

STATUS OF THESIS

Title of thesis

Context-aware Modeling Using Semantic Web and Z Notation

I BAYU ERFIANTO

hereby allow my thesis to be placed at the Information Resource Center (IRC) of Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP.
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

Confidential

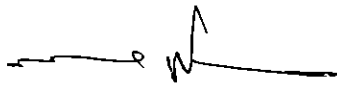
Non-confidential

If this thesis is confidential, please state the reason:

The contents of the thesis will remain confidential for _____ years.

Remarks on disclosure:

Endorsed by



BAYU ERFIANTO

Komplek Bumi Adipura IV
Jalan Pinus V No 28 Gedebage,
Bandung, Indonesia

Date: 21/01/2009



Assoc. Prof. Dr. Ahmad Kamli Bin Mahmood
Head
Computer & Information Sciences Department
Universiti Teknologi PETRONAS
ASSOC. PROF. DR. AHMAD KAMIL MAHMOOD

Universiti Teknologi PETRONAS
Bandar Seri Iskandar, Perak,
Malaysia

Date: 30/01/2009


UNIVERSITI TEKNOLOGI PETRONAS

Approval by Supervisor (s)

The undersigned certify that they have read, and recommend to The Postgraduate Studies Programme for acceptance, a thesis entitled "Context-aware Modeling Using Semantic Web and Z Notation" submitted by Bayu Erfianto for the fulfillment of the requirements for the degree of Master of Science in Information Technology.

Date

Signature :



Assoc. Prof. Dr. Ahmad Kamli Bin Mahmood
Head
Computer & Information Sciences Department
Universiti Teknologi PETRONAS

Main Supervisor :

Date :

30/01/2009

Co-Supervisor :



Abdullah Sani B. Abd. Rahman
Lecturer
Information Technology/Information Systems
Universiti Teknologi PETRONAS
Pulau Tekong, Perak Darul Ridzuan, MALAYSIA

UNIVERSITI TEKNOLOGI PETRONAS

Context-Aware Modeling Using Semantic Web and Z Notation

By

Bayu Erfianto

A THESIS

SUBMITTED TO THE POSTGRADUATE STUDIES PROGRAMME

AS A REQUIREMENT FOR THE

DEGREE OF MASTER OF SCIENCE

INFORMATION TECHNOLOGY

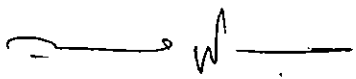
BANDAR SERI ISKANDAR,

PERAK

AUGUST, 2008

DECLARATION

I hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledge. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Signature :  _____

Name : BAYU ERFIANTO _____

Date : 29/01/2009 _____

Acknowledgement

First of all, the praise should be for Allah for all His grace and bounty, hence I could finish my study in UTP by writing up this thesis.

This work would not have been possible without the support of Dr. Ahmad Kamil and Mr. Abdullah Sani under whose supervision I finish this thesis. The gratitude expression would also be addressed to Post Graduate Studies - Universiti Teknologi PETRONAS for providing the graduate research assistantship grant and facilities during my study.

In my daily work I have been blessed with a friendly and cheerful group of fellow IT students in Postgraduate Lab room 02-02-12. I would like to extend my gratefulness to my colleagues for providing fun environment in which I could learn and grow like the normal person.

Finally, I cannot end without thanking my family, on whose constant encouragement and love I have relied throughout my time in UTP. I am grateful also to my parents, my beloved wife *Ilhamdaniah*, and my daughters *Zahra* and *Izza*. Their smile will always inspire me, in my unpredictable way. It is to them that I dedicate this work.

Abstract

Surveys in user context modeling have shown that the semantic web is one of the promising approach to represent and structure the contextual information captured from user's surrounding environment in a context-aware application. A benefit of using semantic web language is that it enables application to reason user contextual information in order to get the knowledge of user's behavior. However, regarding its notation format, semantic web is suitable for implementation level or to be consumed by application run-time.

Context-aware application is a part of distributed computing system. In distributed computing system, the language used for specification should be distinguished from the implementation / run-time purpose. This is known as separation of modeling language. Regarding the context-aware application, for those who are concerned with specification of context modeling, the language that is used for specification should also be distinguished from the implementation one.

This thesis aims at proposing the use of formal specification technique to develop a generic context ontology model of user's behavior at the Computer and Information Sciences Department, Universiti Teknologi PETRONAS. Initially, the context ontology was written in OWL semantic web language. The further process is mapping onto a formal specification language, i.e. onto Z notation. As a result, specification of context ontology and its consistency checking have been developed and verified beyond the semantic web language environment. An inconsistency of context model has been detected during the verification of Z model, which cannot be revealed by current OWL DL reasoner.

The context-aware designers might benefit from the formal specification of context ontology, where the designers could fully use formal verification technique to check the correctness of context ontology. Thus, the modeling approach in this thesis has shown that it could complement the context ontology development process, where the checking and refinement are performed beyond the semantic web reasoner.

Abstrak

Kajian terhadap pemodelan konteks pengguna menunjukkan bahawa web semantik adalah salah satu pendekatan yang mempunyai harapan untuk mewakili dan menstruktur maklumat konteks yang diambil daripada persekitaran pengguna dalam aplikasi sedar-konteks. Manfaat menggunakan bahasa web semantik ialah ianya membolehkan aplikasi untuk memikirkan maklumat kontekstual pengguna untuk mendapatkan pengetahuan mengenai kelakuan pengguna. Walaubagaimanapun, berkaitan dengan format notasinya, web semantik lebih bersesuaian untuk paras pelaksanaan atau untuk digunakan oleh aplikasi masa-lari.

Aplikasi sedar-konteks merupakan sebahagian daripada sistem pengkomputeran teragih. Dalam sistem pengkomputeran teragih, bahasa yang digunakan untuk spesifikasi harus dibezakan daripada pelaksanaan / tujuan masa-lari. Hal ini dikenal sebagai pemisahan bahasa pemodelan. Berkaitan dengan aplikasi sedar-konteks, untuk hal yang berkaitan dengan spesifikasi pemodelan konteks, bahasa yang digunakan untuk spesifikasi juga harus dibezakan dari pelaksanaannya.

Tesis ini bertujuan untuk mencadangkan penggunaan teknik spesifikasi formal untuk membangunkan model ontologi konteks generik kelakuan pengguna pada Jabatan Komputer dan Sains Maklumat, Universiti Teknologi PETRONAS. Mula-mula, ontologi konteks ditulis dalam bahasa web semantik OWL. Seterusnya adalah pemetaan terhadap bahasa spesifikasi formal, seperti notasi Z. Dan hasilnya adalah, spesifikasi ontologi konteks dan semakan kekonsistenan dibangunkan dan disahkan diluar daripada persekitaran bahasa web semantik. Ketidakkonsistenan model konteks telah dikesan semasa pengesahan model Z, yang mana ianya tidak dinampakkan oleh pemikir OWL DL sedia ada.

Pereka bentuk sedar-konteks mendapat manfaat dari spesifikasi formal ontologi konteks, dimana pereka bentuk dapat menggunakan sepenuhnya teknik pengesahan untuk menyemak ketepatan ontologi konteks. Pendekatan pemodelan dalam tesis ini menunjukkan ianya dapat melengkapi proses pembangunan ontologi konteks, dimana penyemakan dan penapisan dapat dilakukan diluar daripada pemikir web semantik.

Contents

1	Introduction	1
1.1	Research Background	3
1.1.1	State of the Art of Context-Awareness Modeling	3
1.1.2	Semantic Web and Formal Specification	3
1.2	Objectives	4
1.3	Research Questions	5
1.4	Approach	5
1.5	Scope of the Study and Limitation	8
1.6	Structure of the Thesis	8
2	Literature Review	10
2.1	Context-Aware Computing	10
2.1.1	The General Architecture	12
2.1.2	Context Modeling Issue	14
2.1.3	Related Works on Context-Aware Deployment	15
2.2	Description Logics and Semantic Web Language	16
2.2.1	Overview of Description Logics	17
2.2.2	Description Logic: Syntax and Language	17
2.2.3	OWL Semantic Web Language	21
2.2.4	OWL Semantic Web Language Tool	26
2.3	Z Formal Specification	27
2.3.1	Z Syntax and Language	27
2.3.2	Z/EVES Tool	30
2.4	Chapter Summary	31
3	Semantic Web Context Model	33
3.1	Modeling Process	33
3.2	Representing Context Ontology in DLs	34

3.2.1	Identify the concepts and develop its taxonomy	34
3.2.2	Identify the individuals belong to concept	36
3.2.3	Distinguish Role to link the concepts	37
3.2.4	Identify sub roles	39
3.2.5	Determine concept and role constraints	39
3.3	Semantic Web Model	41
3.3.1	OWL Header Definition	41
3.3.2	Semantic Web of Class Person	42
3.3.3	Semantic Web of Class Network	44
3.3.4	Semantic Web of Class Device	45
3.3.5	Semantic Web of Class Location	46
3.3.6	Semantic Web of Class Activity	47
3.3.7	Class Restriction	47
3.4	OWL Semantic Checking	48
3.4.1	Consistency checking	49
3.4.2	Concept Subsumption	51
3.4.3	Instantiation Checking	53
3.5	Chapter Summary	55
4	Z Specification of Context Model	56
4.1	Mapping Process	56
4.2	Z Syntax and Semantics (OWL-Z)	58
4.2.1	Class Description	60
4.2.2	Properties	61
4.2.3	Value Constraint	63
4.2.4	Individual	65
4.3	Mapping Context Ontology onto Z Notation	65
4.3.1	Specification of Class Person and Its Related Property	66
4.3.2	Specification of Class Device	69
4.3.3	Specification of Class Activity	70
4.3.4	Specification of Class Location	71
4.3.5	Specification of Class and Property Constraint	71
4.3.6	Specification of Individuals	73
4.4	Checking Z Specification of Context Ontology	73
4.4.1	Consistency Checking	73
4.4.2	Subsumption Checking	76
4.4.3	Instantiation Checking	77

