2009). Individuals represent the objects in the scope. They are known as instance of classes. OWL does not use the Unique Name Assumption (UNA). In logics with the unique name assumption, different names always refer to different entities in the world. Properties are binary relations on individuals just as roles in description logics or attributes in some other formats. Properties in ontology consist of three sub-properties called: Object Property, Data type property and annotation property. These properties are all used to show the relationships existing in the ontology. Classes are explained as the sets of individuals. They are described using formal descriptions that exactly state the requirements for the membership of the classes. Combining Classes, Properties and Individuals, we get an ontology.

As (Consortium, 2009) presented, OWL2 is similar to the overall structure of OWL but with more functionalities. OWL2 is an extension of OWL. OWL2 ontology provides Classes, Properties, Individuals and Data values and is stored as a semantic web document. For OWL2, the following

some distinctions to OWL:

1. Syntactic sugar: some of the new features are syntactic sugar, which means that the added features to the syntax of the OWL to provide more features by OWL2 such as the disjoint union of classes.
2. New expressivity: new features like object properties, data types, data type properties have new expressivities to express elements in them such as keys, property chains, qualified cardinality restrictions and so on.
3. Three new defined profiles. OWL 2 Profiles are sub-languages (syntactic subsets) of OWL 2 that offer important advantages in particular application scenarios. OWL 2EL, OWL2 QL, and OWL2 RL are the three defined profiles. Each profile is defined as a *syntactic restriction* of the OWL 2 Structural Specification. As a result, the set of RDF Graphs that can be handled by the Description Logics reasoner is slightly larger in OWL 2.

According to the OperA, we know that there are all kinds of constraints for the interactions between different elements. In order to represent OperA framework, we should use the language that is capable of representing different specified constraints. OWL2 will provide more powerful syntax for expressing these kinds of constraints by syntax sugar, new expressivity or defined profiles. Therefore, we choose the OWL2 as the basic language for the completely converting. We will introduce the OWL2 ontology components for future designs.

2.3.2 Elements of General OWL2 Ontology

OWL2 ontology mainly consists of three parts: Individuals, classes and properties. We intent to introduce these three part as the basis of our specification.

Individuals

As (Horridge, 2009) has presented that Individuals represent objective in the domain which we are interested in. It should have a name and it must be explicitly stated that individuals are the same or different because of the unique name Assumption is not exist in OWL. Individuals are also known as instances of classes. Designing an OWL OperA case model is creating individuals upon the OWL OperA meta-model.

Properties

Properties are binary relations on individuals. Properties link two individuals together to show the relationship between the two individuals. Properties can be explicitly described further to make us know more about the relations. These relations can endow more semantic meaning and axioms for future inferring. Properties can also link the individuals to a data type. This is another kind of description for the individual. Therefore, in Protégé, the properties are divided into 3 parts: ObjectiveProperties, DataProperties and AnnotationProperties. In our project, ObjectiveProperties and DataProperties need to be detailed described by the constraints and axioms for different elements in OperA approach.

Classes

According to the (Horridge, 2009), classes are elaborated to sets that include individuals. They are described using formal description format that state precisely the requirements for memberships of classes. Classes may be organized into a hierarchy with super-class and sub-classes which is known as taxonomy. These classes represent a category of individuals who may share the same properties.

Taking our project into consideration, we choose OWL2 as the converting basis language as we has described in chapter section 2.3.1. We intend to create OWL2 ontology at the end. Instructed by this knowledge, we are going to specify the OWL OperA meta-model in Chapter 3.

2.4 Protégé

As (Horridge, 2009) mentioned, Protégé is a free, open source ontology editor and knowledge-base framework. The Protégé platform supports modeling ontologies via a web client or a desktop client. Protégé ontologies can be developed in a variety of formats including OWL, RDF(S), and XML Schema. Protégé is based on Java, is extensible, and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development.

An important difference between Protégé's OWL and OWL standard is that OWL standard does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual. For example, "Queen Elizabeth", "The Queen" and "Elizabeth Windsor" might all refer to the same individual. In OWL, it must be explicitly stated that individuals are either the same, or different from each other — otherwise it will be unclear whether they might be the same, or whether they might be different from each other.

If we use Protégé to design OWL2 ontology, three main elements should be paid attention to: Individuals, Classes and Properties as we mentioned in Section 2.3.2. For designing the definitions and constraints of the model, we should define the relationship between individuals and individuals, or individuals and datatype. With the help of ObjectPropertiesRestriction and DataPropertiesRestriction, we can give the semantic meanings to the model to express the relationships.

In addition, As (Tudorache, Noy, Tu, & Musen, 2008) has described that, there are a lot of plug-ins for Protégé. These plug-in make Protégé more powerful by providing different functions. In our project, the reasoner and SPARQ queries will be used in Chapter 6 for the evaluation. These two are going to be introduced in following paragraph. Protégé is the tool for us to implement the OWL OperA meta-model as we specified in Chapter 4.

2.5 EMF

Eclipse Modeling Framework (EMF) as mentioned in (Budinsky, 2004) is a framework to describe

a model, and then generate software related to that model. Especially, it is integrated with and tuned for efficient programming. It brings the high level modeling and low level programming together. EMF is based on Eclipse, the described model concepts can be related to the implementation in Eclipse through EMF.

Models in EMF gave a common terminology in describing the model, the meta-model (Völter, Stahl, Bettin, Haase, & Helsen, 2013). A meta-model is the basic structure of a complete model. In brief, a model is an instance of the meta-model, the meta-model is a model that makes statements about modeling. The meta-model can be used to deal with the challenges of model language syntax description and model validation through constraint defined in the meta-model. The main elements provided by EMF to build a meta model is described in Figure2.2.

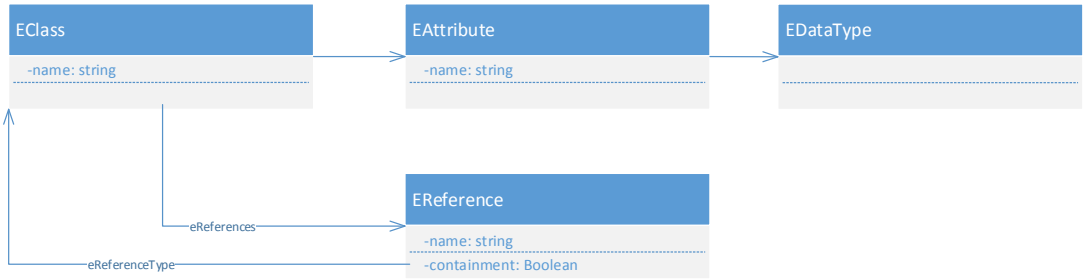


Figure2.2 structure of EMF modeling

The meta-model build through EMF composes of Eclass, EAttribute, EReference , and EDataType. With these elements, they can describe interactions and relationships of a meta-model. After building the meta-model, one can develop software depending on this meta-model. The Figure 2.3 shows the process of creating a model driven software.

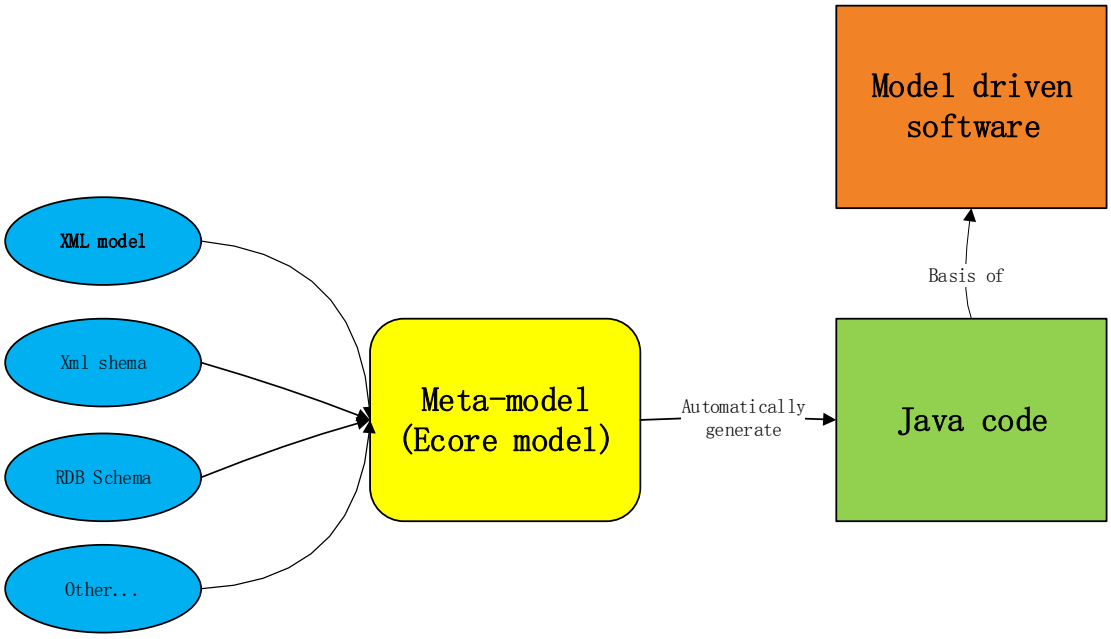


Figure 2.3 model-driven software process

As shown in Figure2.3, the Ecore model or meta-model can be created from an UML model, an XML Schema or a RDB Schema. On the basis of the Ecore model, we are able to develop the related software. The biggest advantage of using EMF is the automatic code generating

functionality. Java implementation code or other forms of models can be automatically generated to develop related software easier.

The description above makes us understand more about the EMF. This Section supports the EMF advantages described in Chapter 1. In our project, the OperA meta-model was built on EMF. So, when we design the OWL OperA meta-model in Chapter 4, we also need to take the OperA meta-model into consideration.

2.6 Reasoner

As (Sirin et al., 2007) has mentioned, the reasoner is a key element for working with OWL. The reasoner is a kind of software or reasoning engine that enables to infer logical consequences from a set of asserted facts or axioms. A reasoner will provide many standard and extended reasoning services for semantic ontologies. Reasoners can be used to check the consistency of the classes, object properties, and data properties in the OWL ontology. There are several important reasoners that have been developed for supporting ontology reasoning such like Pellet, Hermit, or FacT+.

HermiT (Shearer, Motik, & Horrocks, 2008) is the current used reasoner by Protégé mentioned in Section 2.4. Given an OWL model or ontology, HermiT can identify the subsumption relationships between classes so that it does reclassification. It can also check the consistency of the inputted ontology. After using the reasoner, the inconsistencies are shown in Protégé in red and in an explanation window. With the help of the Hermit reasoner, we can get an result for the consistency and classification of the classes, object properties and data properties, and we can judge the efficiency of our OperA OWL meta-model through a comparison between the expected result as we mentioned in above paragraph with the actually results.

In Chapter 1, we have mentioned the reasoner for evaluation. In the Chapter 5, we treat the reasoner as a candidate for solving one of our research questions. Detailed information about the usage of reasoner will be described in chapter 6.

2.7 DL-query Tab

As Nickdrummond[1] has introduced that the DL-query tab is a plug-in for Protégé. It can provide a method to query the OWL ontology consisting of DLs in protégé. DL is a sub-language of OWL2, which we have mentioned in Section 2.3. However, it has a limitation that only classified ontologies can be queried through it. A classified ontology is a correct ontology that is checked for consistency using the reasoner. Manchester OWL syntax (Horridge et al., 2006), which is a friendly and easy syntax, used as the basic for the query language. OWL DL tab use the class expression to query the relevant information of ontology. Class expression is based on the

[1] DL-query tab, retrieve from: http://protegewiki.stanford.edu/wiki/DLQueryTab, 2014.

Manchester OWL. The DL-Query can also be used to add elements to an ontology just like the function of Protégé.

The DL query consists of three parts: Query (class expression), Query result and selectable components. For the Query result, the output of the query will be shown in a query result window. The output will be different by choosing the selectable components which we desire to get. Six components can be selected for querying: super classes, ancestor classes, equivalent classes, subclasses, descendant classes and individuals. These components are all the result categories which we can get from the DL-query tab.

DL-query is treated as a candidate solution for doing the added functionality evaluation in Chapter 5. Information mentioned in this Section will help analyzing and deciding whether the DL-query tab can be used in Chapter 5 or not.

2.8 SPARQL-query

In Section 2.4, it is said that, SPARQL query tab (Rodriguez-Muro et al., 2008) is a plug-in in Protégé. The SPARQL query tab is used for querying ontology information and access data in an ontology by using the SPARQL based query language. This language will be described in the following. We will use it for our evaluation. However, in order to take use of this plug-in, SPARQL query language should be used. That means, we should have ability of using SPAQL query languages.

As (Prud'Hommeaux & Seaborne, 2008) mentioned, SPARQL query is query language for RDF. It can be used to express queries on source data which are stored in RDF. The SPARQL query has its own syntax and semantic expression for querying. The results of the SPARQL will be sets or RDF graphs.

For basic SPARQL queries, Basic Graph Pattern (BGP) is the building block. A BGP consist of a set of triple patterns. A triple pattern is an RDF triple. For complex SPARQL queries, Combinations of BGPs are made by using the following projections: SELECT, OPTIONAL, UNION and FILTER.

After the brief introduction of SPARQL, we should notice that in this thesis, we may focus on querying the OWL-DL by SPARQL for the detailed information. As (Sirin & Parsia, 2007) has presented that, even though there are many query languages for querying RDF and OWL, neither type can be used to query the OWL-DL, so SPARQL-DL was developed. SPARQL-DL is a powerful and expressive language for querying OWL-DL by combining TBox, RBox and ABox. As (Fokoue, Kershenbaum, Ma, Schonberg, & Srinivas, 2006) has described, the TBox contains assertions about concepts such as subsumption and equivalence. The RBox contains assertions about roles and role hierarchies. The ABox contains role assertions between individuals and membership assertions. The SPARQL-DL has a more expressive and powerful vocabulary compared to DL query. Mixed Boxes query in syntax is an extension for normal ABox or Tbox for SPARQL-DL. The semantics of the SPARQL-DL is similar to OWL-DL. This information is the theoretical foundation

for the Terp which is described below.

According to the requirements of SPAQRL queries in Protégé, we need to be capable of querying OWL with SPARQL queries. The query range will be OWL2 ontology in Protégé rather than RDF. We have demand to design a particular kind of SPARQL query statement for querying OWL2. As (Sirin, Bulka, & Smith, 2010) has described, Terp has been developed to design this particular kind of SPARQL query statement. Terp is an extension for SPARQL query from RDF to OWL2. Terp syntax allows class, property, and data range expressions, expressed in Manchester syntax, to be used inside SPARQL queries. With this information, we will gain ability to write queries to query OWL2 ontology in our evaluation in Chapter 6.

In this Chapter, we have introduced the related information that should be known and will be used for the following design and implementation. OperA, OperettA, and EMF are the basic elements for this project. They are the theoretical foundations for the specification of the OWL OperA meta-model in Chapter 3 and the specification of OWLperettA that will be developed in Chapter 4. OWL2 and Protégé should be known and understood for the design and implementation of the OWL OperA meta-model in Chapter 3. Reasoners, DL-queries or SPARQL queries may be applied and used for the evaluation which will be done in Chapter6. In the next Chapter, we will design our tool for simplify the evaluation by using the related knowledge introduced in this Chapter.

3. Specification of the OWL OperA Meta-model

We have introduced the relevant information about the entire project, such as OperA, OperettA, and so on, in the previous Chapter. The OWL OperA meta-model is our desired product because of the problems that occur in the OperA meta-model, as we have described in Chapter1. We also intend to develop a tool based on the OWL OperA meta-model. Therefore, we should specify the OWL OperA meta-model by considering the OperA meta-model as the basis for conversion. With supporting background information supporting, we can find appropriate ways to design the OWL OperA meta-model.

In this Chapter, we aim at providing a specification of the elements into OWL format after analyzing the OperA meta-model as presented by (Aldewereld & Dignum, 2011). We call it the OWL OperA meta-model as another representation of the OperA approach; it is also the basis for developing the tool. In order to design the OWL OperA meta-model, the mapping methodology and structure of the OperA meta-model should first be illustrated clearly.

3.1 Mapping Methodology

As we have described, we are going to map the OperA meta-model into OWL2. The OperA meta-model is based on EMF. Therefore, we must find a method for doing it. Table 3.1 below outlines the methodology of conversion.

Domain in EMF	Ranges in OWL2
Eclass	Class
EAttribute	Data Property
EAttributeType	Data Type
Etype	Date Type
EReference	Object Property
Etype	Class
EEnum	Class
EDataType	Data Type
EOperation	-
EAnnotation	-
EAnnotation	Annotation property
EAnnotation Details	-
Key	constant
value	property value
EAnnotation Reference	-
	Entity URI (Class)
EEnum	Class

Name	Class
Default value	Data property assertion of individual

Table 3.1 EMF to OWL2 Mapping

Table 3.1 has shown the mapping methodology for transforming the core elements of the EMF model into OWL2 elements. By using this mapping table, we are able to perform the conversion in the following sections.

3.2 OperA Meta-model Description

In order to conduct the mapping from the OperA meta-model to the OWL OperA meta-model with a complete transfer, we should first understand clearly the structure of the meta-model. Relationships in SS and relationships between SS and OM are the core aspects we are going to explore. First, we introduce the relationships in SS.

3.2.1 Architecture of SS

According to (Aldewereld & Dignum, 2011) , the social structure of the OperA meta-model presents the definition of organizational roles, with their objectives, rights, norms and the relationship among them. The SS architecture contains several elements and the relationships among them are shown below in Figure3.1:

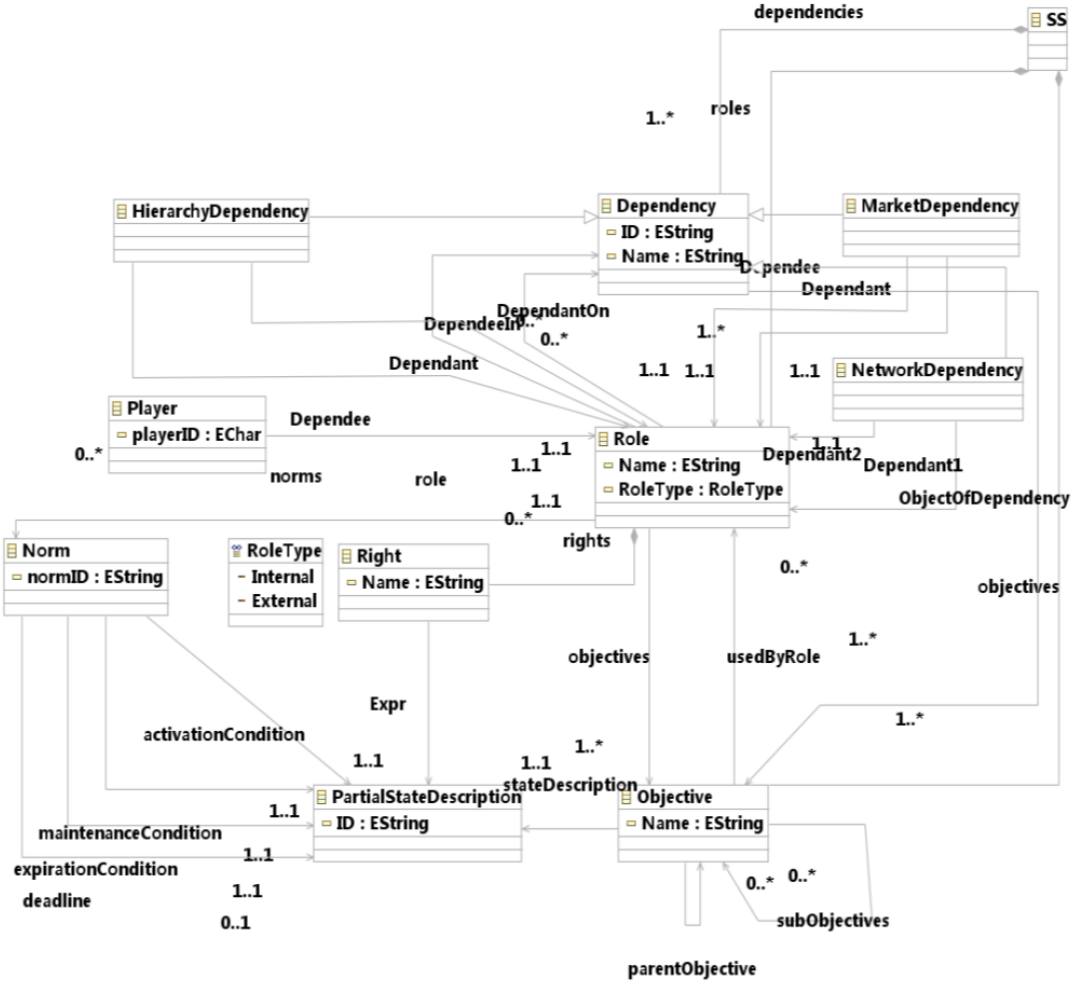


Figure3.1 Social Structure

According to Figure3.1, SS presents the different interactions among Role, Dependency, Objective, etc. Dependency also has three sub classes. Various elements interact with others through different interactions. Role, Objective and Dependency are the elements directly forming the SS. However, OperA is a model consists of SS, IS, CS, and NS. Without the interaction with other structures in OperA, SS will not achieve the functionality of OperA. We will describe the interaction between the SS and the other structures in the OperA meta-model in the following section.

3.2.2 Architecture of SS in OM

We have introduced the SS in the previous section. We will now present the relationships between SS and the OM to explain the interaction among the SS and the other structures in OperA. According to (Aldewereld & Dignum, 2011), SS is a part of the OM and it will interact with the other parts to achieve the model features. A brief high level relationship of SS in the OM is shown in the Figure3.2 below:

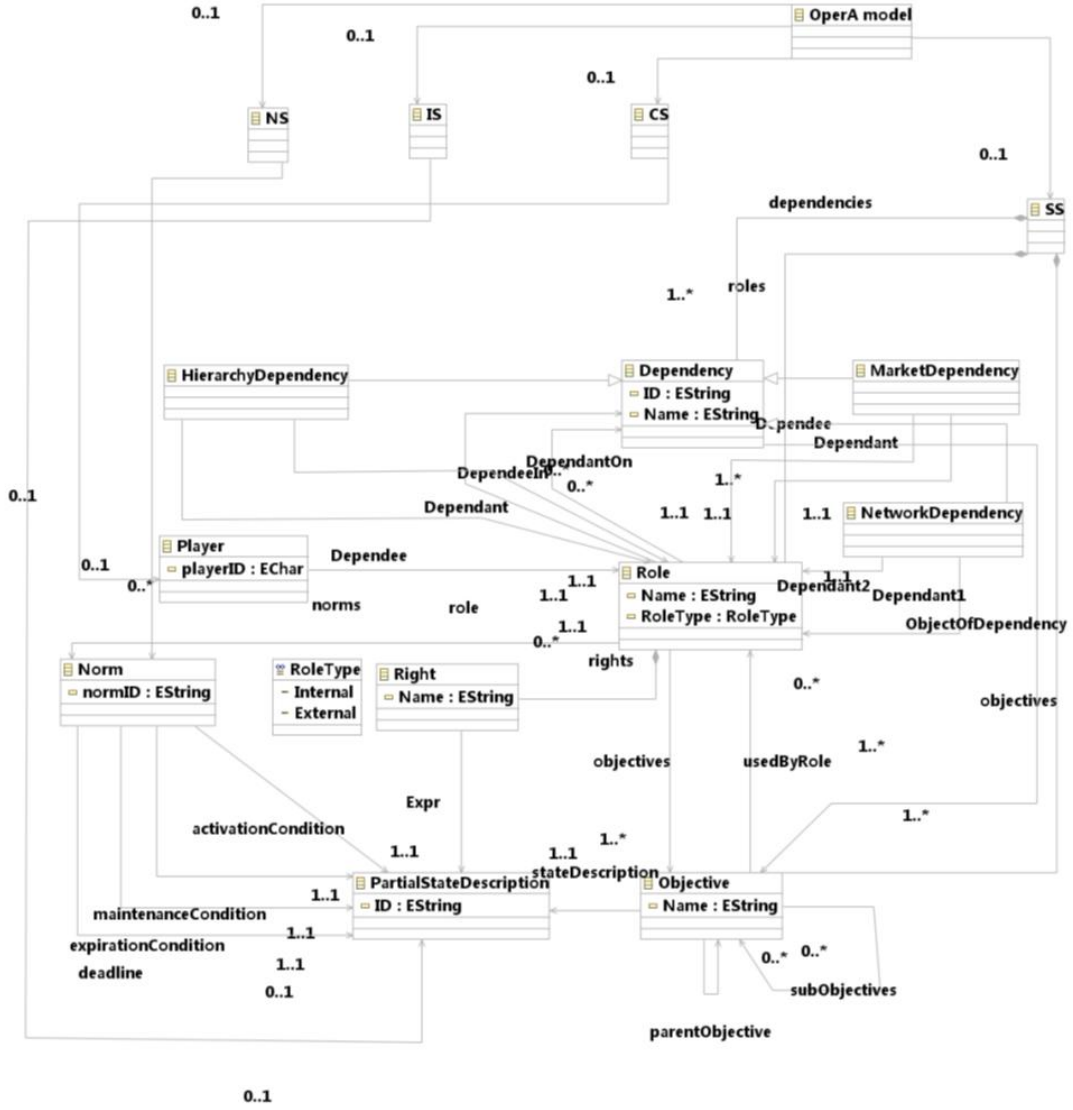


Figure3.2 Relationship of SS in OperA

As Figure 3.2 reveals, SS is a part of the OM. The SS in Figure 3.2 is a representation of Figure 3.1. There are several elements related to role; but not all the related elements are elements of SS. We do not have to define the elements that are out of SS, such as PartialStateDescription in the OperA meta-model, because of our scope. The related elements will be called interacting elements. Role, Objective, Dependency and Rights are the most important interacting classes in SS and elements such as Norm, player and partial state description should be in other structures, albeit related to SS. The SS interact with the other parts of OM through the interaction among Roles in SS and Player, Norms, and PartialStateDescription out of SS. Above all, elements in SS will also interact with each other. All relationships are interactions among instances of different elements. Therefore, OperA can work effectively with complete model features.

Based on these two figures, we can clearly see the elements, different relationships, and interactions in SS and out of SS. We will now abstract the information contained in the SS of the OperA meta-model in the following section.

3.3 OperA Meta-model Definition

Before we specify the OWL OperA meta-model, we should clearly understand the content of the OperA meta-model. Detailed information about the OperA meta-model will be shown in this section. According to (M. Dignum, 2003), the OperA approach is defined as follows.(we provide just give the Role element example of the OperA approach; the others are elaborated in AppendixA.1.) These definitions are provided as the basic background for the existing OperA meta-model defined in (Aldewereld & Dignum, 2011).

Role: It is the presentation of participants that holds in the organization. Roles describe an organizationally-sanctioned, structured bundle of activity types. In the SS, role representations specify the activities and services necessary to achieve social objectives. Role representations also enable abstraction from the individuals that will finally perform the role. We provide the definition of Role elements that should existed in SS in Table3.2.

Role Definition	
Name:	A unique name referring to this role
Objective:	A set of landmarks that describe the desired result of this role
Rights:	A set of expressions about the rights of the role
Norms:	A list of normative expressions that limit this role
RoleType:	Which type of agent can apply for or perform this role: external or Internal

Table 3.2 Role definition

According to Table 3.2, we can see that Role has some attributes. These attributes show the relationship between instances of the role and other elements. As we have described, these definitions are the basis of the existing OperA meta-model. The role elements of the OperA meta-model are defined in Table 3.3.

Element	EAttributes	→ interacting elements	EReference
Role	objectives	Objectives	Name
	norms	Norms	
	rights	Rights	
	roletype	Roletype	
	dependantOn	Dependency	
	DependeeIn	Dependency	

Table 3.3 OperA meta-model definition for Role

We abstract Table3.3 from the existing OperA meta-model built by EMF. This Table shows that EAttribute reflects the relationships between role instances and interacting elements' instances. Interacting elements are the destination ranges of these relationships. EReference shows the relationship between role instances and data type. Among the interacting elements, some of them belong to SS and some do not.

The above information has introduced the Role definition in the OperA approach and the Role specification in the OperA meta-model. There are some additional elements that have been

defined and specified. Due to limited space, we put them in the AppendixA.1. We have already obtained all the specified OperA meta-model elements in SS. Among all the elements, some interacting elements should also be mentioned so that we can read the OperA format documents successfully through the OperA document. However, as we describe in Section 3.1.1, we don't need to specify these interacting elements in detail because they do not belong to SS. These interacting elements are shown below:

Norms: They describe the rules and limitations that some of the classes (like Role) must follow.
PartialStateDescription: It is a type of description of status for different states needed for different classes (like the Objective in Social Structure) in the OM structure.
Player: It is a description for higher level of Role; a player can play different roles in different situations and requirements.

All of the above information represents the definition of the elements of SS. We need to specify each element in Table 3.3 above to the OWL2-based Format and implement through Protégé. Through the support of the OperA meta-model specification, the OWL OperA meta-model can be designed to represent the OperA approach in OWL2. We will now specify the OWL OperA meta-model.

3.4 Specification of SS

In this Section, we present specifications of the elements in SS in the OperA meta-model into OWL2. Then, we implement these specifications through Protégé. We have presented theoretical knowledge of the OWL2 ontology in Section 2.3. We should specify the related model into OWL2 ontology. According to the definition of the OperA meta-model, we can specify elements into OWL2 as follows by using the mapping methodology provided in section 3.1.

Class hierarchy
In order to show clear relationships among elements that are named classes in OWL2, we design the hierarchy of classes of SS in Figure3.3.

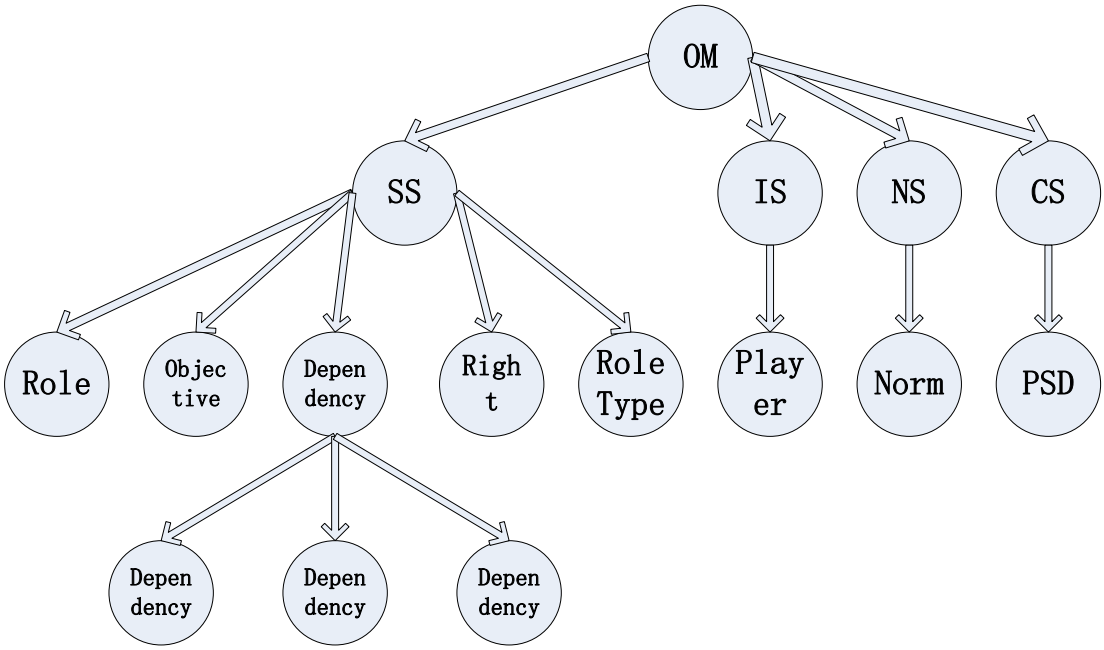


Figure3.3 Class hierarchy of SS in OWL

In the figure, we can see there are four hierarchies in the social structure. Our scope is SS and the sub-classes of SS. The reference classes are shown in the right of the figure as we mention in Section 3.2.1. In the following section, we define the SS for the OperA OWL meta-model mapping from the OperA meta-model. As the Figure shows, there are five classes and three sub-classes that must be defined.

Class definition

It is necessary for us to define the classes and the relationships between different classes in OWL ways. As we described above, all the classes in SS have been provided. We should define them in detail to achieve an entire meta-model.

As (Horridge, 2009) has described, in OWL modeling, we can specify different elements through different features. The confirmed features in the above paragraph has been introduced, we thus have some simple guidance for specification of the OperA OWL meta-model.

1. Elements should be specified into classes.
2. Relationships between classes should be specified into properties. There are two kinds of properties that we should describe: objectproperties and dataproperties.
   - ✓ Objectproperties shows the relationship between individuals or classes. They have their own features and objectproperties restrictions.
   - ✓ Dataproperties shows the relationship between individuals or classes and data. They have their own features and dataproperties restrictions.
3. Instances of a class should be specified into individuals. Individuals can be given value.

According to this guidance and mapping table in section 3.1, we can design our OWL OperA meta-model by designing the classes, object properties, data properties and data type for different elements that exist in the OperA meta-model. In the following section, we provide the

specification for each element. We provide a Role element example. Specification for the other elements is shown in AppendixA.2

Role

The role and its relationships with each other classes are shown in Figure3.4 below:

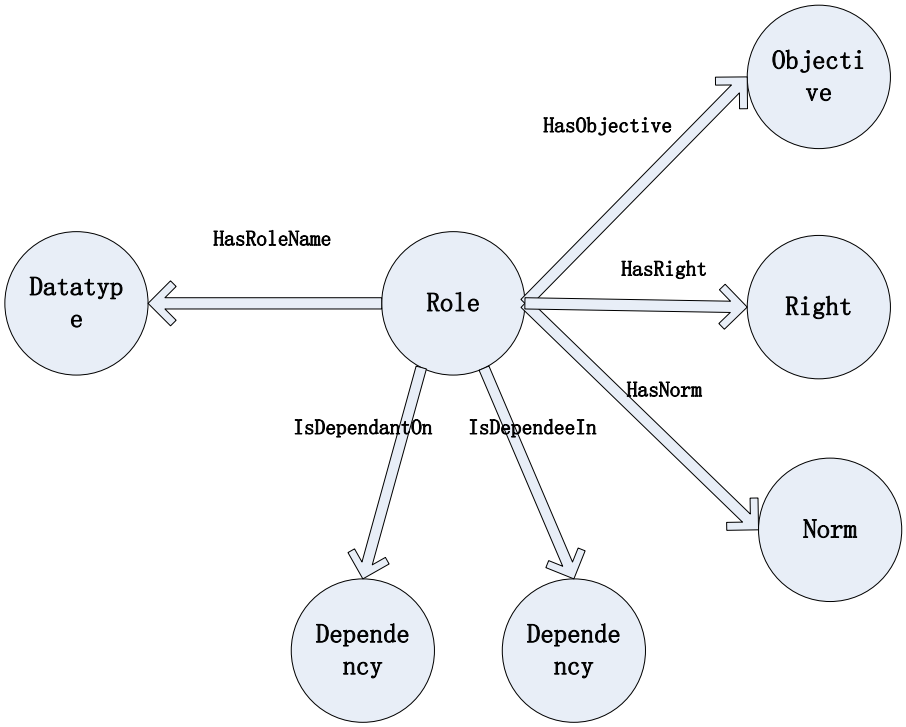


Figure3.4 role & relationship

As the Figure 3.4 shows, for the individuals of a role, the role interacts with a data type and four interacting classes, as in the interacting elements we described in Section 3.2.1. Different interactions represent different relationships. We also give them a unique name for each different interaction. In the interacting classes, Norm is a class out of SS. We call it the OutInteract class. We define the relationships by derive target and destination target. Derive target is the deriving class of a relationship, which will be designed as a domain in the OWL OperA meta-model. Destination target is the destination class of a relationship, which will be designed as a range in the OWL OperA meta-model. For example, in the relationship HasRight, the domain should be class Role and the range should be Right. Based on the description above, we define the class Role in the table below, according to our simple rules and mapping table:

Class	Role
DataProperties	HasRoleName
Dataproperties Restriction	HasRoleName exactly 1 string
Objectproperties	HasRight HasNorm HasObjective IsDependantOn IsDependeeIn
Objectproperties	HasRight some Right

Restriction	HasNorm min 0 Norm HasObjective some Objective (IsDependantOn min 1 Dependency) or (IsDependeeIn min 1 Dependency)

Table3.4 Role

According to Table3.4, we can see that: for the dataproperty, we specify the name of the role. Individuals performing a Role should have a role name. The HasRoleName is the relationship. The domain of this relationship is Role and the range is dataType. These specifications show that: the individuals performing a Role have role names that are dataType.