4.5	Chapter Summary	80
5	Discussion	82
5.1	Context Development Process	82
5.2	Context Modeling Using OWL	83
5.3	Ontology Expressiveness	85
5.4	Reflection on the Proposed Method	85
5.5	Chapter Summary	87
6	Conclusion and Future Works	88
6.1	Thesis contribution	89
6.2	Future Work Directions	90
	Appendix	97
A	DLs Specification of CIS Context Model	97
A.1	Person Conceptual Model	97
A.2	Location Conceptual Model	98
A.3	Device and Network Conceptual Model	99
A.4	Activity Conceptual Model	100
A.5	Axioms of Restriction	100
A.6	Class and Role Data Type	101
B	Context-Aware Ontology Specification	102
C	OWL-Z Semantic Definition	119
D	Z Specification of Context Ontology	124
E	Screenshot of Proof Process	132

Chapter 1

Introduction

In a context-aware computing system, the term "context" is used to describe information about user's surrounding environment. Context information might be gathered from sensors and software agents and modeled by means of the available context modeling approach. Surveys in context modeling, conducted by Strang and Linnhoff-Popien [1] and Bolchini et al. [2], have shown that context-aware computing application is now fully supported by semantic web. This implies that semantic web is one of the promising modeling language to represent, structure user contextual information captured. Chen et al. [3] have developed context-aware application framework (CoBrA), which was also supported by semantic web as its user context modeling approach. Another works initiated by Xiao [4], Gu [5]-[6], and Almeida et al. [7] proposed semantic web as their contextual information model (context model) as well.

Context-aware computing is a part of distributed computing. With regards to the design in distributed computing, many works used formal specification to distinguish modeling language at specification/design level and implementation / run-time level. For example, in his work, Jensen [8] used Colored Petri Net (CPN). Another example of application of formal specification language is CSP (Communicating Sequential Process), which is discussed in [9]. The intention is to design a protocol interaction in distributed system. With regards to formal specification language, Bjøner and Henson [10], summarized that formal specification is a mathematical description about the software or hardware which is used to develop an implementation. Given such a specification, it is possible to use formal verification techniques to look at the correctness of the system being designed or realization of implementation with respect to the specification. Regarding this matter, Nissanke [11] and Bowen [12] used Z notation, and Jackson [13] used Alloy notation as formal specification language in distributed system design. Based-on description above, it is summarized that the

language used for specification/design purpose is separated from the language for the implementation level. This is also known as separation of modeling language.

As a part of research works in ontology and semantic web, formal specification is further taken into account to express ontology beyond the semantic web language. Many works have been proposed as the basis foundation of the logical transformation from semantic web onto another formal specification language. Various formal specification languages have been addressed such as Alloy [13], PVS [14], and Z Notation [13]-[15]-[16]. Once mapped onto formal specification language, their following task was dealing with checking the consistency and reasoning the ontology beyond the semantic web reasoner [14]-[17].

The fundamental issue in this thesis is to address formal specification technique to develop context ontology model and checking the correctness of context ontology beyond the semantic web reasoner. The research domains mentioned above have become a motivation to propose context ontology model by using formal specification technique. In this thesis, context ontology is describing the user's behavior in the Computer and Information Science Department (CIS) environment, Universiti Teknologi PETRONAS.

Initially, CIS context ontology is written in semantic web language format using Web Ontology Language (OWL). Once validated in OWL reasoner, this context ontology model is then mapped onto Z specification by adopting Z syntax and semantics [13]-[16]. Consistency, subsumption, and instance checking of context ontology is further demonstrated in Z environment by making use of Z/EVES, a tool for checking and proving Z specification. As a result, context ontology is expressed in Z formal specification and ontology checking are carried out beyond the semantic web language reasoner, i.e. using Z/EVES.

The context-aware designers might benefit from the formal specification of context ontology model, by which the designers could use formal verification technique to check the correctness of context ontology. Thus, it becomes a complementary approach to develop and check context ontology beyond the semantic web reasoner. During the demonstration, an undetected inconsistency of ontology model has been discovered by Z/EVES. The refinement process might be taken into account to redefine the context ontology prior to the implementation process. The Z context ontology is formally specified hence the correctness of context ontology can be guaranteed not only from the syntactical point of view, but from logical point of view as well. Another benefit of using formal specification is that it is able to specify more expressive logical constraint involved in context ontology model.

In this chapter, an introduction to the conducted research is discussed. It begins

with a research background that contains state of the art of context-aware modeling and semantic web and formal specification. An overview of problems and a proposed solution are also presented in the later section. This chapter ends by presenting the outline of the thesis.

1.1 Research Background

1.1.1 State of the Art of Context-Awareness Modeling

Context-aware computing is a part of ubiquitous computing that is collaboratively able to provide, share, and exchange relevant information (or context) from surrounding user's environment. Context-aware computing concept, which was introduced by Schilit et al. in [18], defines a computing system that was able to acquire context information.

In context-aware computing, it is also important to define what context can be captured. Further, in [19]-[20]-[21]-[22]-[23], a context information incorporates user's surrounding information, such as location information, user profile, time, user activities, existence of computing devices, execution of application and services, and physical condition of the environment.

Upon acquiring data from the user's environment, a run-time application will process such context information hence user can use it for further *reasoning purpose*. Various knowledge-representation techniques, e.g. using ontology in semantic web language, have also contributed to address those challenges, as deployed by [3]-[4]-[5]-[7]-[24]-[25]. They use ontology using semantic web language because it provides a vocabulary of concepts for describing context. The context can be defined as the semantic representation of user's real-world in a machine understandable format. The common format used is OWL, written in XML notation. Further representation and structuring of context become the challenges which are the interest of the researcher to answer in this thesis.

1.1.2 Semantic Web and Formal Specification

Semantic web language family, i.e. DAML+OIL and OWL, are actually developed based-on Description Logics (DLs) semantics. Therefore, specifying ontology in semantic web language is the *implementation of ontology model in DLs*. Though expressing ontology in DLs can be independent from the implementation concern or run-time application phase, nevertheless, the automated tools to explore (specify and

proof) DLs syntax and languages are not available yet.

Current ontology reasoners, such as Pellet and Fact++, are able to classify taxonomy of ontology and able to detect inconsistency of ontology. Unfortunately, such reasoners yet have to carry out ontology checking based-on *implementation-oriented language*, such as OWL DL, because the current DLs reasoner still rely on semantic web language, e.g. OWL DL reasoner.

Dong [13] and Wang [26] proposed formal Z notation, Alloy and PVS as the alternative ways to express ontology beyond the semantic web model. Dong in [17] and Li in [27] then continued the previous works to combine Z Notation with Alloy to design and check *Military Plan Ontology*. They previously generated *Military Plan* ontology using DAML+OIL, and then mapped this ontology onto Z notation. In their approach, Z/EVES is then used to check the consistency of their ontology to remove some trivial syntax errors. They further transformed DAML+OIL *Military Plan* ontology into Alloy. Continuing their works, Lucanu et al. [28]-[29] also came up with the institution morphism approach to prove the similarity between semantic of OWL semantic web language and logical semantic used in Z/EVES, as the common tool to check and prove Z specification.

1.2 Objectives

The aim of this thesis is to provide a methodology to develop context ontology model by addressing the formal specification technique as mentioned in the previous section. This aim can be further expanded into the following objectives.

1. Developing a context ontology model using formal specification language.
 - To represent context ontology model using DLs notation and OWL Semantic Web Language
 - To map the context ontology in semantic web onto Z formal specification (notation)
2. Checking the correctness of context ontology model (consistency, subsumption checking, and instantiation checking)
 - To carry out semantic checking of context ontology in semantic web language using semantic web reasoner
 - To carry out semantic checking of context ontology model in Z notation.

1.3 Research Questions

Several research questions are defined to assist in the fulfillment of the objectives presented in the previous section. To be able to address formal specification technique in developing context ontology, the following research questions are come up.

1. What are the requirements to represent contextual information into ontology?
2. What are the modeling processes involved to develop context ontology using formal specification language?
3. How to validate the context ontology model?

1.4 Approach

The research presented in this thesis is about conceptual work in context ontology modeling. Problems related to this have been raised in the research question presented in the previous section, and the approach to answer those research questions have been proposed as follows:

1. Context information describes relevant aspects of the user's physical environment including its computing devices. Such information can be obtained from the available computing resources, such as from sensors and software agents. The environment to be modeled in this thesis is the behavior and situation of Computer and Information Sciences Department (CIS), Universiti Teknologi PETRONAS. As described in [19]-[20]-[21], information about location, activity, and the presence of computing devices are considered as the aspects to be included into context ontology model in this thesis.

As in Strang and Linhoff-Popien [1], they classified the context modeling approaches into relational data base model, graphical model, logic-based model, mark-up scheme model, and ontology model. This thesis focuses on the use of ontology model to represent and structure contextual information. Ontology is chosen because it can represent the knowledge of the user's behavior in a hierarchical manner to be used for reasoning purpose. Since many context-aware frameworks widely support ontology using semantic web language, hence the reasoning process of contextual information could be carried out in an unambiguity manner.

2. Context information in this thesis is supposedly obtained from sensors and software agents. Such contextual information should be described in an abstraction manner, intentionally designed to be easy to understand by human. This modeling approach can be explored by using either the graphical notation to meet the requirement of context information conceptual modeling, such as described in [30]-[31]. Nevertheless, as the alternative, this thesis presents the abstraction of context information using conceptual modeling in Description Logics (DLs) notation. DLs are chosen because it is the logical foundation of semantic web. Hence, by expressing conceptual model in DLs it could be easily transformed into semantic web language (OWL format). The further detail of the context modeling approach used in this thesis is defined as illustrated in Figure 1.1. The methodology involves the following steps:

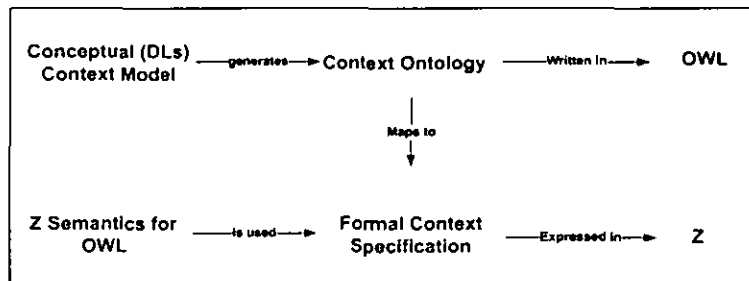


Figure 1.1: Context modeling approach used in this thesis

- Construction of conceptual context model using DLs
Before writing user context ontology in OWL notation, the conceptual model of context is initially written in DLs notation as described in [32]. DLs notations are very helpful to describe conceptual model of context ontology, which is composed of concepts, roles, and individuals. Since DLs is the logical basis of OWL, once completed writing context ontology model in DLs, it could be directly mapped onto OWL notation.
- Writing of DLs model in OWL semantic web language
Semantic web language, e.g. OWL, is the realization of DLs. Due to its feature, the OWL semantic web language of context ontology model can be directly written from DLs notation. As described in the previous section, semantic web language is actually the realization of DLs conceptual model. Therefore, once the context ontology model has been written in semantic web language, it can directly be used by the run-time application.

- Mapping of the OWL context model onto Z notation

Regarding the distributed system modeling described in the previous section, modeling language should be distinguished from the application runtime or implementation language. For example, in the purpose of specification or design, modeling language (or specification language) is not in the executable manner. Semantic web language has widely been used as the context modeling approach. However, since it can directly be instantiated or be used by the application run-time, and due to its notation format as well, in this thesis, it is considered not suitable for context modeling purpose.

Therefore, this thesis adopts the concept initiated by [16]-[28]-[29] to specify ontology beyond the semantic web language format. They defined Z syntax and semantics for each of corresponding OWL syntax. Z formal specification is a chosen language because its logical formalism is derived from set theory and first order logic, which is similar to the DLs logical foundation as well. In this thesis, the Z syntax and semantics to express OWL syntax are redefined and rewritten by directly taking from OWL semantics definition in [33].

The semantic web language consists of class constructors, properties and axioms. They were then mapped onto Z formal notation as well. Afterward, to achieve one of the objectives presented in this thesis, the context ontology model written in semantic web language are mapped onto Z formal notation by using the redefined OWL-Z syntax.

3. This thesis addresses semantics checking to evaluate the correctness of context ontology. Semantics checking covered in this thesis includes inconsistency checking, subsumption checking, and individual checking.

Pellet, as OWL DL reasoner, is used to validate the context ontology written in OWL semantic web language. *Pellet* is chosen since it has the ability to perform term checking and instantiation checking (a.k.a TBox and ABox) in a semantic web language document.

Z notation is not an implementation-oriented language (be prepared for run-time application) like OWL, instead, it is a formal specification language built on top of set theory and First Order Predicate Logic (FOL). Z features are also able to support concepts relation, role, and instantiation. Inconsistency, subsumption, and instance checking is then demonstrated in Z environment by means of

Z/EVES tool. Due to its features, Z notation has been selected to be used in this thesis. As a result, it is demonstrated that context ontology can be expressed in Z formal notation, thus, ontology checking is carried out further in Z environment, i.e. using Z/EVES tool. This shows that context ontology checking independent from OWL DL reasoner (Pellet, FACT++,Racer,etc.).

1.5 Scope of the Study and Limitation

Throughout the work and from the modeling results, some limitations of the thesis were identified. The discussion in this thesis is restricted to as follows:

1. This thesis excluded the context acquisition system, i.e. how to capture contextual information from user's surrounding environment. Due to the limitation of the context-aware and ubiquitous infrastructure in CIS department, therefore, it is assumed that all context information provided in this thesis have been captured by means of sensors and agents. The context was only limited to describe user's surrounding information in CIS Department, Universiti Teknologi PETRONAS.
2. This thesis excluded the development of context-aware application. All context ontology are defined for the verification purpose.
3. This thesis excluded the dynamic context-aware modeling such as how to model interaction system among the context-aware computing elements. However, this concern is suitable to address by using another formal specification language such as π calculus [34].

1.6 Structure of the Thesis

This thesis is organized as follows:

1. **Chapter 1: Introduction.** This chapter discusses research background, aims of the research, problem statements, solution approach and the outline of the thesis.
2. **Chapter 2: Literature Review.** This chapter briefly discusses the background of study and the state of the art in context-aware computing application, semantic web and description logics as foundation of ontology.

3. **Chapter 3: Description Logics and Semantic Web of Context Ontology.** This chapter presents the process of constructing a context ontology using OWL semantic web language. The discussion within this chapter includes a design of class (concept), properties and individuals in OWL. This chapter ends with a semantic consistency checking of the context ontology.
4. **Chapter 4: Z Specification of Context Ontology.** This chapter presents a briefly discussion on Z formal specification. The mapping process of OWL semantic web syntax and axioms onto *OWL – Z* model is further presented. Context ontology given in Chapter 3 is mapped onto Z specification. To check the correctness of the z specification, the Z typed checking has been performed, i.e. to detect typical syntax error, and use Z theorem prover perform ontology reasoning in Z/EVES.
5. **Chapter 5: Discussion.** This chapter presents the discussion on the process of developing context ontology using semantic web language and formal specification. The reflection on the proposed methods ends the discussion on this chapter
6. **Chapter 6: Conclusion.** This final chapter concludes the whole thesis highlighting the summary of contributions followed by a discussion on future and including limitation of the research work.

Chapter 2

Literature Review

The discussion in this chapter begins with the background study and the state of the art of context-aware computing and context modeling approaches. Thereafter, the overview of Description Logics (DLs) as the logical foundation of ontology and Semantic Web Language as the implementation of DLs are presented as well, which is followed by an overview of Z formal specification.

2.1 Context-Aware Computing

In computer science, the term of context-aware computing refers to the situation that computing devices can sense and react to the user environment. Computing devices may have information about the situation, where they are able to operate and based-on given rules to react accordingly. Context-awareness devices may also try to make assumptions (depending on the given deduction rule) about the user's current situation. The term context-awareness is a part of ubiquitous computing, which was introduced by Schilit [18]. They introduced distributed system from the perspective of context-aware computing . Schilit defined the term of context-aware computing as follow ([18] page 85):

"...a computer application that can adapt according to the location of user, the collection of nearby users and objects, as well as the dynamic changes of those objects in the environment..."

For example, Computer and Information Science Department at Universiti Teknologi PETRONAS in the future plan is going to deploy a context-aware meeting room. In a given scenario, the context-aware application automatically recognizes a meeting place and schedule it associates with specific agenda. To achieve this behavior,

context-aware application program will execute the rule that has been defined in ontology. Once a person enters the meeting room, by recognizing the RFID tag used by a person, hence the context-aware application may detect the presence of person, it will turn on the light, projector, microphone, and other related meeting equipment. A context-aware mobile phone may also know that it is currently in the meeting room (e.g. using position sensors to perceive the position of a user), and the mobile phone will condition its profile for a meeting scenario such as by activating vibrate mode and will reject any unimportant calls. This scenario could be possible by deploying context-aware computing application.

In context-aware computing system, the term "context" is used to describe information about user's surrounding environment. Context information is gathered from sensors and software agents which is then represented by means of the available modeling approach [1]-[30]. Abowd and Dey [19]-[35] defined context as

"...any information that can be used to characterize the situation of entities"

Research community in context-aware computing initially perceives that the term *context* is a matter of user's location, as in Dey [19]. However, in the last few years the term context has been considered not simply as a location only, but might also involves computing environment, as explained in [20]-[21]-[22]-[23]. Based-on their investigation, what aspects that might be constructed in a context are identified as follows:

1. Service and application context: context information that describes application and service currently used and run by a user, e.g. email client application, web service run, etc. Kranenburg et al. in [21] also consider context information of all properties in user's desktop that are relevant to running application, running process, display size, percentage of memory and processor usage (computing hardware context).
2. Access Network context: context information that describes all properties of available network resources, e.g. network traffic, bandwidth usage, QoS, status of connected devices, e.g. Bluetooth, WiFi, etc.
3. User profile context: is context information that typically describes about person's environment (people nearby, light, humidity), profile, task, social and spatio-temporal (outdoor and indoor position).

4. User's position context: is usually indicated by location where a person is presence. With regard to the location-awareness in context-aware computing, Domnitcheva in [36] differentiates into physical location model and geographical model. Physical location model is about the *earth coordinate system* and typically provides a magnitude in a latitude and longitude. Geographical location is about geographical objects on earth, such as countries and cities, etc. Both of location models are considered to be used in our context ontology.
5. Personal context: health, mood, schedule, and activity
6. Social context: group, activity, social relationship, and people nearby
7. Physical context: contextual information related to physical aspect of the context aware system
8. Environmental context: weather, altitude, light, etc

2.1.1 The General Architecture

In his book, Loke [20] mentions at least there are three basic functionalities exist in a context-aware computing application. Those three layers are *sensing*, *thinking* and *acting*. Sensing layer in context-aware computing comprises many sensors, for instance a position and a light intensity sensor. Those are together categorized as physical sensors which are used to capture user's physical related information.

Loke also identifies various data processing and analysis techniques considered to process context information. Those techniques involve mathematical modeling, cognitive-based models, and knowledge-based model combined with logical reasoning, and fuzzy logic. Prior to Loke with his idea of modeling and processing context information, Chen et al. [3] and Eunhoe Kim and Jaeyoung Choi [24] have also proposed a context modeling using semantic web ontology, that was identical to *knowledge bases model*.

Processing context using knowledge-based technique fully utilizes ontology written in semantic web language. Therefore, context-aware computing application can further react upon sensing and reasoning process. Actions to be taken are defined in application by means of executing rule via software APIs. As in Dey [19], context is considered in the relation of tasks (or static context model in this thesis) rather than interactions between users and application (dynamic context-aware model).