For Objectproperty, we specify the interacting elements of Role. For different interacting elements, we give them corresponding relationships. The domain of this relationship is Role and the range is the corresponding interacting elements. In particular, for the interaction between Role and Dependency, we design them following the requirements provided by the OperA framework and these logical features cannot be realized by the OperA meta-model. After the specification of these core parts of Role, we can specify other similar elements.

Due to limited space, other class' specifications for the OWL OperA meta-model are shown in AppendixA.2. With regard to the OutInteract classes, because of our focus on SS, it is not necessary to defined them in detail. We just need to briefly declare these classes (we don't have to specify them) to ensure the completeness of SS.

3.5 Conclusions

In this Chapter, a mapping methodology (See Table 3.1) was presented. We also introduced the OperA meta-model, and specified the OperA OWL meta-model explicitly by using this mapping methodology. With the detailed definition of the elements and relationships, we can implement the modeling easily through Protégé. The implemented results are the OperA OWL meta-model. As we mentioned in Chapter 1, the OWL OperA meta-model is the desired product we propose to present. The OperA OWL meta-model will support the development of a tool. However, we should verify the OWL OperA meta-model by using a case study to evaluate it. That means we should design an OWL OperA case model for evaluation. However, it is not easy for a designer to design an OWL2 based OperA case model by editing; therefore, we hope to develop a tool to automatically export the OperA case model into OWL2. Then, we just need to use the OperettA to design the OperA case model as the input of the tool. The tool will be described in the following chapter.

4. OWLperettA: OWL Auto Generator Tool

As we mentioned in Section 1.2, a tool may be used to export the OperA case model into an OWL OperA case model. This tool can also be used to answer research question 3. In order to do so, the tool should be supported by the OWL OperA meta-model. We have specified the OWL OperA meta-model in Chapter 3. Therefore, in this chapter, we introduce the OWLperettA tool. OWLperettA is supported by OWL OperA meta-model. In short, OWLperettA is a plug-in embedded in OperettA. Combining OperettA with OWLperettA, we can create an OWL OperA case model: its usage will be explained in Section 4.4. OperettA is based on the OperA meta-model we have created through EMF. However, OWLperettA should be based on the OperA OWL meta-model mentioned in Chapter 1 as one of our research goals.

With the help of OWLperettA, we can create the SS of an OM through OperettA and export OWL OperA case model as the result. Through Protégé, we should be able to evaluate the efficiency of the OWL OperA meta-model by inputting this result into Protégé. In order to explain OWLperettA, we introduce its motivation and aim.

4.1 Motivation & Aim of OWLperettA

Motivation

As we mentioned in Chapter1, we hope to solve research question 2 by using a tool to automatically form the OWL OperA case model, so that we can save time and reduce editing structure errors. This problem has been extended to research question 3. OWLperettA is the tool that will be developed to solve this problem.

Aim

As mentioned above, on the primary level, OWLperettA aims at solving research question 3. Specifically, OWLperettA is the desired tool for our project to automatically convert an OperA case model (OperA document) into an OWL2 file. We suggest that it be embedded into OperettA to make the project integrated. OWLperettA is developed to support the conversion of OperA document into OWL2 files for the SS of an OM using the OperA OWL meta-model. The OWL OperA meta-model has been described in detail in the previous chapter.

In the following, we will provide a description of OWLperettA. In order to understand OWLperettA, we should first introduce its environment.

4.2 High-level Architecture

We present a high-level architecture of OWLperettA that explains its environments. These processes can export an OWL OperA case model based on an OperA case model. This OWL OperA case model can be validated as shown in the Figure4.2 below:

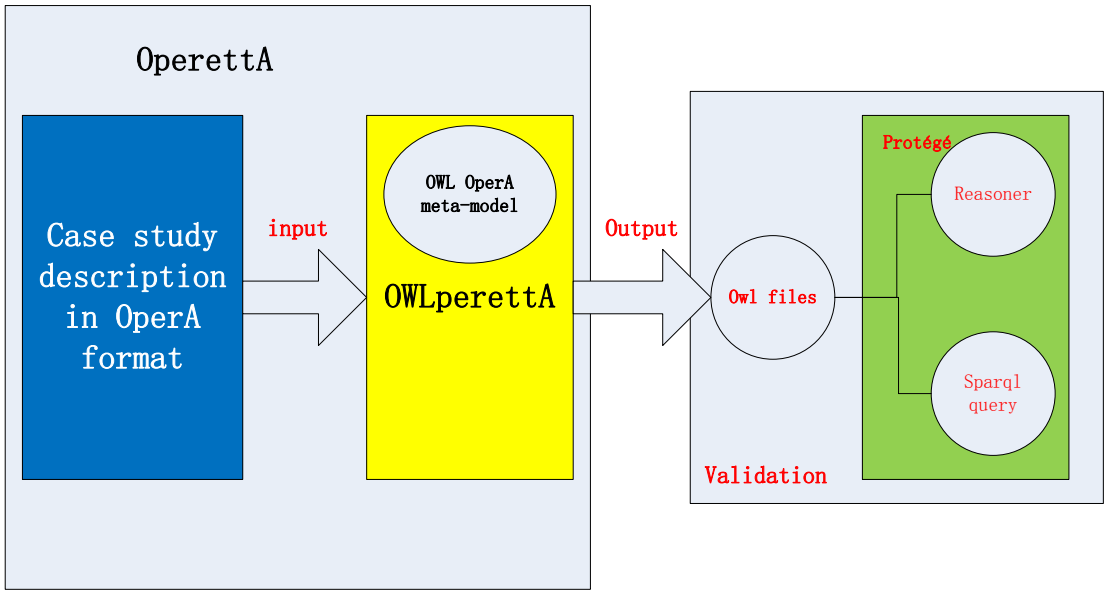


Figure4.1 High-level architecture of OWLperettA

According to Figure 4.1, this architecture consists of three parts: input, OWLperettA and output. It shows how OWLperettA will be used in our project.

- **Input**

In Figure 4.1, an OperA case model designed through OperettA is used as the input for OWLperettA. As mentioned above, OperettA is an existing tool used to design an OperA case model. Considering our goal to evaluate a case model with changes in Chapter 6, we must insert changes into the design of an OperA case model. This OperA case model is used as input for OWLperettA.

- **OWLperettA**

OWLperettA follows the mappings from the OperA meta-model to the OWL OperA meta-model as we have presented in Chapter 3. When an OperA case model is input into OWLperettA, we can output an OWL OperA case model, which can completely cover all the features of an OperA case model. The OWL OperA case model should be in OWL2 format and OWLperettA is embedded in OperettA.

- **Output**

The output portion aims at validating the efficiency of the OperA OWL meta-model by verifying the OWL OperA case model. By using OWLperettA, we can obtain an OWL OperA case model designed through OperettA. The OWL OperA case model is the case labeled by the OWL OperA meta-model and structured by OWL2. Then, we input the output documents into Protégé which we have mentioned in Chapter2. Reasoner, DL-query, and SPARQL-query are three kinds of plug-ins in Protégé that we may use to perform the evaluation. The evaluation procedure will be explained in Chapter 6.

4.3 Specification of OWLperettA

As demonstrated by the presentation of the high level architecture, OWLperettA can be regarded as a connecting component between an OperA case model and an OWL OperA case model. In order to develop OWLperettA to fulfill all the functionalities in the architecture, we must identify the requirements for such development.

4.3.1 Requirements for OWLperettA

In order to fulfill our goal of conversion, we must identify the requirements of OWLperettA clearly. The following list briefly outlines the requirements of this tool:

1. OperA-format input
2. Extracting Raw data
3. OperA meta-model to OWL OperA meta-model Conversion
4. OWL-format output
5. Integration
6. Visual Interface
7. Java based coding

The above list offers a brief impression of the requirements necessary for OWLperettA. We next present them more precisely in terms of what they represent, why they are proposed and how they offer solutions:

- **OperA-format input**

Explanation: OperA format should be the input format for the case document.
Reasons: Firstly, confirmed structured documents make the file easier to be read by a computer. Secondly, our project scope concentrates upon an OperA approach.
Solution: OperettA can be used to design the OperA case model as input for OWLperettA. Integrate OWLperettA with OperettA.

- **Extracting raw data**

Explanation: We must extract necessary and important information (data flow that represents the raw data information) from the OperA case model.
Reasons: When we have an OperA case model as input, we should be able to read the detailed and necessary information from this OperA case model. Extracting raw data provide a way to obtain the data flow.
Solution: An extractor can be used to extract correct raw data from the OperA case model. Many methods in OperettA package can be used as the theories basis of our extractor.

- **OperA case model to OWL OperA case model Conversion**

Explanation: There is a need to convert the raw data labeled by the OperA meta-model into raw data labeled by the OWL OperA meta-model.
Reasons: Because we must convert the OperA case model into an OWL OperA case model, we must perform the conversion by following the mapping from an OperA meta-model to the OWL OperA meta-model.
Solution: A convertor should be designed for the conversion. The converter should be based on the mapping from an OperA meta-model to the OWL OperA meta-model.

- **OWL2-format output**

Explanation: The output should be in OWL2 format.
Reasons: Firstly, we wish to get OWL2 models through the transfer. Secondly, we are trying to evaluate the efficiency of the OperA OWL meta-model; the output of OWLperettA should be able to be used by the reasoner or the SPARQL query tab in Protégé that we described in Chapter2.
Solution: A format generator for exporting and saving OWL files can be designed for fulfilling the requirements. As we mentioned in Chapter 1, Jena (McBride, 2001) can be used for solving this requirement. The OWL2 file export by Jena can be used in Protégé.

- **Integration**

Explanation: There is a demand for embedded OWLperettA into OperettA (Aldewereld & Dignum, 2011) as a plug in.
Reasons: Because we wish to make the tool an integrated tool, we intend to use the OperA case model as the input; the OperA case model can be designed by OperettA.
Solution: OperettA, which is the software used in Eclipse, is the embedded platform for OWLperettA, so we should use the plug-in dependencies of OperettA to embed the OWLperettA into the OperettA.

- **Visual Interface**

Explanation: A visual interface is also needed for exporting the designed document.
Reasons: It will save time and be easy for the user to adopt because of a humanized interface.
Solution: There is a visual button for us called ontology. After pressing this button, we can export our designed OperA case model. In order to fulfill this, we are able to use eclipse.UI.plug-in package because for OperettA, it is used in the same way for establishing a plug-in.

- **Programming language used:**

Explanation: We must use programming language for the programming of this tool.
Reasons: With the programming, we can design independent software that can easily to be connected to any other software with High compatibility.
Solution: Java is a good solution. Java is platform independent and further, OperettA is Java based. It is better to create a plug-in with the same developing language of the platform in which it will be embedded. There are multiple development environments for Java.

With the combination and implementation of the different requirements, we can obtain an integrated OWLperettA for our conversion goal. On the basis of these requirements for OWLperettA, we can develop the framework for OWLperettA below.

In the following section, we introduce the framework of OWLperettA for the detailed specification of this tool.

4.3.2 Framework of OWLperettA

Taking into account the information presented above, there are several processes that should be performed in OWLperettA, such as “read OperA case model”, “convert case model” and so on. Conversion can be implemented by dividing these processes into different components to make OWLperettA work normally and be easily understood. They can be called respectively raw data extractor, converter component and Format generator component. The structure of the OWLperettA will is shown in Figure 4.2 below:

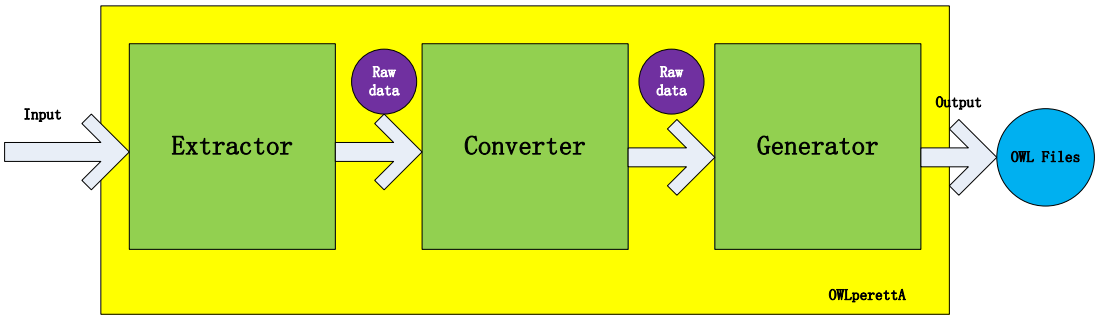


Figure4.2 Framework of OWLperettA

According to Figure 4.2, we can see that different components interact with one another: to form an OWL2 file. There is a data flow that represents the raw data of a case, such as the conference case mentioned in Chapter 1. The data flow from the extractor; to the converter; and then to the generator. In the different components, the raw data are handled in different formats. We describe these components as follows.

- **Extractor component**

In the extractor component, we aim to read the OperA files automatically. We have embedded OWLperettA into OperettA, so we can use the methods provided by the OperettA package to develop OWLperettA as described in Section4.3.1. This component aims at reading the inputted case study to obtain the raw data of the document. As described in Chapter1, we scope our project in the SS of an OM; thus, we should be focused on the provided methods of the SS in the OperettA package.

- **Converter component**

After the extraction of the inputted files, we come to the converter. In this component, we

transfer the extracted data (raw data obtained from the OperettA package) into the required raw data labeled by the OWL OperA meta-model, following the mapping we designed in Chapter 4. Hash Map, List should be used to store information for potential calling. We can simply visit all the related packages of OperettA and store the useful information into Hash Map or List for future converting.

- **Generator component**

After the converter, we should label these raw data with the OWL OperA meta-model and export them into OWL2 files. We should use Jena which we have mentioned in Section 4.3.1 and Chapter 1. Jena can provide methods for us to label raw data with the OWL OperA meta-model and export them into OWL2 (implementing the mapping). Finally, we can obtain the respective OWL2 files, called the OWL OperA case model. We hope that this case model can be input into Protégé.

However, as we have described, OWLperettA cannot be used independently to create an OWL OperA case model. In the following, we will introduce the usage of OWLperettA to create the OWL OperA case model for evaluation.

4.4 Usage of OWLperettA

Above, we have elaborated the development of OWLperettA. However, as we outline in Chapter 1, we must create an OWL OperA case model for the evaluation of research questions 1 and 2. We have mentioned that OWLperettA should be a plug in of OperettA. As we have described above, OperettA can be used to design an OperA case model, and OWLperettA can directly export an OperA case model into an OWL OperA case model. Thus, by using OperettA together with OWLperettA, we can conveniently create an OWL OperA case model. The usage of OWLperettA to design an OWL OperA case model is shown in the Figure 4.3 below.

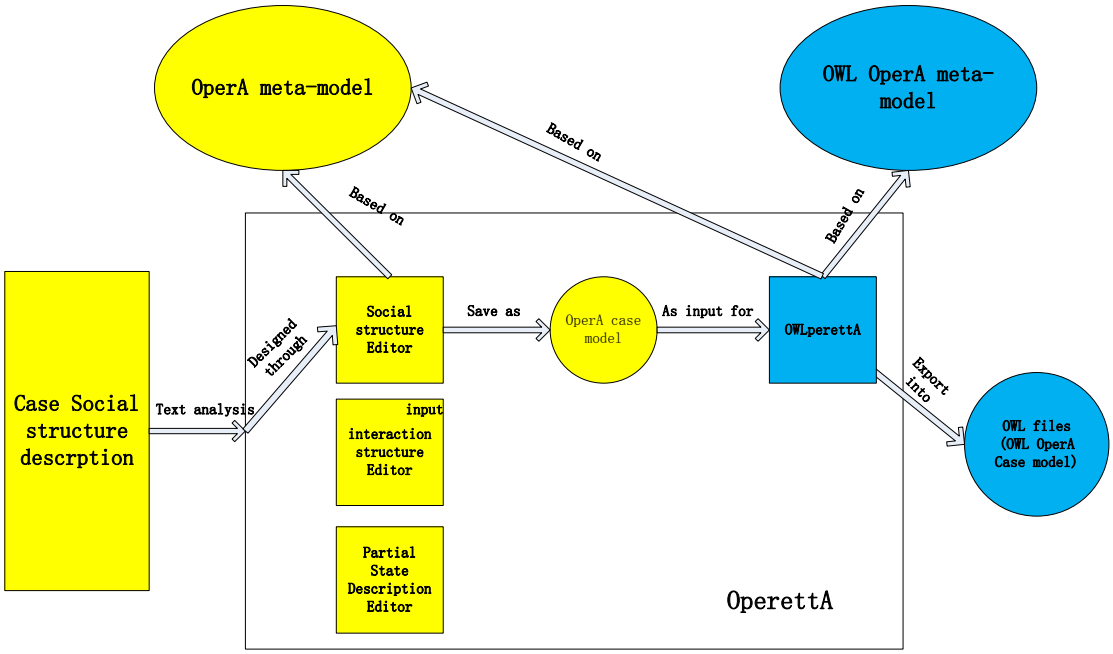


Figure 4.3 Usage of OWLperettA in designing an OWL OperA case model

According to Figure 4.3, we can see that, in order to design an OWL OperA meta-model, we must use both OperettA and OWLperettA. OperettA is a tool that consists of several parts: a Social Structure Editor, an Interaction Structure Editor and a Partial State Description Editor. We need to focus on the Social Structure Editor because our scope is the social structure of OperA; we need to use this part to create an OperA case model for evaluation. Social structure is a graphical environment for the specification, analysis, and design of the SS of an OM. After designing through OperettA, we can obtain an OperA case model that is XML-based. We have embedded OWLperettA into OperettA as well, so we can import the designed OperA case model into OWLperettA directly. Finally, the OWL OperA case model is exported by the OWLperettA.