For the implementation purpose, software agents or sensors might be attached to the existing context-aware application framework. To do so, for example, an instant

messenger-like application can be made context-aware by adding agents or attaching sensors to acquire the information of awareness from user behavior. Thus, this application may deduce the information about user's position (including room name, floor and building name), who is in the room (users nearby), what are activities related to a user (he/she is away from the desktop or he/she is in meeting room), etc. By using context-aware instant messenger-like application, it enables a user to deduce current activities of a person according to his/her current location.

Figure 2.1 illustrates a general context-aware computing architecture. A client can be a mobile device, like PDA or smart phone, personal computer, or notebook. To enable context exchange among the users, context-aware computing application is required to be deployed in a client computing. The application may consist of core context-aware application (including user interface) and software agents [20].

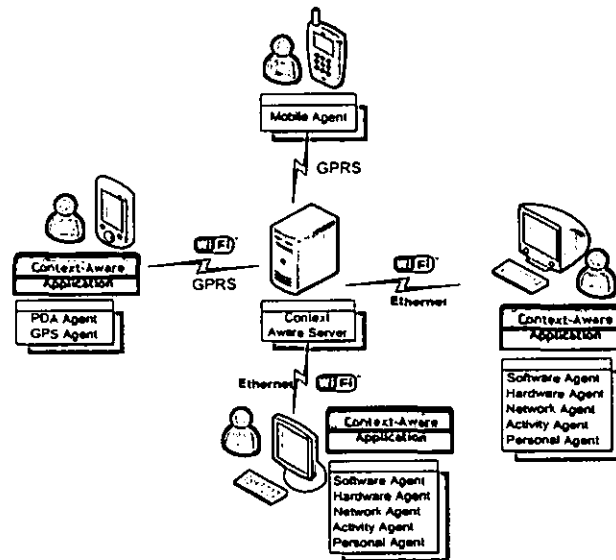


Figure 2.1: General context-aware computing architecture

The core of context-aware computing application can be like an instant messenger application as described in the previous paragraph. An agent is required to capture contextual information related to user's surrounding information. Context-aware server usually acts like a mediation server to receive information from software agents and temporarily store in the database. Mediation server can also receive and process queries from a client who wants to deduce information related to a user, such as information about current location, current activities, etc.

2.1.2 Context Modeling Issue

Upon acquiring context from sensors and software agents, the following task is how to process such user context so that it does make sense for reasoning purpose. Representing, structuring, managing and using context further become the interested challenges and many research are still underway. To address those challenges, various context modeling and representation formalisms and techniques have been proposed such in [3]-[4]-[5]-[7]-[24]-[37]. They used semantic web language ontology since semantic web provides a vocabulary for describing context-awareness and it also enable reasoning with formal logical representations.

Strang et. al. [1] classify context modeling approaches into relational data base model, graphical model, logic-based model, mark-up scheme model, and ontology semantic web model. They also denoted another modeling, i.e. object-oriented model that is intentionally developed to support web-based ubiquitous computing application. Another important thing, which is also mentioned in their findings, is the easiness to build application derived from the object-oriented model. Nevertheless, the object oriented model still lacks with logical expressiveness for context reasoning purpose, because it is not supported by logical form.

Context modeling using semantic web language, as introduced in [5]-[24]-[38], aims at overcoming the lack of formality and logical expressiveness of the previous context model. They build context model in semantic web language because it enables knowledge sharing in dynamic context-aware application, and also well-defined semantic web language model which provides a mechanism for context-aware application to reason or deduce awareness information.

The context modeling approach identified by Gu in [5]-[6] and Eunhoe Kim and Jaeyoung Choi in [24] are summarized as follows:

1. Application oriented approach: the specific application programming interface functionalities were developed for context-aware system application.
2. Model oriented approach: a conceptual model commonly used to represent the context. Many researches proposed context model based-on ER (entity relationship).
3. Ontology Oriented Approach: since OWL was introduced by W3C, many context-aware computing applications make use of OWL semantic web language as its ontology language to represent and structure context model. The context-aware application also makes use the OWL APIs to reason the information captured from the sensors and software agents.

2.1.3 Related Works on Context-Aware Deployment

In this section, some examples of works on developing of context-aware applications using semantic web context model are presented. In this thesis, the identified domain of context-aware applications are mostly deployed for smart home [24]-[38]-[39]-[40], smart office [41], smart space [22].

CONON is OWL ontology developed by Wang et al. [4]. They developed CONON, dedicated for home and office ubiquitous environment. Context in CONON was structured in semantic web ontology because the use of logical reasoning in ontology can detect inconsistency of context information using logical deduction.

Figure 2.2 shows CONON ontology presented using OWL graphical notation, which are grouped into home domain and office domain, and folded into upper and lower ontology for each particular domain. With regard to what can be a context, CONON already accommodated user context as discussed in the previous section.

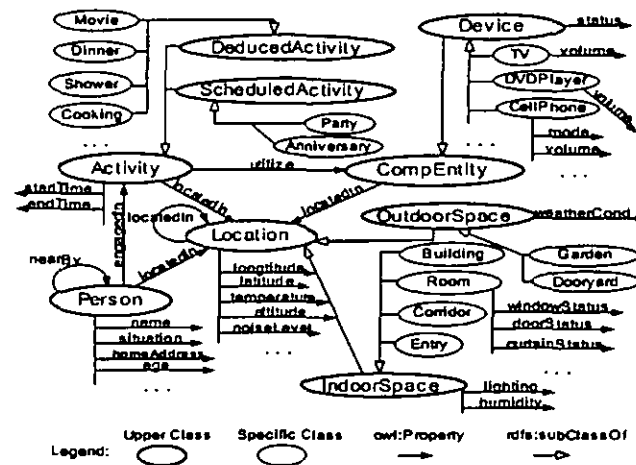


Figure 2.2: Context ontology model in CONON. This picture is taken from [4]

Chen et al. [3] proposed CoBrA infrastructure for context representation and knowledge sharing. In CoBrA, context information is shared by all devices in smart space computing application. CoBrA provides ontology written in OWL semantic web language. CoBrA architecture is illustrated in Figure 2.3. Regarding to its architecture, CoBrA has four functional components: context knowledge base, context reasoning engine, context acquisition module, and context policy management module.

The following reasons are the motivation of why CoBrA architecture makes use of semantic web as its context model.

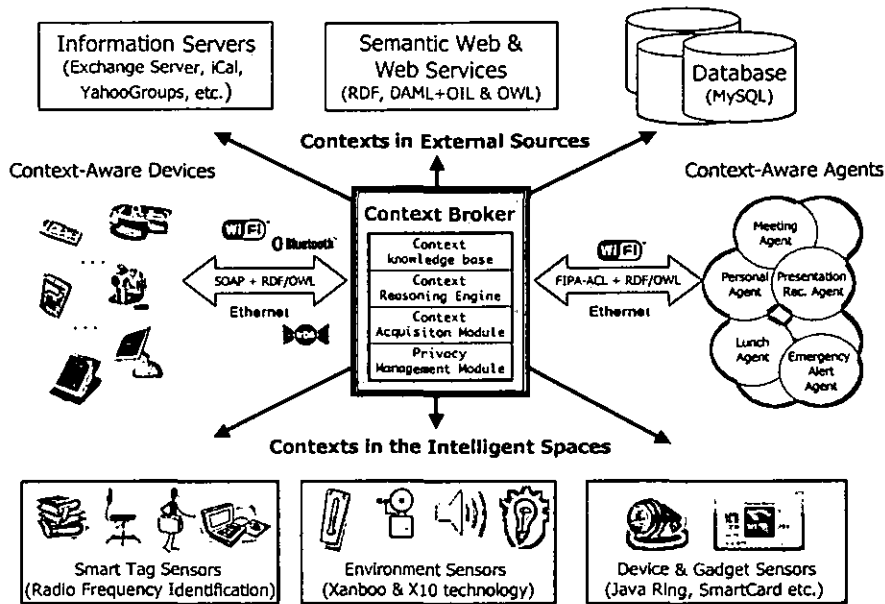


Figure 2.3: COBRA Architecture. Taken from [3]

1. Semantic web ontology provides a mean to develop context-aware computing application that is able to share context knowledge with minimum redundancy.
2. OWL as ontology is expressive enough to model contextual information ontology in CoBrA, e.g. information about person, events, devices, places, time, etc.
3. Context ontology has explicit semantics, hence they can be reasoned by current semantic web ontology reasoners to detect the inconsistency of concepts.

There are three types of reasoning purposes provided in CoBrA, i.e. reasoning with physical location ontology, reasoning with device ontology, and reasoning with temporal ontology. In CoBrA architecture, context-aware device may include device profile, device ownership relation, user temporal properties associated with device, and spatial properties of associated device.

2.2 Description Logics and Semantic Web Language

This section discusses the Description Logics (DLs), which are used as logical foundation of semantic web language. Regarding the DLs, semantic web language is the implementation of DLs. Related ontology tools are discussed as well in this section.

2.2.1 Overview of Description Logics

The term of Description Logics (DLs) refer to concept descriptions used to describe a domain and to the logic-based semantics which can be given by a translation into first-order logic. Description logic was designed as an extension to semantic networks. DLS was introduced in the 1980s as terminological systems and concept languages [42]. Today DLs have become a basis of the semantic web in the design of ontologies [43].

With regard to Baader et al.[43], DLs are designed to represent and reason about knowledge in an application domain. DLs language provides a set of constructor to build a concept (class) and role (property) description. Description language consists of distinct concept name (C), role name (R), and individual or object names (I).

Nowadays, DLs become a foundation of ontology language. In computer science, an ontology is data model that represents a set of concepts within an application domain and the relationships between those concepts [43]. Besides semantic web, ontologies are also used in artificial intelligence, software engineering, biomedical informatics and information architecture as a form of knowledge representation about the world or some part of it [43].

2.2.2 Description Logic: Syntax and Language

DLs are built on top of theoretical semantics, which are defined in term of interpretation. An Interpretation \mathcal{I} is composed of a domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$. Interpretation function also maps object or individual name $a \in I$ into an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Definition 2.1. Let $A \in C$ be an atomic concept name, $r \in R$ be a role name, C and D are the concept name. Regarding to [42], this concept and role are defined by the DLs syntax:

$$C, D \rightarrow A | \top | \perp | \neg C | C \sqcap D | C \sqcup D | \forall R. C | \exists R. C \quad (2.1)$$

where A is atomic concept, \top is top concept, \perp is bottom concept, R is an atomic relation, C and D are concepts name, \forall is universal quantifier and \exists is existential quantifier.

The family of DLs language above is known as \mathcal{ALC} , which stands for *Attributive Language with Complements*. \mathcal{ALC} has been introduced by Manfred Schmidt-Schauß and Gert Smolka in [42]. Other constructors may also include restrictions on roles such as inverse, transitivity, and functionality. The other DLs languages are extended

from \mathcal{ALC} language. To understand the relation between \mathcal{ALC} and its semantics, the examples are given as follows.

Example 2.1. Let $\{Professor, PhDStudent, AcademicStaff, FullTimeStaff\} \in C$ be concept name, $supervise \in R$ be role name, thus the constraints could be determined

$$Professor \equiv \exists supervise. PhDStudent$$

$$Professor \sqsubseteq AcademicStaff \sqcap FullTimeStaff$$

therefore, a deduction can be made such that

$$\forall supervise. PhDStudent \sqsubseteq AcademicStaff \sqcap FullTimeStaff$$

The above DLs axioms describe a situation in a University that a Professor, who has a PhD student, must be a full time academic staff accordingly. Such description is composed of concept *conjunction* (\sqcap), existential quantification $\exists R.C$. Such composition forms minimum DLs language, which is described in Definition 2.1.

To perceive the semantics of 2.1, the second example is given below.

Example 2.2. Interpretation of $\mathcal{I} = (\Delta^{\mathcal{I}}, \mathcal{I})$ is model of $\forall supervise. PhDStudent$ where the facts or individual(in capital) could be determined as follows:

$$AcademicStaff = \{ARTALE, MCGUINNESS, HAVERKORT, BAADER, SATLER\}$$

$$FullTimeStaff = \{ARTALE, HAVERKORT, BAADER, HORROCKS\}$$

$$Professor^{\mathcal{I}} = \{HAVERKORT, BAADER\}$$

$$PhDStudent^{\mathcal{I}} = \{KHATTRI, KATOEN, JEFF\}$$

$$supervise^{\mathcal{I}} = \{\langle HAVERKORT, KATOEN \rangle, \langle BAADER, JEFF \rangle\}$$

According to Definition 2.1, the individuals can be involved in the axiom:

$$\exists supervise. PhDStudent = \{HAVERKORT, BAADER, KATOEN, JEFF\}$$

The interpretation function and interpretation domain are illustrated in Figure 2.4. In that figure, individual HAVERKORT and BAADER are subset of domain $\Delta^{\mathcal{I}}$. The concept of Professor, PhDStudent, and Student are also sub set of $\Delta^{\mathcal{I}}$. The role or property *supervise* is sub set of cross function of interpretation domain $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Table 2.1 shows DLs concepts and constructors. From this table, the minimal DLs \mathcal{ALC} can be extended to form another more expressive language, e.g. with notation \mathcal{R}^+ as Transitive Role, \mathcal{I} as Inverse Role, \mathcal{Q} is Qualified cardinality restriction, \mathcal{F}

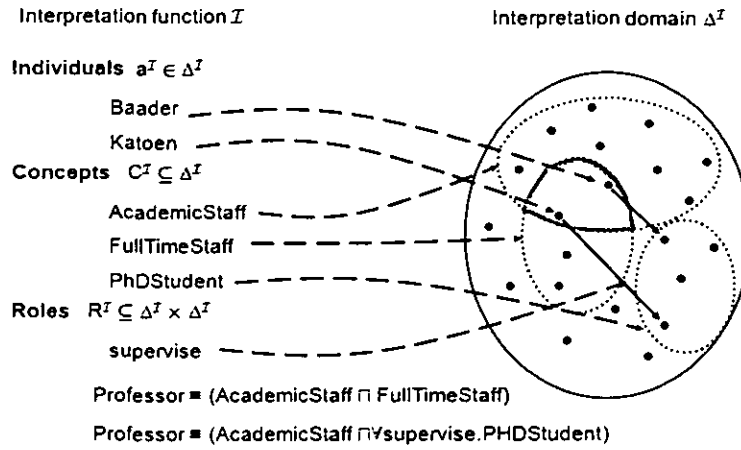


Figure 2.4: An Illustration of concept, role and individual interpretation in DLs

is Features functionality, and \mathcal{O} is Individuals enumeration. The extension of DLs determines the expressiveness of DLs language.

Typically, knowledge-base in Description Logics comes into two parts, namely *terminological concept* (TBox), i.e. knowledge about problem domain and *assertional concept* (ABox), i.e. knowledge about specific situation.

Terminological Box

Terminological Box (TBox) is set of axioms describing how concepts are related to each other in a problem domain. TBox can be built in the form of concept inclusion ($C \sqsubseteq D$), role inclusion ($R \sqsubseteq S$), concept equality $C \equiv D$ and role equality $R \equiv S$ [43]. For example, the axiom

$$\exists \text{supervise.PHDStudent} \sqsubseteq \text{Professor} \sqcup \text{Doctor}$$

determines a policy in a university that only Professor and Doctor who can supervise a PhD Student.

In TBox, interpretation \mathcal{I} satisfies $A \doteq C$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ and $A \sqsubseteq C$. Definition axioms in TBox introduces names for concept such as $A \doteq C$ and $A \sqsubseteq C$. In definition axioms, $A \doteq C$ is equivalent to $A \sqsubseteq C$ and $C \sqsubseteq A$.

Assertional Box

ABox, or Assertional Box, is set of axioms describing concrete situation of concept and role. In ABox, concept assertion is described as $a : C$, where a is an individual and C is a concept. The example of this concept assertion is *Haverkort* : *Professor* \sqcap

Table 2.1: Description Logic Concepts and Constructors, taken from [44]

Name	DLs Syntax	DL Semantics	Language
Top	\top	$\Delta^{\mathcal{I}}$	\mathcal{AL}
Bottom	\perp	\emptyset	\mathcal{AL}
Atomic Concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	\mathcal{AL}
Atomic Role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	\mathcal{AL}
Union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	\mathcal{U}
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	\mathcal{C}
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	\mathcal{AL}
Value Restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b.(a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$	\mathcal{AL}
Existential Quant	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b.(a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$	\mathcal{AL}
Unqualified number restriction	$\geq nR$ $\leq nR$ $= nR$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}}\} = n\}$	\mathcal{N}
Qualified number restriction	$\geq nR.C$ $\leq nR.C$ $= nR.C$	$\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\}$ $\{a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} = n\}$	\mathcal{Q}
Role-value map	$R \sqsubseteq S$ $R = S$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b \in R^{\mathcal{I}} \rightarrow (a, b) \in S^{\mathcal{I}}\}$ $\{a \in \Delta^{\mathcal{I}} \mid \forall b \in R^{\mathcal{I}} \rightarrow (a, b) \in S^{\mathcal{I}}\}$	
Agreement and disagreement	$u_1 \doteq u_2$ $u_1 \neq u_2$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b \in \Delta^{\mathcal{I}}. u_1^{\mathcal{I}}(a) = b = u_2^{\mathcal{I}}(a)\}$ $\{a \in \Delta^{\mathcal{I}} \mid \exists b_1, b_2 \in \Delta^{\mathcal{I}}. u_1^{\mathcal{I}}(a) = b_1 \neq b_2 = u_2^{\mathcal{I}}(a)\}$	\mathcal{F}
Nominal	I	$I \subseteq \Delta^{\mathcal{I}} \mid I = 1$	\mathcal{O}
Inverse Role	$(^{-})R$	$\{(x, y) \mid (y, x) \in R^{\mathcal{I}}\}$	\mathcal{I}
Transitive Role	$(^{+})R$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{\mathcal{I}}$	\mathcal{R}

$\forall \text{supervise. PhDStudent}$. Role assertion is described as $\langle a, b \rangle : R$. The example of this axiom is $\langle \text{Baader}, \text{Jeff} \rangle : \text{hasPhDStudent}$, which describe that BAADER supervise a PhD Student named JEFF.

In Assertional Box the interpretation \mathcal{I} satisfies $a : C$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $\langle a, b \rangle : R$ iff $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$.

Ontology Checking

In DLs, reasoning with DLs ontology is based on process of discovering implicit knowledge entailed by the ontology. Reasoning in ontology will involve the checking of the truth of statements or axioms exists in ontology.

Let \mathcal{O} is the knowledge bases in ontology, C and $D \in \Delta^{\mathcal{I}}$, and $a \in \Delta^{\mathcal{I}}I$ is individual name.

The DLs basic reasoning service provides:

1. Consistency checking. The intention is to check whether the knowledge is meaningful or not, so that ontology \mathcal{O} is consistent, thus $\mathcal{I} \models \mathcal{O}$, or concept

C is consistent, thus $C^{\mathcal{I}} \neq \emptyset$ iff $\mathcal{I} \models \mathcal{O}$

2. Subsumption checking. The intention is to check the structure of knowledge and to obtain the taxonomy of knowledge, so that $C \sqsubseteq D$ i.e. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ iff $\mathcal{I} \models \mathcal{O}$.
3. Equivalence reasoning. The intention is to check if two concepts denote the same set of instances, so that $C \equiv D$ i.e. $C^{\mathcal{I}} = D^{\mathcal{I}}$ iff $\mathcal{I} \models \mathcal{O}$
4. Instantiation reasoning. The intention is to check if individual i is instance of concept C , i.e. $i \in C^{\mathcal{I}}$ iff $\mathcal{I} \models \mathcal{O}$

2.2.3 OWL Semantic Web Language

OWL, or Web Ontology Language, is semantic web language initiated by W3C. This semantic web language provides ontology vocabularies for implementation of Description Logics. Prior to OWL, semantic web language has been introduced by the Defense Advanced Research Projects Agency (DARPA), which was known as DARPA Agent Markup Language (DAML+OIL).