4.5 Conclusions

In this chapter, with the support of the OWL OperA meta-model, we have developed OWLperettA. Through OWLperettA, we can automatically export OWL2 files by transforming an OperA case model into an OWL OperA case model. Combining OWLperettA and OperettA, we can create an OWL OperA case model through an interface in Eclipse instead of editing manually. As we outlined in Chapter1, we wish to create a tool to solve research question 2, which leads to research question 3. After our implementation, we have created OWLperettA, which can answer research question 3. However, even though the combination of Chapters 3, and 4, we still cannot verify whether a correct meta-model has been successfully designed or not. We propose methods to evaluate research questions 1 and 2. Based on these research questions, we come up with some added functionalities that the OWL OperA meta-model may bring us. These added functionalities are described in following chapter. After evaluating these added functionalities, we can verify the OWL OperA meta-model by answering research questions 1 and 2.

5. Predicted Added Functionalities

In the previous chapter, we designed the OWL Opera meta-model under the guidance of the OperA meta-model. Comparing the OperA meta-model to the OWL OperA meta-model, there are several obvious functionalities that can't be fulfilled by the former; however, they can be realized through the latter. The OperA meta-model and OperettA can be improved by realizing our predicted added functionalities which can answer the research questions we described in Chapter 1. All these predicted added functionalities occur because of the semantic meaning and stronger constraints expression of the OWL2 and complete structural advantages. They will be evaluated in Chapter 6.

In this chapter, we introduce several functionalities that can't be identified or fulfilled when using the OperA meta-model to design the OperA case model; but can be realized through the OWL OperA meta-model. All these functionalities will contribute to future design of organizational models. After a detailed description of the functionalities, we briefly explain how we can evaluate or fulfill the functionalities. All the improved functionalities should be limited in the scope of social structure (SS).

5.1 Categories of Consistencies

In the OperA meta-model and OperettA described in (Aldewereld & Dignum, 2011), some limited problems will exist when designing through EMF. The OWL OperA meta-model is predicted to be able to solve some of the problems. Predicted added functionalities may occur because of the more powerful vocabulary and the semantic meaning of the expressions in OWL2. All these predicted functionalities are the key to our research questions 1 and 2 as outlined in Chapter1. We intend to divide the predicted added functionalities into easily comprehensible categories of consistencies according to the research questions. The consistencies are divided into two categories: structural consistencies and semantic consistencies.

Category1: structural consistencies.
As we described in Chapter 1, research question 1 intends to check the consistency of a case model of OperA. Added Structural functionality represents the added functionality that occurs through the improvement in solving problems in the structure of an OWL OperA case model. Functionality 1 mentioned below is the core predicted structural functionality we will describe and evaluate in our project.

Functionality 1: Checking the structural consistencies (errors) of the ontology.
OperettA, as described by (Aldewereld & Dignum, 2011), is a complete software to design the OperA case model. All the features of the OperA meta-model can be realized by OperettA. So the result of the designed OWL OperA case model should be structured correctly if we design the OperA case through OperettA for exporting by OWLperettA. However, when we design an OWL

OperA case model manually by editing xml, there is a high potential to make structural inconsistencies or errors. However, these structural inconsistencies of the manually edited OWL OperA case model should be checked.

Category2: semantic consistencies.

As we described in Chapter 1, research question 2 intend to check semantic consistency of the case model of OperA. When we address it, there should be added semantic functionality that cannot be checked, even through OperettA. Added semantic functionalities represent the added functionalities caused by the improvement in solving the semantically consistent problems of the case model. Functionalities 2 through 4 mentioned below are the core predicted semantic functionalities we will describe in our project. As we mentioned in the abstract, there are two ways to create an OWL OperA case model. Semantic inconsistencies have high a potential occur in both ways.

Functionality2: Different Individuals in the same class should not have the same individual name.

When we design an OWL OperA case model, for the same class, different instances can be given the same individual name at the same time. For example, in a simple OperA case model "Conference" with two role instances: General-chair and pc-chair, they both have the role name attribute called: chairman. Normally, we know that these two instances are different and they should not have the same role name so that they can be identified easily from the semantic level by the future users or designers. In other words, they should have semantic consistency. However, the OperA meta-model allows for the OperA case model to share the same name for different instances in same class. After the design of the OWL OperA meta-model, we should be able to check the semantic inconsistencies that occur in the OWL OperA case model designed through OWLperettA. In this functionality, it means that different individuals in the same class with the same individual name can be checked.

Functionality3: Different individuals in different classes should not have the same individual name.

The situation described above is almost the same for functionality 3. For different classes, the instance of one class can be given an individual name equal to the individual name of an instance in another class at the same time in the same model. For example, in a simple OperA case model "Conference" with one Right SubmitPaper and one objective instance PublishPaper, they both have the right name equal to the objective name called: PublishPaper. Normally, we know that these two instances are different in different elements. They should not have the same name so that they can be identified and queried easily from the semantic level by the future users or designers; as above, they should be semantically consistent. However, the OperA meta-model allows for the OperA case model to share a given individual name for instances in respectively different classes. After its design, we should be able to check the semantic inconsistencies that occur in the OWL OperA case model through OWLperettA. In this example, the occurrence of different individuals in different classes with same individual names can be checked.

Functionality4: Objectives should not have a Sub Objective of themselves.
A particular semantic inconsistency will occur when we design the OWL OperA case model by editing. We can design objective individuals to have a Sub Objective of themselves. For example, let us assume that there are three objective individuals, O1, O2, and O3. O2 is a Sub Objective of O1. O3 is a Sub Objective of O2. It is possible for us to create a semantic inconsistency in which O3 has a Sub Objective of O1 while editing the OWL OperA case model. Because the OWL OperA meta-model is the basis of the OWL OperA case model, we should be able to check the semantic inconsistencies that occur in editing the OWL OperA case model. In other words, the objective individuals that have a Sub Objective of themselves can be checked.

5.2 Possible Solutions

As mentioned in Chapter 1, there are three potential methods in Protégé to evaluate the case model: the reasoner (Hermit or FACT+), the DL-Query tab and the SPARQL Query tab (described in Chapter 2). In this project, SPARQL Query should be the main method for added semantic functionalities. The Reasoner and the DL-query tab cannot work so effectively in semantic added functionalities, but reasoner may be suitable for added structure functionalities. The reasons are elaborated below.

The reasoner
As (Shearer et al., 2008) have described, the Hermit reasoner follows the novel “hyperTableau” calculus algorithms to realize inconsistent checking, class, objective functionalities, and data functionalities classification. Reasoner is also the basis for the DL-query tab, which will be explained in the next section. The Hermit reasoner is a functional reasoner for OWL2.

The Hermit reasoner can identify the subsumption classes, objective properties or data properties. It will check the inconsistencies for the classes, objective properties, data properties and even the Individual by the constraints definition or axioms offered in the OWL-DL. These inconsistencies can be regarded as structural inconsistencies. However, it doesn’t take initiative to determine by inputted detailed data information that contains semantic inconsistencies. For our added functionalities 3 through 5, semantic consistencies can be used as the identification of the core elements we wish to obtain. Therefore, the reasoner doesn’t work well in functionalities 2 to 3.

But for the added functionality 1, the reasoner may be used for the evaluation. The reasoner can be used for checking consistency of ontology. In this project, some structural errors may occur during the design of the OWL OperA case model by editing if the workload for editing is large. By inputting the OWL OperA case model in Protégé and starting the reasoner, we can ascertain whether the OWL OperA case model is consistent or not. Inconsistency will be presented in a red text or explanation box. Reasoner can also be used as an auxiliary means to export inferred ontology.

The DL-query tab

As the DL-query tab[2] has introduced, there are two main limitations of our analysis that would make it impossible to realize the added functionalities:

Limitation 1: The DL-query tab is a type of querying based on the reasoner. It should ensure that, before using the DL-query, the reasoner should be used first. The active ontology will be classified. The DL-query can only execute a query when the ontology is classified (as we have mentioned in Chapter2). One more step should be taken before using the DL-query tab. For functionality 1, if we wish to use the DL-query, we should use the reasoner first. However, the reasoner can directly check the inconsistencies in functionality 1, and thus, we do not need to use DL-query tab.

Limitation2: The DL-query tab is based on the Manchester OWL syntax by (Horridge & Patel-Schneider, 2009). It can be used to query the data information store in the OWL ontology. However, it can't query the individuals who share the same specific data information that can be regarded as semantic information. Manchester syntax only offering the expression for query (with basic logical symbols), and it does not support the syntax to search specific data information. However, this kind of query is the core of our evaluation of functionalities 2 through 4. We simply wish to query these individuals and objective individuals with a semantic relation without knowing the specific value. So the DL-query can't be used as the method for realizing the added functionalities 2 through 4.

Combining the above situations, we can conclude that the DL-query is not appropriate fit for our evaluation to realize the predicted added functionalities.

The SPARQL-query Tab

As (Sirin et al., 2010) have described, Terp has been designed and implemented. Terp syntax allows class, functionality, and data range expressions, expressed in Manchester syntax, to be used inside SPARQL queries. With this information, we will gain the ability to write queries to query OWL2 ontology in our evaluation in Chapter6. The SPARQL query tab of Protégé provides us an environment to query OWL2-based ontology by using Terp. It is possible for the SPARQL query language to query the relevant information through semantic information. Therefore, we believe the SPARQL query tab could be a suitable method for the added functionalities evaluation.

However, for functionality 1, it is proposed to check the structural consistencies occurring in the case model. When the OWL OperA case model is edited with structural inconsistencies, it is not a correct OWL2 ontology. When we use the SPAQRL query tab into a wrong OWL2 ontology, it is not possible to check the inconsistencies of a wrong ontology. So, the SPARQL query cannot be used for evaluate the functionality 1.

In our project, we prefer to use the SPARQL query tab to perform the evaluation of functionalities 2 through 4. Because of functionalities 2 to 3, we should check relevant information based on semantic information in the OWL OperA case model. Individual names are treated as the

[2] DL-query tab, retrieve from: http://protegewiki.stanford.edu/wiki/DLQueryTab, 2014

semantic information contained in the OWL OperA case model. While using the SPARQL query tab, we should input relevant query statements for specific queries. These relevant query statements are shown in AppendixC. Detailed evaluation methods will be provided in the evaluation part. However, we should use the SPARQL query tab for checking the objective relations information, which is regard as semantic information as well. However, through OperettA, it is not possible to create these types of relationships inconsistencies as well. However, when we create an OperA case model, it is possible to create these types of inconsistencies by editing these types of inconsistencies. Therefore, in order to evaluate this functionality, we should edit these types of semantic inconsistency manually instead of using OperettA. Combining the reasoner and the SPARQL query tab, we are able to check the inconsistency of the OWL OperA case model.

In this chapter, we introduced the predicted added functionalities that the OWL OperA meta-model will bring to us after we design it, by comparing it to the OperA meta-mode. We also introduced the candidate methods for the different added functionalities evaluations. Generally, we can divide the functionalities into two main categories: added structural functionalities and added semantic functionalities. Research questions 1 and 2 may be solved through these methods as we have discussed in this chapter. But the predicted added functionalities should also be evaluated by these methods. In the following chapter, we will perform the evaluation through providing detailed methods and the detailed OWL OperA case model for respective functionalities.

6. Case Study and Evaluation

In Chapter 3, we introduced the OWL OperA meta-model. Because of the uncertainty of this model, we must evaluate it by designing a case model. This evaluation should be able to verify the added functionalities we predict to fulfill. We should abstract the social structure parts for evaluation. As the methodology for our project described in Chapter 1, the case study should be used as the basis of evaluation. As(Yin, 2009) has pointed out, if we use case study as evaluation method, the case study should be a less desirable form of research question, does not require control behavioral events and focuses on a contemporary event. Therefore, we would like to use the case study for our project evaluation, dependent on our project's methodology. In our project, we wish to verify the effectiveness of the OWL OperA meta-model, which could be ascertained by evaluating the predicted added functionalities. In order to realize this evaluation, we create corresponding scenarios for the evaluation to obtain respective results. After analyzing the results, we reach our conclusion.

In this chapter, a case called "conference" will be introduced for the evaluation. We are able to create the OperA case model through OperettA and OWLperettA or through editing. These are two ways in which we create a case, as we discussed in Chapter 4. Evaluation is described in detail to verify the OWL OperA meta-model. For all the added functionalities we predicted in Chapter 5, we set up the methods and scenarios respectively. Analyzing the result by putting the method into the scenario, we can ascertain whether the functionalities are validated or not. We will introduce the conference case at the beginning of this chapter as the basis of the case study.

6.1Description of Case

Conference organization is an exemplary case for the organizational model. We choose conference organization as our case example for several reasons. Firstly, all types of conferences are held in various places at any given time, which makes the conference universal. In addition, a conference can cover all the elements of SS in OperA, which will make the evaluation credible. Finally, a conference is relatively easily understood by all of us without any difficult academic knowledge. Below, we elaborate some of the main descriptions that have been provided by researchers.

Case description

As (Zambonelli, Jennings, & Wooldridge, 2001)have described, a conference organization is used to support the management of an international conference. Setting up and running a conference is a multi-phase process that involves several types of individuals. During the submission phase, authors must be informed that their papers have been received and they need to be assigned a submission number. Once the submission deadline has passed, the program committee (PC) must handle the review of the paper, typically by contacting potential referees and asking them to review a number of the papers. Eventually, the reviews come in and they are used to decide

about the acceptance or rejection of the submission. Authors must be informed of these decisions and if their paper has been accepted, they must be asked to produce a revised version of it. Finally, the publisher must collect these revised versions and print the proceedings.

(M. Dignum, 2003)has stated that the global objective of this society is to realize a conference. Stakeholders in this society are the organizers, the authors, PC members and participants. The objective of the organizer is to organize a successful conference, authors wish to have their papers accepted, PC members aim at assuring the quality of the program and participants hope for a high quality conference. Facilitation activities can be described in terms of an organizer role that administrates the conference and a chairperson role responsible to regulate conference sessions.

According to (DeLoach, 2002), the system starts by having authors submitpapers to an individual in the paper database (PaperDB) role, who is responsible for collecting the papers, along with their abstracts, and providing copies to reviewers when requested. Once the deadline has passed for submissions, the person responsible for partitioning the entire set of papers into groups to be reviewed (the Partitioner role) asks the PaperDB role to provide him or her with the abstracts of all papers. The Partitioner partitions the papers and assigns them to a person (the Assigner) who is responsible for finding *n* reviewers for each paper. Once assigned a paper to review, a Reviewer requests the actual paper from the PaperDB, prepares a review and submits the review to the Collector. Once all (or enough) of the reviews are complete, the Decision Maker determines which papers should be accepted and notifies the authors.

Based on the information provided above, we already have enough to abstract the case into a well-formatted description. In this description, all elements, relationships, interactions and values will be added. We abstract them in the following section.

Case abstraction

According to the descriptions of three kinds of conference organization cases, we can abstract their core elements and interactions as shown below in Figure 6.1:

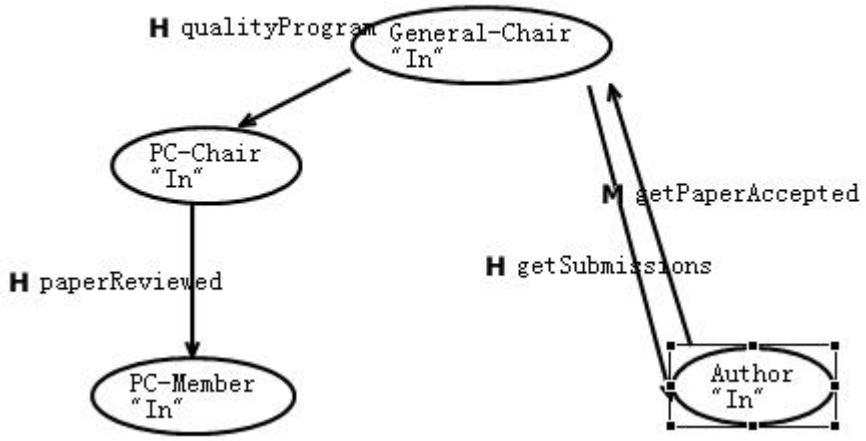


Figure6.1 Conference organization interaction

According to Figure 6.1, four stakeholders exist in this organization. Interactions are different among different stakeholders. We create the abstraction on the basis of the information in the case descriptions presented above, as set forth in Tables6.1 and 6.2below:

Stakeholders	Explanations
General-chair	The general-chair is the organizer, who aims at holding a successful conference. In order to guarantee the success of the conference, we must fulfill several objectives, such as obtaining submissions from authors, maintaining the quality of the Program, forming the PC committee for the reviewing and so on.
PC-chair	The program committee chair is the leader of the PC committee, who, along with the management of the PC-members, maintains the quality of the program.
PC-member	The program committee member is the basic component of the PC committee, who aims at correcting the reviews.
Author	Authors are the core elements of the conference. They publish a paper of their own. These papers are checked by the PC committee of the conference.

Table 6.1 Stakeholders

Stakeholders	Interactions	Interaction aims
General-chair	PC-chair, Author	The quality of the program can be fulfilled through the interaction.
PC-chair	PC-member, General-chair	They ensure that the papers are reviewed, and the quality of the program can be fulfilled through the interaction.
PC-member	PC-chair	They fulfill the review of the papers
Author	General-chair	The general-chair obtains submission of the papers from the authors. In return, the author will obtain the information concerning the paper's acceptance status from the general-chair.

Table 6.2 Interactions among Stakeholders

The information in the above tables represents the basic components of conference organization. If we wish to design the conference organization into an OperA case model, we need detailed specification of these basic components by mapping them onto an OperA meta-model.

In Tables 6.1 and 6.2, we have the analysis of the stakeholders and the various interactions within the conference case. With these elements, we are able to design the conference case into OperA in the next section.

6.2 OperA Case Model Designing

Because of our scope, we only need to design the SS of the conference organization. According to the abstraction above, we can assign different elements in the case to the OperA meta-model. According to Figure 6.1, we can see that Role, Dependencies and Objective of Dependencies of the conference case model have been presented clearly. The interactions between role and the dependencies are also indicated by Figure 6.1. However, there are some types of information that have not been described yet. We must supplement our OperA case model with the Right, Objectives and Objective of Role. The complementary is shown in Table 6.3:

Role	Right	Objective	Sub-onjective
PC-chair	empty	QualityOfProgram	PaperReviewed AssignPaper DecisionOnPaper
PC-member	empty	CorrectReview	empty
General-chair	chooseAuthor	SuccessfulConference	GetSubmission, QualityProgram, FormPC, SendCFP Notify.
Author	empty	PublishPaper	empty

Table 6.3 Conference specification in OperA

According to this abstraction, we can implement these designs through OperettA or by editing. Thus, we have successfully designed the conference OperA case model. This conference OperA case model is used to support the evaluation methods described in the following section.

6.3 Evaluation Methods

As we mentioned in Chapter5, we have designed methods for evaluating the added functionalities. Because of the different categories of added functionalities, we should specify different methods to verify different functionalities. These methods provide the processes for performing case evaluation (through an experimental design). Combining figures and text in the following paragraphs, we describe each method clearly and separately.

Method for evaluating functionality 1:

We introduced functionality1 in Chapter5. The case ontology based on the OWL OperA meta-model designed by the designer can be checked for inconsistency automatically when an error occurs. At present, there are several methods to create the case ontology. Firstly, we can use OperettA to create the case model, which can be exported into OWL2 by OWLperettA. Secondly, editing an OWL document is another way to design a case ontology based on the OWL OperA meta-model. In setting up our method, we choose the second way to design the case ontology.

For the first way, OperettA is a tool to create a structurally correct OM. The designed OperA case model can be exported into OWL2, guided by the OWL OperA meta-model, which is a conversion of the OperA meta-model without changes. Thus, after we design a structural correct case model through OperettA and export it into OWL2 ontology, the case ontology will always be structurally consistent. For the second way, there is the possibility to design the case ontology with structural errors and the case ontology can be input into Protégé. Accordingly, we should use the second method to design the case, with all other processes handled in Protégé. The method to evaluate functionality 1 is shown in Figure 6.2.

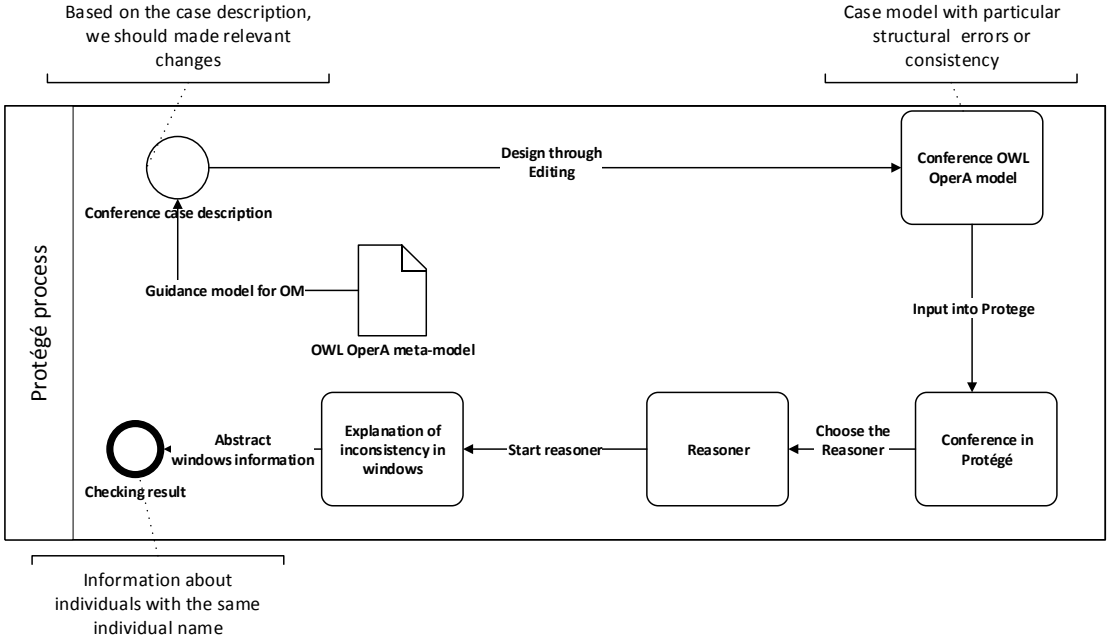


Figure 6.2 Method to evaluate functionality 1

Based on Figure6.2, we can set up the steps as follows.

Firstly, there is an existing OWL OperA meta-model, (designed in Chapter 3). We design the conference case based on the OWL OperA meta-model. As we have described above, we design the case by editing OWL2.

Secondly, we should make specific structural errors in designing the conference case. A conference ontology will be created. We name it conference_with_editing_error.OWL. It represents the conference OWL OperA model in Figure 6.2. Then, we input this OWL document into Protégé.

Finally, we start the reasoner. A window called “help for inconsistent ontologies” will occur and the ontology on the left side will turn red, which means that the ontology is inconsistent. After pressing the explain button, the inconsistent results are presented in this window.

Method set up for evaluating functionality 2:

We introduced functionality2 in Chapter5 – different Individuals in the same class shouldn’t have the same individual name. In particular, we demand to export a correct ontology so that the result is credible. For individuals in the same class, we should not enter the same individual name for each individual. The same individual name for different individuals will cause a misunderstanding of the ontology. We regard these as semantic inconsistencies. However, OperettA allows for defining different individuals with different individual names. So we should

find solutions for the designer to allow for identifying those individuals that have the same name. After locating these individuals, we can modify these names so that the ontology can be expressed more clearly. The method to evaluate functionality 2 is shown in Figure 6.3.

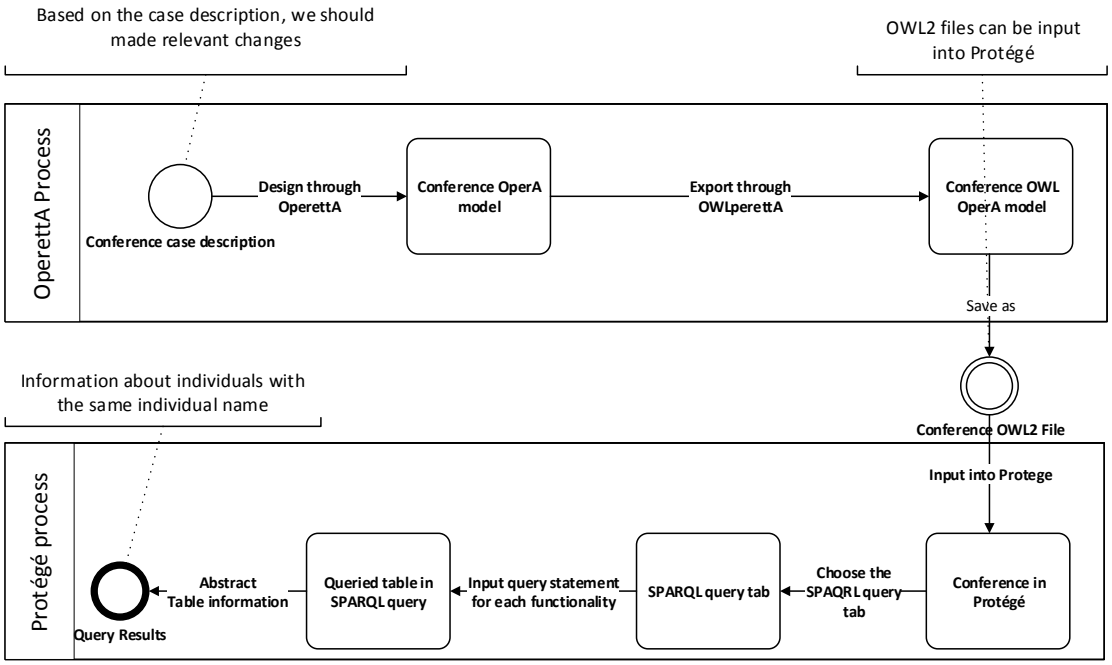


Figure 6.3 Method to evaluate functionality2

Based on Figure6.3, we can set up the steps as follows.

Firstly, through OperettA, we can create the conference OperA model easily. Then, we make changes associated with individuals' names based on this model.

Secondly, after we implement all these changes, we can obtain "conference_sameclass_name.OWL" by using the OWLperettA. This document represents the conference OWL OperA model for evaluating functionality 2.

Thirdly, we should input this OWL file into Protégé, and then choose the SPARQL query tab. After inputting our designed query statements listed in AppendixC.1 for SPARQL queries, we press the "execute" button.

Finally, we obtain a table that contains the results of all the information that we wish to query.

Method for evaluating functionality 3:

Functionality3 is also introduced in detail in Chapter5 – different Individuals in different classes shouldn't have the same individual name. The set-up for the method to evaluate this functionality is very similar to the method for functionality 2 described above. The steps are the same as Figure6.2 describes. However, there are two divergent points.

1. In the first step, the changes made here are different from the changes made in the method for functionality3. However, these changes are all based on the "conference OperA model" and are so-called semantic inconsistencies.
2. In the third step, the designed query statements shown in AppendixC.2 are different from

those used in the method for functionality2.

These points will be distinguished in the scenarios described in the following section. With these two distinctions, our changed OWL OperA case model, called "conference_differentclass_name.OWL," can be designed. We can set up the correct method for evaluating functionality3as shown in Figure 6.3.

Method for evaluating functionality 4:

As we described in Chapter 5, functionality 4 indicates that individuals of objective shouldn't have a Sub Objective of themselves. In designing the OWL OperA case model, for the individuals of objective, we shouldn't create the relationship "has Sub Objective" between individuals of the objective and themselves. Through OperettA, execution errors will be reported when this kind of situation occurs. According to the empirical circle of our project methodology described in Chapter 1, we are not able to continue designing in OperettA because of these execution errors' appear, we should restart it for continuous designing. However, there is a high potential for designers to create these types of interactions, which are semantic inconsistencies, by editing. Therefore, we should find solutions to check for this semantic inconsistency. After checking the consistencies, we can make modifications so that the conference OWL OperA model (conference ontology) can be expressed more clearly without semantic inconsistencies. According to the description in Chapter 5, we select the SPARQL query tab to conduct the evaluation. Figure6.4 shows the processes of evaluating functionality 4 through the SPARQL query tab.

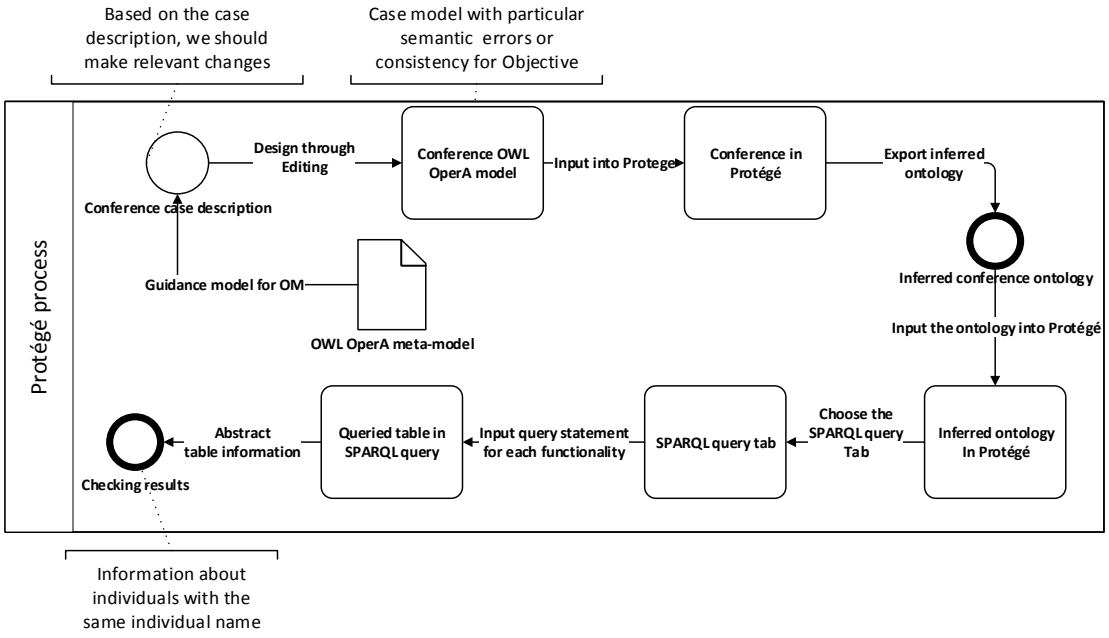


Figure 6.4 processes of methods to evaluate functionality 4

Based on Figure6.4, we can set up the steps as follows.

Firstly, there is an existing OWL OperA meta-model. We design the conference case based on the OWL OperA meta-model. As we have described above, we design the conference case by editing OWL2.

Secondly, we can make specific structural inconsistencies in designing the individuals of Objective.

Then, we save this file as “conference_objective.OWL.”
Thirdly, we should input this OWL file into Protégé. After using the reasoner, we can export the inferred conference ontology called “conference_objective_inferred.OWL.”
After inputting “conference_objective_inferred.OWL” into Protégé again, we should choose the SPARQL query tab. We then input our designed query statements, which are listed in AppendixC.3, for SPARQL query, and press the “execute” button.
Finally, we obtain a table that contains the results of all the information that we wish to query.

With these four methods as guidance, we can perform the evaluation separately for different functionality. In order to implement the methods we must set up experimental scenarios. These various experimental scenarios are elaborated in the following section.

6.4 Experiments

In our case study, we use purposive samples.[3]Purposive sampling is a case study method for evaluation, which is used when an evaluator is studying a particular phenomenon and wants to ensure that examples of it show up in the study. We aim at obtaining some phenomenon or results demonstrating that something can be reasoned or that it occurs as the purposive samples show us. As evaluator, we must know the potential of the example to reveal those kinds of reasons. As stated above, for all functionalities, we create corresponding experiments for evaluating them. Four simple experiments are created by including the scenario and the experiment results. Expected results are the designed inconsistencies that include both the structural inconsistencies and the semantic inconsistencies. The expected results will be described while we are elaborating the scenarios. Each scenario will have its own experimental results.

6.4.1 Experiment One

6.4.1.1 Scenario Functionality One

For evaluating functionality1, we should design the case ontology by editing the document. In order to create the case conference ontology, we should create several individuals for each element in SS. We should also create functionality to show the relationships between different individuals. We design several structural inconsistencies in the case ontology. These errors are shown below in Table 6.4:

NamedIndividuals	structural inconsistencies
PC-Chair	<HasRoleType rdf:resource="&untitled-ontology-63;Ex"/> <HasRoleType rdf:resource="&untitled-ontology-63;In"/>

[3]Edith D.Balbach, Using Case Studier to do Program Evaluation, retrieve from:http://www.case.edu/affil/healthpromotion/ProgramEvaluation,2014

PC-Chair	<HasObjective rdf:resource="&untitled-ontology-63; chooseAuthor_5ae8816a-3f1d-429e-bed5-41e6930248e9"/>

Table 6.4 structural inconsistencies

In Table 6.4, with regard to the PC-Chair, we create two specific inconsistencies. The PC-chair should have only one role type, but, we design two role types, “In” and “Ex”. The PC-Chair has several objectives. We design it such that it has the Objective of chooseAuthor. Nevertheless, chooseAuthor is an individual of type right. Therefore, these two errors will make the ontology inconsistent. The ontology with structural inconsistencies will be called conference_with_editing_error.OWL.

After inputting this document into Protégé, we will perform testing on this ontology by using the reasoner. We expect that the reasoner can locate this inconsistent place automatically. These inconsistencies are the key to transform the ontology into a correct ontology.