OWL now becomes W3C recommendation for semantic web language model. The aim of OWL W3C semantic web language is to share the knowledge by means of web environment. Since then, OWL is widely used as a common ontology language to share information in distributed application by means of web environment, which replace the functionality of DAML+OIL. Both DAML+OIL and OWL are constructed based-on Description Logics.

OWL is split up into 3 distinct language distinguished by its logical constructors, i.e. Lite, DL, and Full. The sub language OWL Lite supports simple constructs feature that conforms to DLs (*SHIF*) family. Meanwhile, OWL DL supports all OWL Lite features with some extension on logical constructs. OWL DL conforms to DLs *SHOIN(D)* family. OWL DL fully supports DLs logical constructs, hence this languages is decidable and commonly supported by OWL DL reasoner. OWL Full sub language is meant for user who wants to express syntactic freedom of ontology specification. OWL Full supports both OWL Lite and OWL DL. However, this sub language cannot be used to reason the ontology due to the undecidable of OWL Full syntax.

With respect to ontology language in Table 2.2, DLs *SHIQ* becomes the cornerstone language for W3C Web Ontology Language. *SHIQ* is DLs extension with S + role hierarchy \mathcal{H} + inverse role \mathcal{I} + qualified number restrictions \mathcal{Q} . S is often used to describe *ACC* extended with Transitive Roles $(^+)R$.

OWL Lite extends DLs *ACC* with *Transitive* restriction on role, inverse role, and functional restriction. Thus the logical expressiveness of OWL Lite is equivalent to DLs *SHIF* (*SHIQ* extended with functional number restriction). Meanwhile, OWL DL extends *SHIQ* with nominals, i.e. *SHOIN*). As described in the previous paragraphs, additional letters indicate other extensions of DLs family (see Table 2.2).

Table 2.2: OWL Family Extensions

Symbol	Meaning	Example
\mathcal{H}	role hierarchy	$hasDaughter \sqsubseteq hasChild$
\mathcal{I}	inverse roles	$isChildOf \equiv hasChild^{-}$
\mathcal{O}	nominals/singleton classes	$Mars$
\mathcal{N}	number restrictions	$\geq 2hasChild, \leq 3hasChild$
\mathcal{Q}	qualified number restrictions	$\geq hasMother. Actrees$
\mathcal{F}	functional number restrictions	$\leq 1hasMother$

Class, Property and Individual Axioms and Description

A *Class* in OWL reflects a concept in DLs. A Class can also contains individuals or class instances. In OWL class description, there is class *owl:Thing* that superclass of all OWL class and *owl:Nothing* as inverse of *owl:Thing* (see Table 2.6). The axiom *subClassOf* is *rdfs* vocabulary to express class hierarchy in OWL. An owl class may be classified as a sub class of another class.

As described in the previous section, DLs falls into two parts, namely TBox and ABox. TBox consists of a number of class axioms (see Table 2.3) and property axioms (see Table 2.4); meanwhile ABox consists of a number of individual assertions (see Table 2.5). In Table 2.3, Table 2.4, and Table 2.5, letters *C, D* refer to class, *T* refers to a *concrete* data type, whereas *R* refers to an object property, *U* refers to data type property; *P* refers to an object or data type property, *o* and *t* refer to object and concrete values.

A class axiom in the TBox consists of two class descriptions, separated with the GCI (General Class Inclusion, or class subsumption \sqsubseteq) symbol or the equivalence symbol (\equiv), which is equivalent to GCI in both direction (i.e. $C \sqsubseteq D$ equivalent to $D \sqsubseteq C$).

Like in DLs, a property in OWL semantic web language is used to state:

1. Relationship between class instances, this relation refers to *owl:ObjectProperty*.
2. Between class or instance of class with instance data type, and this second relation is *owl:DatatypeProperty*

Table 2.3: OWL DL Class Axioms, taken from [44]

OWL Abstracts Syntax	DL Syntax	Example
<i>subClassOf</i> (C_1, C_2)	$C_1 \sqsubseteq C_2$	Human \sqsubseteq Animal
<i>equivalentClass</i> ($C_1 \dots C_i$)	$C_1 \equiv \dots \equiv C_i$	Man \equiv Human \sqcap Male
<i>disjointWith</i> ($C_1 \dots C_i$)	$C_j \sqcap C_n \sqsubseteq \perp$	Male $\sqsubseteq \neg$ Female
<i>enumeratedClass</i> ($A o_1 \dots o_n$)	$A \equiv o_1, \dots, o_n$	Animal \equiv Cat, Dog, Bear

Table 2.4: OWL DL Property Axioms. Taken from [44]

OWL Axioms	DL Syntax	Example
<i>subPropertyOf</i> (P_1, P_2)	$P_1 \sqsubseteq P_2$	<i>hasDaughter</i> \sqsubseteq <i>hasChild</i>
<i>equivalentPropertyOf</i> ($P_1 \dots P_i$)	$P_1 \equiv \dots \equiv P_i$	<i>hasCost</i> \equiv <i>hasPrice</i>
ObjectProperty (R <i>super</i> (R_1)... <i>super</i> (R_n) [<i>inverseOf</i> (R_o)] <i>domain</i> (C_1)... <i>domain</i> (C_n) <i>range</i> (C_1)... <i>range</i> (C_n) [<i>Symmetric</i>] [<i>Functional</i>] [<i>InverseFunctional</i>] [<i>Transitive</i>])	$R \sqsubseteq R_n$ $R \equiv R_o^-$ $T \sqsubseteq \forall R^- . C_i$ $T \sqsubseteq \forall R . C_i$ $R \equiv R^-$ $T \sqsubseteq \leq 1R$ $T \sqsubseteq \leq 1R^-$ R^+	<i>hasChild</i> \equiv <i>hasParent</i> ⁻ $T \sqsubseteq \leq 1$ <i>hasMother</i> $T \sqsubseteq \leq 1$ <i>hasChild</i> ⁻ <i>ancestor</i> ⁺ \sqsubseteq <i>ancestor</i>
Datatype(T) DatatypeProperty (U <i>super</i> (U_1)... <i>super</i> (U_n) <i>domain</i> (C_1)... <i>domain</i> (C_n) <i>range</i> (C_1)... <i>range</i> (C_n) [<i>Functional</i>])	$U \sqsubseteq R_i$ $T \sqsubseteq \forall U^- . T_i$ $T \sqsubseteq \forall U . T_i$ $T \sqsubseteq \leq 1U$	XSD $T \sqsubseteq \leq 1$ <i>hasName</i>

A property P is said to be *Transitive* such that $P(x,y)$ and $P(y,z)$ implies $P(x,z)$. A property is said to be symmetric property such that $P(x,y)$ iff $P(y,x)$. P is functional property such that $P(x,y)$ and $P(x,z) \Rightarrow y = z$. P is inverse functional property such that $P(y,x)$ and $P(z,x) \Rightarrow y = z$. Similarly with Class axioms, property axioms consists of a two property names, separated with subsumption \sqsubseteq or the equivalence (\equiv) symbol.

In DLs, the abstract and concrete properties are distinguished by describing the range of the property, i.e. is abstract or concrete. OWL DL reflects this distinction by using object properties and datatype properties, where an object property may only have a class description as its range and a data type property may only have a datatype as its range. Class descriptions and data type are disjoint each other.

A description in the TBox is either a named class (A), an enumeration ($o_1 \dots o_n$), a property restriction ($\exists R.D, \forall R.D, \exists R.o, \geq nR, \leq nR$, analogously for datatype property restrictions), or an intersection ($C \sqcap D$), union ($C \sqcup D$) or complement ($\neg C$) of such descriptions (see Table 2.6). Individual assertions in the ABox are either class membership ($o \in C_i$), property value ($\langle o_1, o_2 \rangle \in R_i, h o_1, o_{1,1} \in U_i$), or individual

Table 2.5: OWL DL Individual Assertion

OWL Abstract Syntax	DL Syntax	Example
Individual (o $type(C_1 \dots type(C_n))$) $value(R_1(o_1)) \dots value(R_m(o_m))$ $value(U_1(t_1)) \dots value(U_m(t_m))$ $SameIndividual(o_1 \dots o_n)$ $DifferentIndividual(o_1 \dots o_n)$	$o \in C_i$ $\langle o, o_i \rangle \in Q_i$ $\langle o, t_i \rangle \in U_i$ $o_1 = \dots = o_n$ $o_1 \neq \dots \neq o_n$	$God \equiv Great_Creator$ $Zubair \neq Ackerman$

Table 2.6: Description in OWL DL *SHOIN*, taken from [44]

OWL Abstract Syntax	DL Syntax	Example
A(URI Reference) owl:Thing owl:Nothing	A T \perp	
$intersectionOf(C_1 \dots C_n)$ $unionOf(C_1 \dots C_n)$ $complementOf(C)$ $oneOf(o_1 \dots o_n)$	$C_1 \sqcap \dots \sqcap C_n$ $C_1 \sqcup \dots \sqcup C_n$ $\neg C$ $o_1 \dots o_n$	$\neg Male$ $john, zubair, dalton$
$restriction(RallValuesFrom(C))$ $restriction(RsomeValuesFrom(C))$ $restriction(Rvalue(o))$ $restriction(UmaxCardinality(n))$ $restriction(UminCardinality(n))$	$\forall R.C$ $\exists R.C$ $\exists R.o$ $\leq nR$ $\geq nR$	$\forall hasStudent.Teacher$ $\exists hasStudent.Professor$ $\forall hasStudent.JOHN$ $\leq 1hasStudent$ $\geq 3hasStudent$
$restriction(UallValuesFrom(T))$ $restriction(UsomeValuesFrom(T))$ $restriction(Uvalue(t))$ $restriction(UmaxCardinality(n))$ $restriction(UminCardinality(n))$	$\forall U.T$ $\exists R.T$ $\exists R.o$ $\leq nU$ $\geq nU$	$\forall hasName.BOB$ $\exists hasStudent.BABA$ $\forall hasStudent.JOHN$ $\leq 1hasStudent$ $\geq 3hasStudent$

(in)equality ($o_1 = o_2, o_1 \neq o_2$) assertions (see Table 2.5).