6.4.1.2 Experiment Scenarios One Results

Through the design of scenario one, we can obtain the case ontology conference_with_editing_error.OWL with structural inconsistencies in it. Based on this ontology, we use reasoner to do the checking. The experiment results are presented in AppendixB.1. We abstract the information into the Table6.5 below:

individual	inconsistency location	explanation
PC-Chair	PC-Chair HasObjective of ChooseAuthor	Objective DisjointWiht Right HasObjective Range Objective choosAuthor is Right
PC-Chair	PC-Chair HasRoleType In PC-Chair HasRoleType Ex	Ex different from In Role HasRoleType exactly 1 RoleType In, Ex Type RoleType.

Table 6.5 Experiment results of scenario one

Table 6.5 shows the location of the inconsistencies. In addition, explanations are presented for easier modification. For the first inconsistency, after analyzing the explanations, the PC-Chair shouldn’t have objectives out of the Range Objective. However, chooseAuthor is not in the range of Objective. Therefore, we should modify chooseAuthor into an individual in the Range Objective. For the second inconsistency, explanations tell us that the PC-Chair is unable to have two different Role Types because of the candidate constraints of exactly one role type. In order to modify it, we should delete one of the relationships. In conjunction with Table6.4, all the inconsistencies have been located. They explanations also provide an easy way to modify these inconsistencies. That means, for the OWL OperA meta-model, there is automatically an easier way for us to check consistency and design the correct ontology.

6.4.2 Experiment Two

6.4.2.1 Scenario Functionality Two

For evaluating functionality2, as mentioned above, we must make changes based on the "conference opera model". There are four main elements that we must input as data into this model. These elements are respective role, dependency, objective, and right. Each of them has several individuals and all the individuals should have the name. In order to evaluate the individual names, we must make some modifications to the names of individual for each of the elements. In Table 6.6 below, we can see the added information we need.

Element	Create new individual?	Individual name
Role	✓	PC-Chair
Dependency	✓	paperReviewed
Right	✓	chooseAuthor
Objectives	✓	correctReview

Table6.6 Semantic inconsistencies 1

As Table 6.6 shows, we have created two individuals with the same name in each elements. Currently, there are two individuals with the name PC-Chair in the class role and two dependencies individuals named paper Reviewed in the class Dependency. It is the same with the Right and the Objectives, with two individuals with the same individual name chooseAuthor and correctReview. "Conference_sameclass_name.OWL" will be designed based on the Conference opera model.

However, for a model, it not wise to have two individuals in the same element with the same name. OperettA allows for creating two individuals in the same element with the same name, which will make the designer puzzled about the model. Therefore, we expect that, after we input the changed model into OWLperettA, export it into the OWL file, with the method mentioned in section 6.3, we can obtain the information concerning all the individuals who have the name PC-Chair, paperReviewed, chooseAuthor, and correctReview. All this information has been added into Table6.2.

6.4.2.2 Experiment Scenarios Two Results

After performing the evaluation with the method provided for functionality3, by inputting model "conference_sameclass_name.OWL" and using the particular query statements presented in AppendixB.2, we can obtain the information abstracted into the Table 6.7 below:

element	Individual 1	Individual 2	Name
Role	PC-Chair_10	PC-Chair_74	PC-Chair
Dependency	D10	D69	paperReviewed
Objective	correctReview_0f	correctReview_0c	correctReview

Right	chooseAuthor_255	chooseAuthor_492	chooseAuthor

Table6.7 Experiment results of scenario two

In conjunction with Table 6.6, we see that all the added individuals have been queried by the SPQRAL query tab. The results also contain the name value for each added individual. In addition, individual 2 is queried for each element because it has the same individual name as individual 1. After we have designed the "conference_sameclass_name.OWL", we hope that, there is a solution for finding the individuals in the same elements with the same name. What's more, when we located these individuals, we can re-design them and modify their individual names to make the ontology distinct and easily to be understood by users of the OM. After this evaluation, semantic inconsistencies have been checked and we can verify that functionality 2 has been realized well.

6.4.3 Experiment Three

6.4.3.1 Scenario Functionality Three

Scenario four will take use of method for functionality3 which is quite similar to method for evaluating functionality3. However, their scenarios are totally different. Even though, we should base on conference opera model to make changes, these changes are quite different from the changes in scenario two. As we know, there are four main elements in the model that we need for inputting data. Each of them has several individuals and all the individuals should have a name. To evaluate the name, we should add three individuals in one element with individual names equal to the names of individuals in other elements respectively. As a consequence, there should be six designed semantic inconsistencies for the names to cover all the elements. These changes are shown in Table 6.8:

Changes.No	element	Operation
1	Right	Create individual1 with name General-Chair (role name)
2	Right	Create individual2 with name publishPaper (objective name)
3	Right	Create individual3 with name paperReviewed (dependency name)
4	objective	We should create the objective PC-Chair as objective of role PC-Chair.
5	Objective	Create a individual with name PC-Chair (role name)
6	dependency	Create a individual with name PC-Chair (role name)

Table6.8 semantic consistencies 2

As Table 6.8 has shown, "conference_differentclass_name.OWL" is created through these operations based on the Conference OperA model. We create three rights individual with

different names equal to the names of individuals in class role, objective, and dependency respectively. According to the OperettA tutorial[4], dependency names are equal to the objective names, which depend on dependant and dependee. We don't have to create more objective individuals with the same individual name in dependency individuals, because they are always equivalent when the dependency only has one objective. Therefore, if we have the demand to create a dependency individual, we should create an objective individual (such as the no.4 PC-Chair objective individual) that is related to this dependency individual and can give the dependency individual a name.

However, individuals of different elements must be different individuals. The name is the key to identify them from the visual interface. Thus, creating these individuals and relationships will make the whole model vague for OM users, because of the so called semantic inconsistencies. After inputting the changed model, we export it through OWLperettA and input it into Protégé, as the method explained that, information that individuals in different elements with the same individual names is expected to be queried.

6.4.3.2 Experiment Scenarios Three Results

When the related query experiment was performed on "conference_differentclass_name.OWL", some information was presented in the table at the bottom of Protégé. This information has been saved in AppendixB.3. We have abstracted the relevant information into Table 6.9:

Element1	Individual1	Element2	Individual1	Name value
Right	paperReviewed_558	Dependency Objective	D10 paperReviewed_3f6	paperReviewed
Right	General-Chair_187	Role	General-Chair_5ae	General-Chair
Right	correctReview_6f7	Objective	correctReview_3b9	correctReview
Dependency	D102	Role	PC-Chair_eab	PC-Chair
Objective	Author_ae3	Role	Author_b3b	Author
Objective	PC-Chair_365	Role	PC-Chair_eab	PC-Chair

Table6.9 Experiment results of scenario three

We should now compare it to Tables 6.8 and 6.9. Through the query statements we designed using the SPARQL query tab, all semantic inconsistencies that individuals in different elements have same individual names have been queried. Individual1 represents the individual of element1, individual2 represents individual of element2. We learned that Element1 and individual1 are consistent to the added information we have created in "conference_differentclass_name.OWL" through comparative analysis. Element2 and individual2 are the equivalent individuals that exist in the model. In the Table 6.9, one more pair of equivalent individuals occurs just because of the passive added objective individual PC-Chair for getting a dependency individual D102 .After we designed "conference_differentclass_name.OWL", we desired to find out the solution for locating individuals in different elements of the same name. According to the Table 6.9, each row of the table shows the results of two particular

[4] TUTORIAL: OperettA, retrieved from http://ict1.tbm.tudelft.nl/operetta/downloads/tutorial.pdf, 2014

individuals in element1 and element2 with the same individual name. That proves that semantic inconsistencies have been checked. The method is the validated solution for functionality 3.

6.4.4 Experiment Four

6.4.4.1 Scenario Functionality Four

In order to evaluate functionality 4, we propose to design a case ontology by editing a document. In order to create the conference ontology, we should create several individuals for each element in SS. We should also create property to show the relationships between different individuals. However, there are potential semantic inconsistencies during the design process. We design several semantic inconsistencies in the case ontology about objective semantic inconsistencies. These inconsistencies will are below in Table 6.10:

NamedIndividuals	inconsistencies (relations)
decisionOnPapers _3d1efd5f-385f-4cb4-a9df-d8177eedc487	<HasSubObjective rdf:resource="&untitled-ontology-63; getSubmissions_04796a09-266e-43cf-93a8-17712e0632bf"/>
getSubmissions _04796a09-266e-43cf-93a8-17712e0632bf"/>	<HasSubObjective rdf:resource="&untitled-ontology-63; decisionOnPapers_3d1efd5f-385f-4cb4-a9df-d8177eedc487/>

Table 6.10 Semantic inconsistencies_objective

As Table 6.10 has shown, “conference_objective_inferred.OWL” is created through these operations based on the conference OperA model. For the individuals named decisionOnPapers and getSubmission, we create a relationship with themselves through “sub objective”. DecisionOnPapers has the sub objective of getSubmission. GetSubmission has the sub objective of DecisionOnPapers as well. There is no structure error in syntax. They are the semantic inconsistencies we have mentioned above. These inconsistencies cannot be checked by either editing or OperettA.

However, these semantic inconsistencies show that individuals of objective can have a sub objective of themselves. This will make the ontology vague. After inputting the changed model, we export it through reasoner and input it into Protégé again. As the method explained, the SPARQ query tab is expected to present the individuals who have sub objectives of themselves in objective class.

6.4.4.2 Experiment Scenario Four Results

According to scenario four, we export “conference_objective_inferred.OWL” through the reasoner in Protégé. By operating through the SPARQL query tab as we described in the method

for functionality4, we can obtain a table with the relevant inconsistency information in it. This information has been saved in AppendixB.4. We have abstracted the information into Table 6.11 below.

Objective	**Sub objective**
decisionOnPapers _3d1efd5f-385f-4cb4-a9df-d8177eedc487	decisionOnPapers _3d1efd5f-385f-4cb4-a9df-d8177eedc487
getSubmissions _04796a09-266e-43cf-93a8-17712e0632bf"/>	getSubmissions _04796a09-266e-43cf-93a8-17712e0632bf"/>

Table6.11 Experiment results of scenario four

As shown in the Table 6.11, there are two individuals of the objectives called decisionOnPapers and getSubmissions. Both have sub objectives of themselves. By Comparing Tables 6.8 and 6.4, the semantic inconsistencies we have created while designing the "Conference_objective.OWL": "individual decisionOnPapers has the sub objective of getSubmissions" and "individual getSubmissions has the sub objective of decisionOnPapers". Therefore, for the inferred ontology "Conference_objective.OWL", both will have a sub objective of themselves in the semantic level. Through our evaluation methods, we can query these two individuals out. In other words, with our design of the OWL OperA meta-model, we are able to check the semantic inconsistencies about a particular objective that cannot be checked when we use the OperA meta-model to design the OperA case. This proves that functionality 4 is validated as well.

According to the evaluation and discussion above, all the functionalities can be proven to be validated. There are indeed functionalities that occur when we convert the OperA meta-model into the OWL OperA meta-model. These functionalities also contribute to our research questions.

6.5 Conclusions

In this chapter, we give the description of a case called "conference organization" and perform the specifications of the OperA model design for the case. With this case, we try to perform the evaluation of the OWL OperA meta-model. With the design of the case into OperA, we can have the basic input for the OWLperettA we need to create. We just need to make specific changes to this model for different properties' evaluation. With one changes, expectations for this changes will be produced after using OWLperettA. According to the methods, scenarios, and experiments we have designed, we can obtain actual results. After we compare the expectations with the actual results, we can conclude the evaluation. All the added functionalities that we predicted in Chapter 5 can be verified by the different methods in evaluation. That means that for research questions 1 and 2, the OWL OperA meta-model can be improved by the added functionalities that have been proven by the evaluation.

7. Discussion and Conclusions

In the previous chapters, we have presented the introduction of our project, OWLperettA, the

OWL OperA meta-model, and evaluation by case study. We regard this information as the contributions that this project brings to us. All these information presented in the above Chapters is considered as the basis for this chapter. In this chapter, firstly, we reveal the meaning of the results from each Chapter and discuss the relationships between the results and the research questions. In addition, we elaborate some of the limitations we encountered during our processes of development. Finally, some future directions for research are introduced so that others are able to embark on related work more easily, and perhaps resolve our remaining limitations. We first introduce the discussion of the results.

7.1 Discussions

As we stated in Chapter 1, we seek to answer three research questions. For each of these questions, we should discover a solution. Therefore, we will discuss the research questions separately to show how we solve each one.

Automatically checking structural inconsistencies

Our first research question concerns the need to check structural inconsistencies for the case model based on OperA. That is, can we design the OperA meta-model in another format that satisfies all the requirements of the OperA meta-model, and on the basis of this newly designed meta-model (the OWL OperA meta-model), are we capable of creating an OWL OperA case model? Finally, with regard to this case model, can we automatically check for structural consistencies when some inconsistencies occur while we are designing the case model?

In Chapter1, we selected OWL2 as the format for our new meta-model, which we call OWL OperA meta-model. The OWL OperA meta-model presented in Chapter 3 was specified to satisfy OperA approach description requirements and mapping from OperA meta-model to build a meta-model in OWL2. With this OWL OperA meta-model, we can a design OWL OperA case model.

In order to verify the OWL OperA meta-model, we should use it to build the SS of an OM case that we call the OWL OperA case model. In Chapter 6, we designed an experiment, a scenario, and a method to check the inconsistencies in the OWL OperA case model. These inconsistencies in Table6.1 appear when we edit the OWL OperA case model. According to the results of the experiment in Table 6.5, compared to Table 6.4, the inconsistencies have been automatically checked out by reasoner after we input the case model into Protégé. All of the above show that the research question 1 can be solved by using the OWL OperA meta-model to edit the OWL OperA case model.

OWL2 automatically exporting

Research question 3 is an extension of solving the other research questions. It relates to the demand to create an OWL OperA case model automatically instead of editing documents. In other word, can we automatically export the OperA case model into the OWL OperA case model?

In order to design the OWL OperA case model, we should perform the following two tasks. Firstly, we should design the OWL OperA meta-model, as we did in Chapter4. Secondly, we should edit

the case based on the OWL OperA meta-model. In accordance with Chapter3, we intended to design a tool called OWLperettA for exporting OWL2. As we introduced in Chapter2, OperettA is an existing tool for designing an OM based on the OperA meta-model. Thus, we suggest to build OWLperettA based on OperettA to make the editing of the SS of an OM easier, that is, developing the OWLperettA as a plug-in of OperettA. As Figure4.2 shows, OWLperettA has been designed to export OperA case model into an OWL OperA case model in OWL2 format. By performing these tasks, we simply begin designing the OWL OperA case model through OperettA, then by using the OWLperettA embedded in OperettA, we are able to obtain the OWL case model automatically instead of editing the case model manually. According to the Chapter6, OWLperettA is used in the method of evaluating functionalities 2, 3, and 4. We show that this tool can successfully export the OWL OperA case model. Therefore, according to these discussions, we can see that research question 3 has been worked out through the designing of the OWL OperA meta-model and OWLperettA.

Semantic inconsistencies checking

In accordance with research question2 in Chapter 1, we hope to find solutions to identify the semantic inconsistencies in the OWL OperA case model because we think the OWL OperA meta-model can help us do so. We regard it as the demands to locate individuals with the same individual names and to check individuals of Objective class that have a Sub-Objective of themselves (these two are regarded as semantic inconsistencies that cannot be treated as structural errors). These demands are divided into demand 1 and demand 2 respectively.

For demand 1, when designing the OWL OperA case model, we prefer to understand or identify different individuals with their names. However, for designing the SS of an OM through OperettA, it is not possible to do so because of the possible repetitive names of individuals. In order to check this kind of inconsistency, we should design the OWL OperA case model following the OWL OperA meta-model built in Chapter3. By using the OWLperettA described in Chapter 4, we can obtain this case model more easily. As the evaluation in Chapter 6 has presented, we evaluate demand 1 through two functionality scenarios. Through OperettA and OWLperettA, we provide some examples with same names for individuals in Tables 6.6 and 6.8. After we execute the methods offered by Chapter6, we are capable of obtaining the results in Tables 6.7 and 6.9. These tables reveal that all the individuals of the same name have been queried and located by our method. Next, we are able to modify the individuals' names because of these locations.

For demand 2, when we design the OperA case model by editing, there is a high potential to edit an objective individual that has a Sub Objective of itself when there are a lot of objective individuals. This inconsistency can be seen as semantic inconsistency and it is hard to identify. In order to check this kind of inconsistency, we should edit the OWL OperA case model by using the OWL OperA meta-model detailed in Chapter 3. As described in Chapter 6, inconsistencies are designed in Table 6.10. We evaluate these inconsistencies by using the method we have provided in Chapter 6. The results in Table 6.11 reveal that this kind of consistency can be checked successfully. The added functionality we described in Chapter 5 related to objective is proven to be true.

The discussions of demand 1 and 2 have shown that research question 3 can be handled by the development of OWLperettA and the OWL OperA meta-model.

After discussing the research questions and results, we are able to confirm the contributions described in Chapter1 for OWLperettA and the OWL OperA meta-model. With the help of evaluation, four predicted functionalities have been verified for the OWL OperA meta-model. And these functionalities prove that the three research questions have been solved.

7.2 Limitations

In our thesis, we successfully resolve the short-comings that exist in the OperA meta-model. However, not all the short-comings are resolved perfectly. There are several limitations that may exist. These limitations may not affect the effectiveness of the model and the added functionalities; but there are risks that affect the validity or feasibility of the OWL OperA meta-model. We divide the limitation into three categories: limitation during the meta-model conversion; limitations during the evaluation; and limitations during the development of OWLperettA.

Limitations during the meta-model conversion

For the meta-model, our scope may be too narrow. We focus on the social structure (SS) of an organizational model (OM). However, SS is a small part of the OperA model. There are several other parts, namely interaction structure (IS), communication structure (CS) and Norm structure (NS). Because of time limitations, we focus efforts on SS instead of all the parts. However, SS cannot fulfill all the features of the OperA approach.