OWL semantic web language is written in XML format. Such that, it contains header that must be declared first. OWL header consists of name space definitions. Name space indicates the identifiers of what specific vocabularies are being used in semantic web ontology. In the example, the built in OWL W3C *namespace*, namely *owl*, *rdf*, *rdfs*, and *xsd* must be declared. Further, the specific name space for our semantic web ontology model are defined as well. In the following example, the specific name space is declared as *prf*.

OWL Headers

```
<!-- Ontology Information -->
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY prf "prf#">
```

```

<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="prf"
  xmlns:owl="#owl;"
  xmlns:prf="#prf;"
  xmlns:rdf="#rdf;"
  xmlns:rdfs="#rdfs;">
  <owl:Ontology rdf:about=""/>

```

The OWL header must be followed by ontology declaration. In the previous example 4 classes have been declared: *Professor*, *PhDStudent*, *FulltimeStaff*, and *AcademicStaff*. Class *Professor* represents academic staff that supervise some PhD students. Class *AcademicStaff* represents a person (or individual) who works as academician, while class *FullTimeStaff* is for full time staff who are non academician. Those above description are written in OWL semantic web as follows.

Classes Definition

```

<owl:Class rdf:about="#AcademicStaff"/>
<owl:Class rdf:about="#FullTimeStaff"/>
<owl:Class rdf:about="#PhDStudent"/>
<owl:Class rdf:about="#Professor">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#supervise"/>
      <owl:someValuesFrom rdf:resource="#PhDStudent"/>
    </owl:Restriction>
  </owl:equivalentClass>
  <owl:intersectionOf rdf:parseType="Collection">
    <rdf:Description rdf:about="#AcademicStaff"/>
    <rdf:Description rdf:about="#FullTimeStaff"/>
  </owl:intersectionOf>
</owl:Class>

```

Object Properties Definition

Relation between concept or class with other class is defined by OWL built in Object property, i.e. *owl* : *ObjectProperty*. In the previous example, a given object property is declared as *supervise*. This object property is determined by its domain and range, which restrict the source and destination of object property. Domain and range of a *owl* : *supervise* object property is defined using *rdfs* (Resource Description Format Schema) name space, defined as follows.

```

<owl:ObjectProperty rdf:about="#supervise">
  <rdfs:domain rdf:resource="#Professor"/>
  <rdfs:range rdf:resource="#PhDStudent"/>
</owl:ObjectProperty>

```

Instances Definition

Instance in OWL reflects with the individuals which are the member of a class. In the previous example, the name of Professors with the name of PhD Students are linked. The name of Professor, full time staff, academic staff, and PhD student are defined as individuals, and declared in OWL semantic web as follows.

```

<prf:PhDStudent rdf:about="#Katoen"/>
<prf:AcademicStaff rdf:about="#Baader">
  <rdf:type rdf:resource="#FullTimeStaff"/>
</prf:AcademicStaff>
<prf:PhDStudent rdf:about="#Kahttri"/>
<prf:FullTimeStaff rdf:about="#Faizal"/>
<prf:AcademicStaff rdf:about="#Baader">
  <rdf:type rdf:resource="#FullTimeStaff"/>
</prf:AcademicStaff>
<prf:FullTimeStaff rdf:about="#James"/>
<prf:AcademicStaff rdf:about="#Satler">
  <rdf:type rdf:resource="#FullTimeStaff"/>
</prf:AcademicStaff>
<prf:FullTimeStaff rdf:about="#Nancy"/>
<prf:PhDStudent rdf:about="#Jeff"/>
<prf:PhDStudent rdf:about="#Ochman"/>
<prf:FullTimeStaff rdf:about="#Stacy"/>

```

2.2.4 OWL Semantic Web Language Tool

OWL semantic web language tools are distinct into editor and reasoners [45]. Various OWL tools have been developed to support features such as composing ontology, management, merging, reasoning, and checking [46]-[47]. In the rest of this section, briefly introduction of semantic web tools that are used in this research are discussed.

The core reasoning in DLs are concepts satisfiability, concept subsumption, and instantiation [48]-[49]-[50]. Those DLs core reasoning is used as the basis of OWL semantic web language core ontology reasoning. Many tools are available to carry out semantic web ontology core reasoning through a DLs reasoner application, such as discussed in [46]-[51]-[52].

FaCT++ (Fast Classification of Terminologies) is the implementation of description logics reasoner developed at University of Manchester. FaCT++ supports concept subsumption and satisfiability checking [53]. However, this tool only supports TBox checking and reasoning, and has no support for individual level reasoning (ABox reasoning) [52]. Currently FaCT supports both DAML+OIL and OWL semantic web language.

RACER (Renamed ABox and Concept Expression Reasoner) [54] is an commercial DLs reasoner and support DLs $ALCQHIR + (D)$. It has a much richer set

of functionalities than FaCT++ has, including ontology creation, query, retrieval and evaluation, knowledge base conversion to DAML+OIL/OWL.

Pellet [55] is also free software for ontology reasoner. It has more features than FaCT++. Pellet can be used to check and reason ontology either in TBox or ABox [52]. This DLs reasoner can be connected to many ontology editors, such as Protege [56] and SWOOP [57]. Pellet is able to check ontologies with various DLs language such as $SHI(\mathcal{D})$, $SHOIN(\mathcal{D})$, and $SHOIQ$. In this thesis, SWOOP and Pellet reasoner are used to evaluate and reason the context ontology written in OWL format.

2.3 Z Formal Specification

The Z notation (formally pronounced zed) is a formal specification language used for describing and modeling computing systems. "*It is targeted at the clear specification of computer programs and the formulation of proofs about the intended program behavior*" [12]. Z is a formal specification language which is based on ZF set theory and first-order predicate logic [12]-[58]. Z contains a standardized mathematical toolkit of commonly used logical (mathematical) functions and predicates. Expressing system specification in Z is to describe what a system does. The way of specifying system in Z can be distinguished from another specification language, such as imperative programming and functional programming language. Imperative programming pays attention on how it does, while all functional programming concentrate on how the outcome is to be achieved [12]. Both imperative and functional programming language are executable [12]-[58].

2.3.1 Z Syntax and Language

Z is not a programming language. In Z, a name must be declared before it is referenced. Properties of systems are stated using Z predicates. Hence, declarations and predicates form Z specifications.

Z Declaration

The basic form of Z declarations is $x : A$, where x is the introduced variable of the free type A . This type A , however, should be defined previously. In Z, a variable can be declared either as global or local. A global variable can be used by Z specification from the point of declaration to the end of specification. For more details are provided in Spivey [59].

Predicates in Z

Predicates in Z are Boolean-valued. Z predicates can be the forms of:

Equality and Set Membership

Basic predicates in Z notation are equalities, which is denoted by $=$ and membership relationships, which is denoted by \in . For example, the predicate $p \in \mathbb{N}$ states that variable p is a member of natural numbers \mathbb{N} .

In Z, a set relationship operator such as subset (\subseteq) can be derived using set membership. In general, the subset relationship $A \subseteq B$ can be expressed as $A \in \mathbb{P} B$ [59], where \mathbb{P} is the power set symbol. The expression $\mathbb{P} B$ denotes all the sets that are subsets of B .

Propositional Operators

These include propositional logic connectives, i.e. \neg , \wedge , \vee , \Rightarrow , and \Leftrightarrow . Logical connectives are used to connect simpler predicates to construct more complex predicates.

Quantifier

Z language also defines quantifiers in predicates, like in first order logic. These include the universal quantifier \forall , the existential quantifier \exists and the unique existential quantifier \exists_1 .

Z Language Constructs

Z also defines language constructs. These include basic type definition, axiomatic box, schematic box, constraints, theorems and proofs.

Basic Type Definition

This language construct introduces uninterpreted basic types, which are treated as sets in Z. For example:

[Identity]

introduces a given type of *Identity*, which are a set.

Axiomatic Definition

An axiomatic definition is used to define global variables, and optionally constrains their values using predicates. These global variables cannot be globally reused.

For example, the following axiomatic definition declares two variables *Name* and *Address* as subsets of *Identity*. Furthermore, these two sets are also defined mutually disjoint, which means that their intersection is an empty set. By using Z axiomatic definition, such variables could be defined as follows.

$$\left. \begin{array}{l} \textit{Name} : \mathbb{P} \textit{Identity} \\ \textit{Address} : \mathbb{P} \textit{Identity} \end{array} \right| \textit{Name} \cap \textit{Address} = \emptyset$$

Generic Axiomatic Definition

A generic axiomatic definition is a generic form of axiomatic definition, parameterized by a parameter.

The formal generic parameters are local to the definition, and each variable introduced by the declaration becomes a global generic constant. These identifiers must not previously have been defined as global variables or generic constants, and their scope extends from here to the end of the specification. The predicates must determine the values of the constants uniquely for each value of the formal parameters.

$$\left[\textit{XSD} \right] \left. \begin{array}{l} \textit{gatewayNumber}, \textit{proxyNumber} : \textit{DatatypeProperty} \\ \textit{gatewayIP}, \textit{proxyIP} : \mathbb{P} \textit{XSD} \\ \textit{domain}(\textit{gatewayNumber}) = \textit{Gateway} \\ \textit{rangeD}(\textit{gatewayNumber}) = \textit{gatewayIP} \\ \textit{domain}(\textit{Proxy}) = \textit{proxyNumber} \\ \textit{rangeD}(\textit{proxyNumber}) = \textit{proxyIP} \end{array} \right|$$

In the above generic axiomatic definition, *gatewayNumber* and *proxyNumber* are defined with a type of *DatatypeProperty*, while *gatewayIP* and *proxyIP* as a type of *XSD*.