Limitations during the evaluation

- ✓ Bounded of the evaluation method

A limitation also stems from evaluation of functionality 3. In the OWL OperA case model, we have four classes that we should deal with to create comparison in evaluating functionality 3. Evaluation for Functionality 3 tries to locate individuals in different classes with the same individual names. However, dozens of classes will increase for the meta-model in the future because of developing the other parts of the OM. That means the number of comparisons between individuals of different classes will add up to hundreds even. The workload for designing SPAQL query statements for each pair will be also large.

- ✓ Low Integration

During the evaluation, firstly, there are many steps in the evaluation methods. Then, while using the SPARQL query part, we must input each corresponding designed SPARQL query statement at each query. These factors make the evaluation too complicated and low integrated.

Limitations during the development of OWLperettA

- ✓ RDF and OWL exporting vague

In OWLperettA, we indeed export the documents that can be read by Protégé. However, these

documents are not well formed OWL2 documents. They are a type document based on RDF, which is also the basis of OWL2. Even though they can be directly transferred into OWL2 by inputting into Protégé and saving them, it represents low integration.

- ✓ Bounded exporting

The current OWLperettA is able to export the SS of an OM designed by OperettA. However, an OM consists of not only SS but also NS, IS, and CS. Thus, the exporting is circumscribed.

- ✓ Lack of Evaluation

Because this tool is a simple document converting tool, we have already realized the converting. However, we didn't evaluate whether the tool can successfully convert all the information included in the OperA model into the OWL model.

After analyzing the limitations of our project, we provide a summary of our thesis in the next section.

7.3 Conclusions

In our thesis, we present a meta-model called the OWL OperA meta-model for organization-oriented model developments. This OWL OperA meta-model is derived from the OperA approach description and converted from the OperA meta-model designed by EMF. The OWL OperA meta-model is able to express all features in the OperA approach description. It adds semantic and stronger constraints to the expressions to make the meta-model more powerful. On the basis of the OWL OperA meta-model, the tool OWLperettA is developed. OWLperettA is used to support OWL2 files exporting and it is embedded in OperettA. OWLperettA can convert the OperA case model into an OWL OperA case model, which is a representation of the case model built upon the OWL OperA meta-model. Combining these two consequences with evaluation, we verify that we have developed a powerful OWL OperA meta-model and OWLperettA.

Our work realized the conversion from a simple EMF-based model to OWL2. We have provided a referable mapping table for converting an EMF-based model into OWL2. We also present how to add the semantics into the SS of an OperA meta-model. We can use our work in adding semantics into an EMF-based model. We can also consider our work as the beginning of adding semantics into the complete OperA meta-model. By using an OperA meta-model that contains semantics, we can design a comprehensive OM.

7.3 Future Work

An organizational model is a relatively large model. In this thesis, we propose to transfer an OperA meta-model into OWL2 and evaluate the effectiveness of our OWL OperA meta-model. In order to evaluate it more easily, we design the OWLperettA to automatically export OWL2 ontology. Based on the information mentioned above, future research for future could concern

integrated OperA meta-model conversion, completed OWLperettA implementation, and evaluation method improvements.

Integrated OperA meta-model converting

As has been described in the limitations section, there are three sub-structures of OMs that must be converted for complete OperA meta-model conversion. We have already converted SS. The meta-model transformations from the IS, CS, and NS of an OM into OWL2 could be direction. The processes of converting are similar. Finally, converting all OperA meta-models into OWL2 will make the OWL OperA meta-model more valuable to design a complete OM.

Completed OWLperettA implementation

According to Section 7.2, firstly, the NS, the IS, and the CS may be converted in the future. In order to find an easier way to evaluate the designed ontology for them respectively, we must also modify OWLperettA. Currently, OWLperettA is able to export the SS portion. There is a demand to add functionalities in OWLperettA to export other three part (NS, IS and CS) into OWL2. Secondly, if we wish to perform the exporting more clearly and perfectly, we can change the OWL2 source package Jena to OWL API. OWL API is the source package used by Protégé. If we replace Jena with OWL API, we are able to export the format in the same manner as with the documents from Protégé.

Evaluation Method improvements

According to the limitations in Section7.2, firstly, we have indicated that our evaluation methods would cause a high workload because of the increasing number of classes if we develop other parts of OperA. Hopefully, we need to find solutions to solve the situation that there is an individual in one class, comparing its individual's name to other individuals' names with those individuals are in different classes, if some pairs have the same individual names, we should query them and obtain all the results by one query. Secondly, at present, we copy the SPARQL query statements and paste them into the SPARQL query tab of Protégé for each of the different queries. We prefer to have humanizing interfaces do the querying. We can design a plug-in of Protégé by embedding all these query statements into the program. Integrating with a simple interface, we can query the ontology by pressing one button called "name checking". We thus query individuals in all classes with the same name, and these query results can be displayed in one table window.

We have discussed the results and research questions. All research questions have been answered by the development of the OWL OperA meta-mode, and OWLperettA by evaluation of this meta-model. However, there still exist some limitations or weaknesses. Our project is just a beginning representation that adding the semantic expression into the OperA meta-model. There are many future projects that can be undertaken to complete OperA approach conversion. In the future, a complete OWL OperA meta-model that covers all aspects of the OperA approach may be developed to build a better OM.

Reference

Aldewereld, H., & Dignum, V. (2011). OperettA: Organization-oriented development environment *Languages, Methodologies, and Development Tools for Multi-Agent Systems* (pp. 1-18): Springer.

Budinsky, F. (2004). *Eclipse modeling framework: a developer's guide*: Addison-Wesley Professional.

Consortium, W. W. W. (2009). OWL 2 web ontology language document overview.

DeLoach, S. A. (2002). Modeling organizational rules in the multi-agent systems engineering methodology *Advances in Artificial Intelligence* (pp. 1-15): Springer.

Dignum, M. (2003). *A model for organizational interaction: based on agents, founded in logic*.

Dignum, V., Meyer, J.-J. C., Dignum, F., & Weigand, H. (2003). Formal specification of interaction in agent societies *Formal approaches to agent-based systems* (pp. 37-52): Springer.

Epstein, J. M. (2006). *Generative social science: Studies in agent-based computational modeling*: Princeton University Press.

Ferber, J., Gutknecht, O., & Michel, F. (2004). From agents to organizations: an organizational view of multi-agent systems *Agent-Oriented Software Engineering IV* (pp. 214-230): Springer.

Fokoue, A., Kershenbaum, A., Ma, L., Schonberg, E., & Srinivas, K. (2006). The summary abox: Cutting ontologies down to size *The Semantic Web-ISWC 2006* (pp. 343-356): Springer.

Group, W. C. O. W. (2009). {OWL} 2 Web Ontology Language Document Overview.

Horridge, M. (2009). A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition1. 2. *The University Of Manchester*.

Horridge, M., Drummond, N., Goodwin, J., Rector, A. L., Stevens, R., & Wang, H. (2006). *The Manchester OWL Syntax.* Paper presented at the OWLed.

Horridge, M., & Patel-Schneider, P. F. (2009). OWL 2 web ontology language manchester syntax. *W3C Working Group Note*.

Klyne, G., & Carroll, J. J. (2006). Resource description framework (RDF): Concepts and abstract syntax.

Knublauch, H., Fergerson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protégé OWL plugin: An open development environment for semantic web applications *The Semantic Web–ISWC 2004* (pp. 229-243): Springer.

McBride, B. (2001). *Jena: Implementing the RDF Model and Syntax Specification.* Paper presented at the SemWeb.

McGuinness, D. L., & Van Harmelen, F. (2004). OWL web ontology language overview. *W3C recommendation, 10*(2004-03), 10.

Okouya, D., & Dignum, V. (2008). *OperettA: a prototype tool for the design, analysis and development of multi-agent organizations.* Paper presented at the Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers.

Okouya, D., Penserini, L., Saudrais, S., Staikopoulos, A., Dignum, V., & Clarke, S. (2008). *Designing MAS Organisation through an Integrated MDA/Ontology Approach.* Paper presented at the TWOMD.

Prud'Hommeaux, E., & Seaborne, A. (2008). SPARQL query language for RDF. *W3C recommendation, 15*.

Rodriguez-Muro, M., Lubyte, L., & Calvanese, D. (2008). *Realizing ontology based data access: A plug-in for protégé.* Paper presented at the Data Engineering Workshop, 2008. ICDEW 2008.

IEEE 24th International Conference on.

Shearer, R., Motik, B., & Horrocks, I. (2008). *HermiT: A Highly-Efficient OWL Reasoner.* Paper presented at the OWLED.

Sirin, E., Bulka, B., & Smith, M. (2010). *Terp: Syntax for OWL-friendly SPARQL Queries.* Paper presented at the OWLED.

Sirin, E., & Parsia, B. (2007). *SPARQL-DL: SPARQL Query for OWL-DL.* Paper presented at the OWLED.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web, 5*(2), 51-53.

Tudorache, T., Noy, N. F., Tu, S., & Musen, M. A. (2008). Supporting collaborative ontology development in Protégé *The Semantic Web-ISWC 2008* (pp. 17-32): Springer.

Völter, M., Stahl, T., Bettin, J., Haase, A., & Helsen, S. (2013). *Model-driven software development: technology, engineering, management*: John Wiley & Sons.

Walter, T., Parreiras, F. S., & Staab, S. (2012). An ontology-based framework for domain-specific modeling. *Software & Systems Modeling*, 1-26.

Yin, R. K. (2009). *Case study research: Design and methods* (Vol. 5): sage.

Zambonelli, F., Jennings, N. R., & Wooldridge, M. (2001). Organisational rules as an abstraction for the analysis and design of multi-agent systems. *International Journal of Software Engineering and Knowledge Engineering, 11*(03), 303-328.

Appendix

A. OWL OperA meta-model Specification

1. OperA approach & OperA meta-model definition

Objective: Objective defined the state of affairs that the role expected to achieve or realize in the environment. Role Objective is become part of proposes for enacting the role for actor. It is a definition for role's ideal state. Roles are specified by its objective and each role at least has one objective. It may have parent-objective or sub-objective. Objective can be further described through the explanation of the sub-objective.

Objective Definition	
Name:	Identify the particular objective.
Sub-Objective:	it is objectives that contribute to the realization of another objective
Parent-objective:	it is higher level objective that this objective can only achieve a part of the goals of the higher level objective.
State:	use the LCR to express what the objective is for the environment.
User:	it is the role which contains the objective

Table1 Objective Definition

Element	EAttributes → interacting elements		EReference
Objective	stateDescription	PartialSateDescription	Name
	subObjective	Objective	
	parentObjective	Objective	
	usedByRole	Role	

Table2 OperA meta-model definition for Objective

Dependency: it is the link between roles to describe the interaction between roles. It describe how actors can interact and contribute to realization of the objective of each other by different kind of coordination. If there are roles A and B, A depends on B to realize the objective C. C is one of the objectives of A to realize. The description of the process is call dependency. Coordination type of society that is divided into three types are assigned to the dependency.

As the Table described that the Dependency is consist of 3 kind of dependencies: HierarchyDependency, MarketDependency and NetworkDependency. According to [], we can see that:

1. HierarchyDependency: these work like delegation; the dependees have to achieve the objective which is assigned to him through the dependency.

2. MarketDependency: it's like a kind of bidding or auction. The dependees can apply for some objective through the dependency. However, it's the dependants' right to choose which

dependee he prefer to fulfill the objective.

3. NetworkDependency. In this situation, all roles are equal to the objective. They have to coordinate among them to achieve the objective.

Dependency Definition	
ID:	it is a particular number to make the organizational model easier to be remembered
Name	it is a unique identifier for the dependency.
Objective:	A set of landmarks that describe the desired result of this role
Dependant:	it is the role which depend on another role through the dependency to realize the objective of it
Dependee:	it is the other role described in the upper row.

Table3 Dependency definition

Element	EAttributes → interacting elements		EReference
Dependency	ObjectiveOfDependency	Objective	Name
Sub-classes			ID
HierarchyDependency	HasDependant	Role	
	HasDependee	Role	
MarkerDependency	HasDependant	Role	
	HasDependee	Role	
NetworkDependency	HasDependant1	Role	
	HasDependant2	Role	

Table4 OperA meta-model definition for Objective

Right: it is a definition for the right of roles, it tells about the inherent capability of the role. Rights are always represented by especially format LCR.

Right Definition	
Name:	it has a unique name to be identified
Capability expression:	It is a kind of format to express the capability of a actor while it play a role

Tabel5 Right Definition

Element	EAttributes → interacting elements		EReference
Right	Expr	PartialSateDescription	Name

Table6 OperA meta-model definition for Right

RoleType: it shows that the role was divided into 2 categories. It is represented by two individuals with cardinality value. In OperA, roles can be two types called: institutional roles and external roles. The institutional roles' actors are complex and controlled by society and designed to force the social behavior and global activity. The external roles enact by actors according to the rules and principles describe the overall objectives of society by society specified rules. They are

represented by two individuals with cardinality value

RoleType	
Value:	it has two confirmed value: ‘External’ equal to 1 and ‘Internal’ equal to 0

Table7 RoleType Definition

2. Specifications of OWL OperA meta-model

Objective

The Objective and it’s relationships with each other classes can be described as the Figure below:

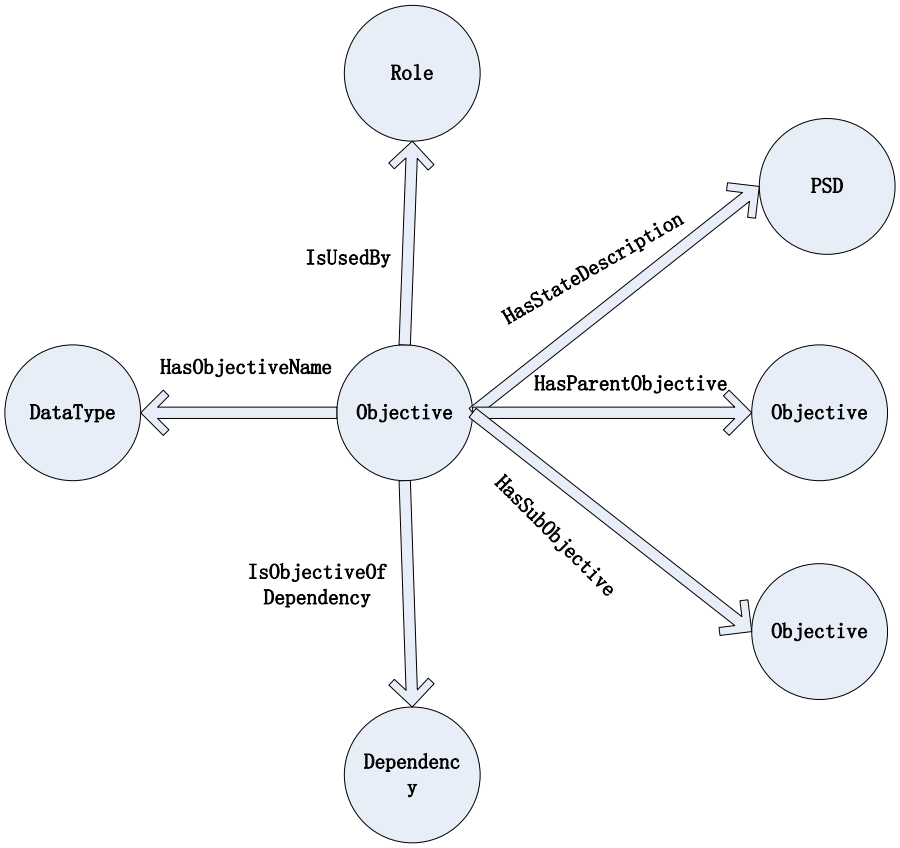


Figure4.5 Objective & Relationship

We can define the class Objective in the below Table according to our simple rules:

class	Objecitve
DataProperties	HasObjectiveName
Dataproperties Restriction	HasObjectiveName exactly 1 string
Objectproperties	HasParentObjective HasSubObjective HasStateDescription IsObjectiveOfDependency IsUsedBy
Objectproperties	HasParentObjective min 0 Objective

Restriction	HasSubObjective min 0 Objective HasStateDescription exactly 1 PSD IsObjectiveOfDependency some dependency IsUsedBy some Role

Table8 Objective definition in OWL2

Dependency

The Dependency and it's relationships with each other classes can be described as the Figure below:

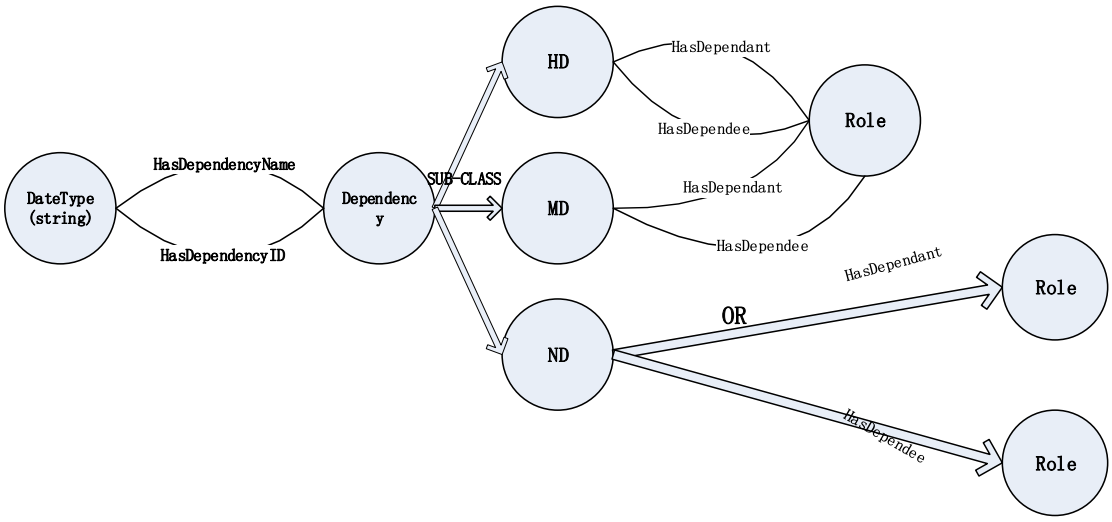


Figure4.6 Dependency & Relationship

We can define the class Dependency in the below Table according to our simple rules:

Class	Dependency		
DataProperties	HasDependencyID HasDependencyName		
Dataproperties Restriction	HasDependencyID exactly 1 string HasDependencyName exactly 1 string		
Sub classes	HD	MD	ND
Objectproperties	HasDependant HasDependee	HasDependant HasDependee	HasDependant HasDependee
Objectproperties Restriction	HasDependant exactly 1 Role HasDependee exactly 1 Role	HasDependant exactly 1 Role HasDependee exactly 1 Role	HasDependant1 exactly 1 Role or HasDependant2 exactly 1 Role

Table9 Dependency definition in OWL2

Right

The Right and it's relationships with each other classes can be described as the Figure below:

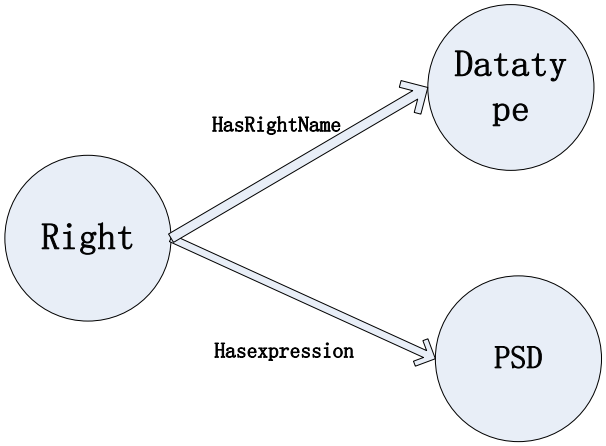


Figure4.7 Right & Relation

We can define the class Right in the below Table according to our simple rules:

class	Right
DataProperties	HasRightName
Dataproperties Restriction	HasRightName exactly 1 string
Objectproperties	HasExpr
Objectproperties Restriction	HasExpr exactly 1 PSD

Table10 Right definition in OWL2

RoleType

The class RoleType is different from the other classes, it is treated as a datatype of role. It includes two individuals external and internal with their own value to represent different things.

class	RoleType
Individuals members	External Internal
DataProperties	HasLabelValue
Dataproperties assertion	External HasLabelValue 1 Internal HasLabelValue 0
Objectproperties	Empty
Objectproperties Restriction	Empty

Table11 RoleType definition in OWL2

B. Evaluation Results

1. Screenshot of Property 1:

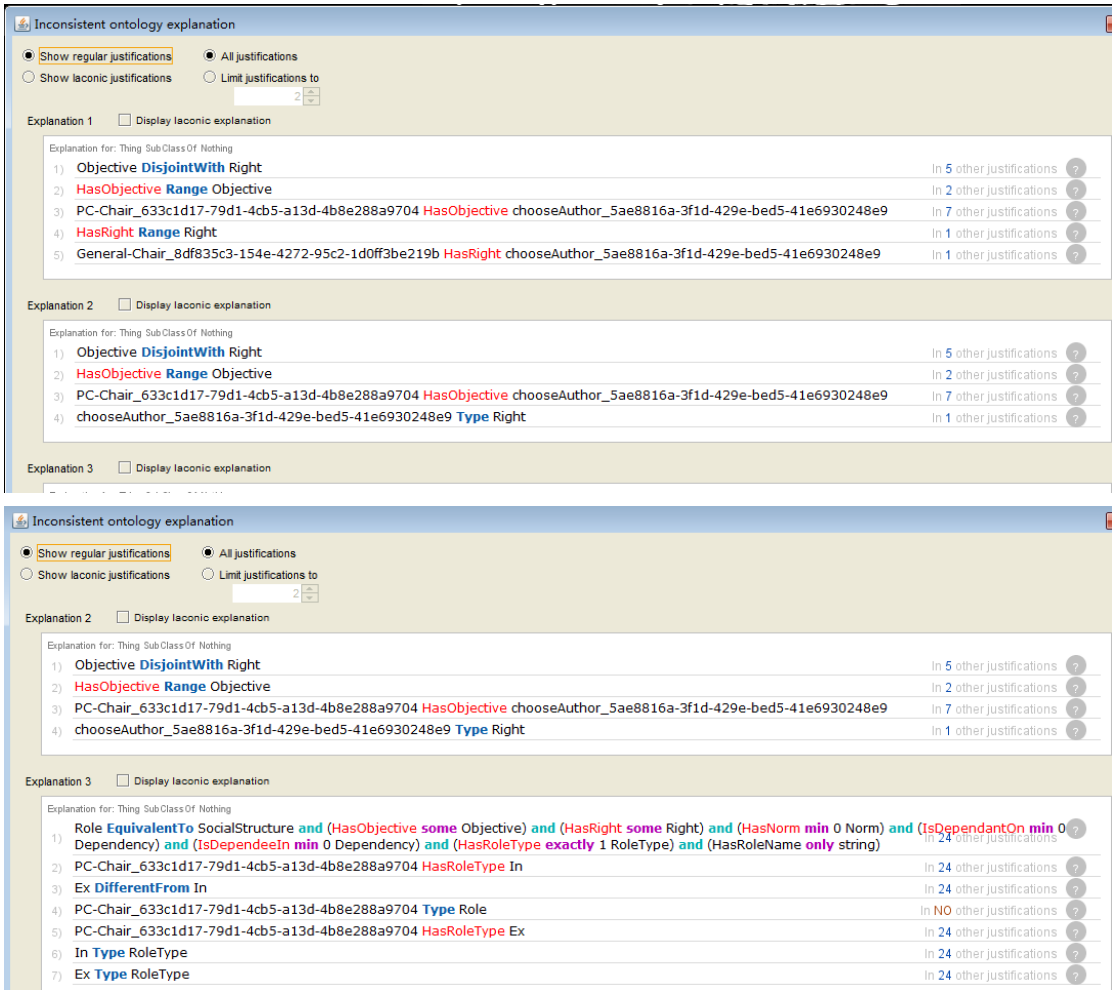


Figure1 Screenshot for the explanation

2. Screenshot of Property 2

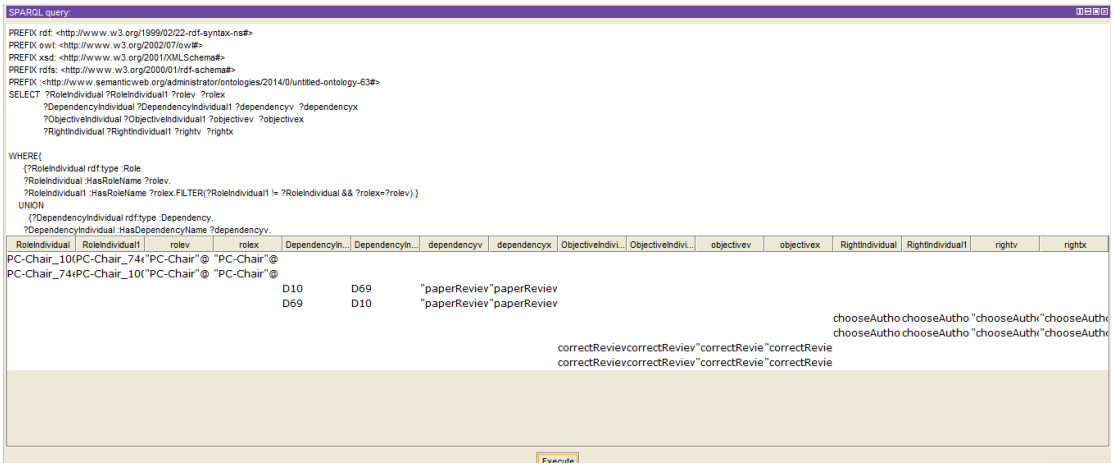


Figure2 Screenshot for all the added elements

3. Screenshot of Property 3

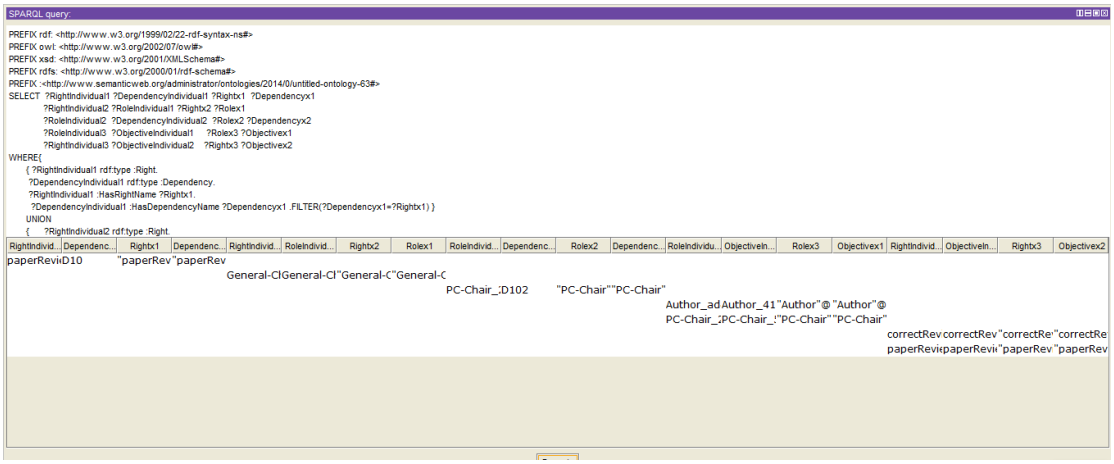
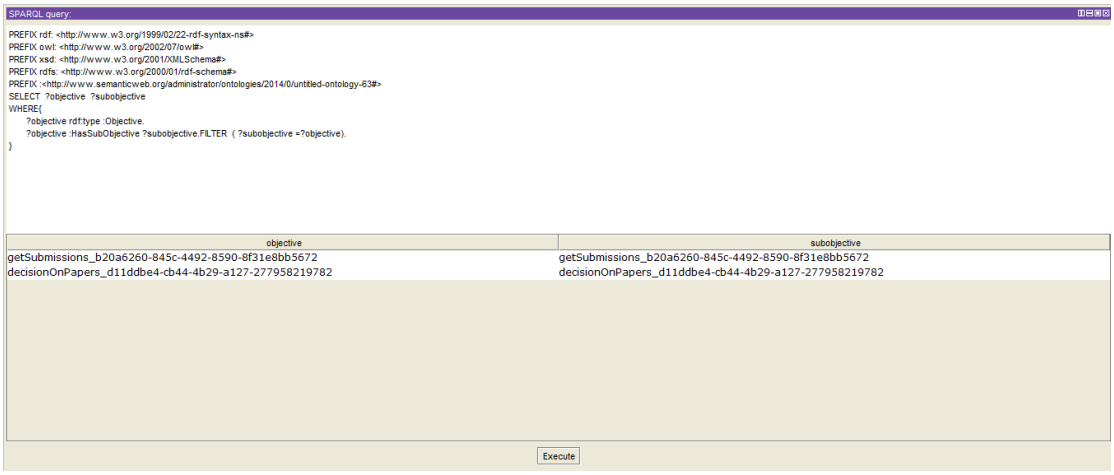


Figure3 Screenshot for all the added elements

4. Screenshot of Property 4



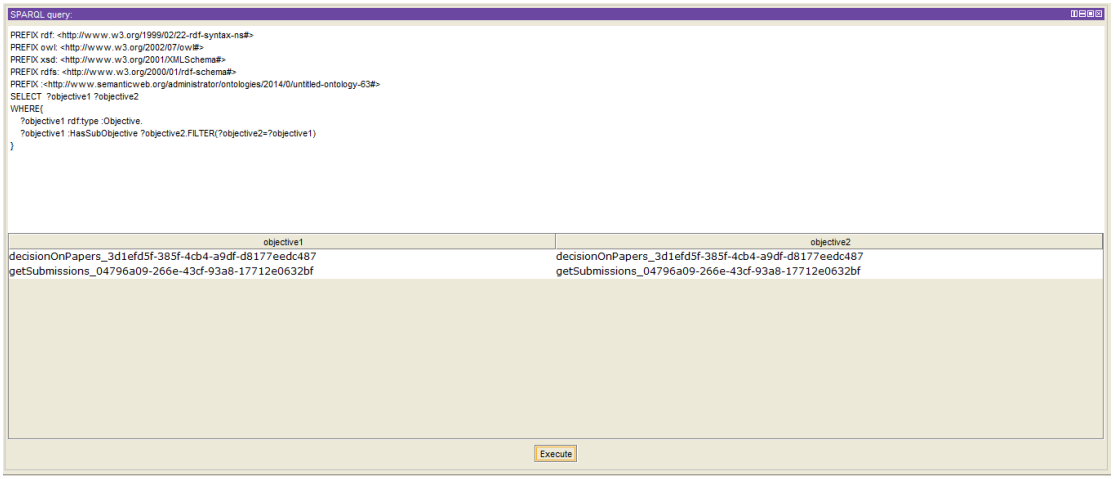


Figure4 Screenshot for objective query

C. Query Statement for the Added functionalities

1. Query statement for property2

```
PREFIX :<http://www.semanticweb.org/administrator/ontologies/2014/0/untitled-ontology-63#>
    SELECT    ?RoleIndividual ?RoleIndividual1 ?rolev    ?rolex
              ?DependencyIndividual ?DependencyIndividual1 ?dependencyv    ?dependencyx
              ?ObjectiveIndividual ?ObjectiveIndividual1 ?objectivev    ?objectivex
              ?RightIndividual ?RightIndividual1 ?rightv    ?rightx

    WHERE{
              {?RoleIndividual rdf:type :Role.
              ?RoleIndividual :HasRoleName ?rolev.
              ?RoleIndividual1 :HasRoleName ?rolex.FILTER(?RoleIndividual1 != ?RoleIndividual
    && ?rolex=?rolev).}
        UNION
              {?DependencyIndividual rdf:type :Dependency.
              ?DependencyIndividual :HasDependencyName ?dependencyv.
              ?DependencyIndividual1 :HasDependencyName ?dependencyx.FILTER(?Dependen
    cyIndividual1 !=    ?DependencyIndividual &&    ?dependencyx=?dependencyv).}
        UNION
               {?RightIndividual rdf:type :Right.
               ?RightIndividual :HasRightName ?rightv.
               ?RightIndividual1 :HasRightName ?rightx.FILTER(?RightIndividual1 != ?RightIndivi
dual &&?rightv=?rightx).}
         UNION
                {?ObjectiveIndividual    rdf:type :Objective.
                ?ObjectiveIndividual :HasObjectiveName ?objectivev.
                ?ObjectiveIndividual1  :HasObjectiveName  ?objectivex.FILTER(?ObjectiveIndivid
    ual1 !=?ObjectiveIndividual && ?objectivex=?objectivev).}
}
```

2. Query statement for functionality 3

```
PREFIX :<http://www.semanticweb.org/administrator/ontologies/2014/0/untitled-ontology-63#>
SELECT    ?RightIndividual1 ?DependencyIndividual1 ?Rightx1    ?Dependencyx1
          ?RightIndividual2 ?RoleIndividual1 ?Rightx2 ?Rolex1
          ?RoleIndividual2    ?DependencyIndividual2    ?Rolex2 ?Dependencyx2
          ?RoleIndividual3    ?ObjectiveIndividual1         ?Rolex3 ?Objectivex1
          ?RightIndividual3 ?ObjectiveIndividual2       ?Rightx3 ?Objectivex2
WHERE{
        { ?RightIndividual1 rdf:type :Right.
```

```
        ?DependencyIndividual1 rdf:type :Dependency.
        ?RightIndividual1 :HasRightName ?Rightx1.
        ?DependencyIndividual1 :HasDependencyName ?Dependencyx1 .FILTER(?Dependency
x1=?Rightx1) }
      UNION
      { ?RightIndividual2 rdf:type :Right.
        ?RoleIndividual1 rdf:type :Role.
        ?RightIndividual2 :HasRightName ?Rightx2.
         ?RoleIndividual1 :HasRoleName ?Rolex1.FILTER(?Rolex1=?Rightx2)
}
      UNION
      { ?RoleIndividual2 rdf:type :Role.
         ?DependencyIndividual2 rdf:type :Dependency.
         ?RoleIndividual2 :HasRoleName ?Rolex2.
         ?DependencyIndividual2 :HasDependencyName ?Dependencyx2.FILTER(?Dependency
x2=?Rolex2)
}
       UNION
       { ?RoleIndividual3 rdf:type :Role.
        ?ObjectiveIndividual1   rdf:type :Objective.
        ?RoleIndividual3 :HasRoleName ?Rolex3.
        ?ObjectiveIndividual1      :HasObjectiveName   ?Objectivex1   .FILTER(?Objectivex1
=?Rolex3)
}
        UNION
        {?RightIndividual3 rdf:type :Right.
        ?ObjectiveIndividual2     rdf:type :Objective.
        ?RightIndividual3 :HasRightName   ?Rightx3.
        ?ObjectiveIndividual2     :HasObjectiveName  ?Objectivex2.FILTER(?Objectivex2=  ?Rig
htx3)
}
}
```

3. Query statement for Property4:

```
PREFIX :<http://www.semanticweb.org/administrator/ontologies/2014/0/untitled-ontology-63#>
SELECT   ?objective   ?subobjective
WHERE{
      ?objective rdf:type :Objective.
      ?objective :HasSubObjective ?subobjective.FILTER   ( ?subobjective =?objective).
}
```