2.3.2 Z/EVES Tool

In this research, Z/EVES tool is used to evaluate the correctness of Z specification. It is a common automated prover that provides integrated interface for composing, checking, and analyzing Z specification. Z /EVES supports syntax checking, type checking in structured specification (using schema), and general theorem proving [60]. Z/EVES supports editing of Z specification in \LaTeX format and GUI interface as well. In Z/EVES, properties about a specification can be specified as theorems. These prop-

```

Z/EVES (Z/LaTeX mode)
allValuesFrom\#declaration, subPropertyOf\#declaration, select\_2\_1,
select\_2\_2, Transitive\#declaration, WiFi\#declaration,
AccessPoint\#declaration, Desktop\#declaration, Room\#declaration,
Server\#declaration, Network\#declaration, Activity\#declaration,
Intranet\#declaration, Internet\#declaration, Software\#declaration,
Device\#declaration, range\#declaration, Person\#declaration,
Class\#declaration, Property\#declaration, fun\_type, domain\#declaration,
'Adom\#declaration', ObjectProperty\#declaration, '[internal items]' to ...
true
Proving gives ...
true
Beginning proof of ...
      allValuesFrom (Person, currentActivity) = Planned ==
Niplies allValuesFrom (Person, currentActivity) = Deduced
Assuming PlannedRule generates ...
      (Deduced, Planned) \nin disjointWith ==
Niplies allValuesFrom (Person, currentActivity) = Planned ==
Substituting allValuesFrom (Person, currentActivity) = Planned produces ...
      (Deduced, Planned) \nin disjointWith ==
Niplies Planned = Deduced
Which simplifies
forward chaining using KnownKeaber\#declarationPart, knownKeaber,
'[internal items]'
with the assumptions currentActivity\#declaration, Person\#declaration,
allValuesFrom\#declaration, PersonRunningBrowser, PersonConnectedToInternet,
PersonConnectedToIntranet, PersonRunningOffice, PersonCurrentActivityIsPlanned,
PersonRunningIM, PersonRunningEmail, PersonUseDevice, PersonLocatedIn,
disjointWith\#declaration, select\_2\_1, select\_2\_2, Planned\#declaration,
Deduced\#declaration, PlannedRule, '[internal items]' to ...
Planned = Deduced
Proving gives ...
Planned = Deduced
Beginning proof of ...
NOVEL \nin instances Server
Which simplifies
forward chaining using KnownKeaber\#declarationPart, knownKeaber,
'[internal items]'
with the assumptions Server\#declaration, instances\#declaration,
NOVEL\#declaration, ServerInstance, '[internal items]' to ...
true

```

Figure 2.5: Proofing Process Using Z/EVES (Z/LaTeX Mode)

erties include facts and expected facts that are to be facts. By proving theorems of a particular specification, the confidence about its correctness can be gained. To prove the specification, Z/EVES provides general commands to use, described as follows (take from Z Reference Manual).

Proof Command: Simplification

The simplifications performed by the simplify command are equality, integer, and predicate calculus reasoning, together with tautology checking. Simplification is affected by *grules* and *frules* whenever their hypothesis matches a sub-formula.

The conclusion of these lemmas are then included as assumptions. Simplification offers the user the opportunity to perform direct proofs because it allows the smallest

of the transformations.

Proof command: Rewriting

Rewriting is given by the `rewrite` command. It performs simplifications together with automatic application of enabled rewriting rules that matches any sub-formula.

For example, $e \in \{x : T \mid x \subseteq f(x)\}$ is rewritten as $e \in T \wedge e \subseteq f(e)$.

Proof Command: Reduction

Reduction is the most complex transformation scheme and is given by the `reduce` command. It performs rewriting together with further clever, but simple deduction schemes. This leads to the biggest step on the transformation of formula with the worst performance. In fact reduction is more than simply expansion together with rewriting. It recursively performs these activities until the formula stops changing.

Proof Command: Prove by Reduce

There two commands that implicitly combine tactics. They are *prove by reduce* and *prove by rewrite*. Both commands can also be written as *prove*. They repeatedly apply tactics on the formula until no effect is observed.

2.4 Chapter Summary

In this chapter, first of all, the state of the art of context-aware computing are discussed. Many works have contributed to this research domain, including context modeling, context acquisition, and the deployment of context-aware computing application. One of the promising model is using ontology in semantic web format.

The merits of using semantic web model is that it provides a mechanism to reason the information structured in the context model. Therefore, context-aware application can sense and react based-on the reasoning process which is supported by the logical form (DLs). Another feature is that semantic web provides vocabulary to describe the DLs conceptual model using XML format. Regarding the XML notation, semantic web language could be categorized as an executable language during the application run-time. From the reasoning point of view, some DLs reasoners also still rely on semantic web language instead of on DLs syntax (with mathematical symbol) it self.

Since context-aware is a part of distributed system, designing and specification of a context model must consider a language that is not executable at design or specifi-

cation level. Thus, semantic web language still lacks of formality, due to its notation that could not express more expressive logical constraint. Therefore, researchers have proposed another way to express ontology beyond the semantic web language, hence the consistency of ontology can be verified independently from the such executable notation. Z notation, Alloy, PVS are the formal specification language which are proposed to specify ontology. As the consequence, consistency of ontology will be verify beyond the semantic web reasoners.

In the next chapter, the development of CIS context ontology will be presented. First of all, the ontology is specified in DLs notation. Once completed, mapping of context ontology from DLs notation onto OWL semantic web language is take place

Chapter 3

Semantic Web Context Model

This chapter presents the development of context ontology. Context ontology is firstly specified in DLs notation. Thereafter, the generation of context ontology from DLs notation into OWL semantic web language is discussed. Semantic consistency checking is further carried out to detect inconsistency, subsumption checking, and instantiation checking. This ends up the discussion in this chapter.

3.1 Modeling Process

In this section, the main steps for developing context ontology is presented. During the requirement step, the behavior to model context ontology is also identified. As mentioned in Chapter 1, the intention of this section is to model the behavior of CIS Department environment, at Universiti Teknologi PETRONAS. The remainings of the thesis will use the term "CIS context ontology" to refer to the ontology of CIS contextual information.

Capturing information about context, such as information about user's profile, activities, location, and computing device are still fundamental entity to be included in the context ontology. Further in the implementation, sensors and software agents are used to capture context information about user's surrounding information. This thesis, however, excludes a context acquisition system, e.g. to acquire context information from software agents and sensors. Context information provided in this thesis is supposedly acquired from agents and sensors.

The further step is about conceptual modeling with Description Logics as mentioned in [32]. The intention is to represent context information by classifying concept and sub concepts, defining relations among concepts, and defining individuals belong to a concept(s)(see step ② in Figure 3.1).

The OWL semantic web of CIS context ontology is generated from the conceptual model which is initially presented in DLs notation (see ③ in Figure 3.1). As depicted in Figure 3.1, Swoop 2.3.1 and Protege 4.0 are chosen to support modeling context ontology in OWL semantic web format. Both ontology editors are featured with visual interface, which is very helpful to develop rapid and complex ontology.

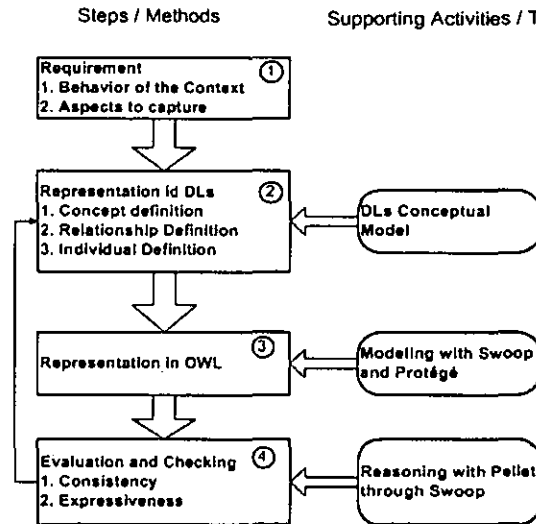


Figure 3.1: Steps to develop context ontology in OWL semantic web language

Once context ontology has been completely defined, it is further required to evaluate the ontology (③). To do so, Swoop OWL editor is connected to Pellet OWL DL reasoner. The evaluation of context ontology will arrive to the conclusion of consistency of CIS context ontology (see step ④), and the expressiveness of CIS context ontology being designed could also be identified.

3.2 Representing Context Ontology in DLs

Borgida [32] mentioned about the steps to create conceptual modeling in DLs. Besides using DLs syntax, Borgida also proposed abstract syntax to construct conceptual model, which is further used as OWL semantic web syntax. This section discusses the steps to create conceptual modeling as mentioned by Borgida.

3.2.1 Identify the concepts and develop its taxonomy

By referring to [20]-[21]-[22]-[23], 5 aspects have been defined to be included in the CIS context ontology, namely *Person*, *Device*, *Activity*, *Location*, and *Network*.

Concept *Person* is used to describe involved user or person profile, such as full name and email address, in CIS Department. The computing devices used by a person are described by concept *Device*. Concept *Network* is used to draw the computer network infrastructures and resources belong to the CIS Department. Activities belong to a person is described in concept *Activity*. And the last, concept *Location* describes the person current position around CIS Department building or UTP campus.

Figure 3.2 shows the highest level of CIS context ontology presented in informal RDF graphical notation. *Person*, *Device*, *Activity*, *Location*, *Network* are defined as main concepts, which are sub class of *ContextAware* ontology.

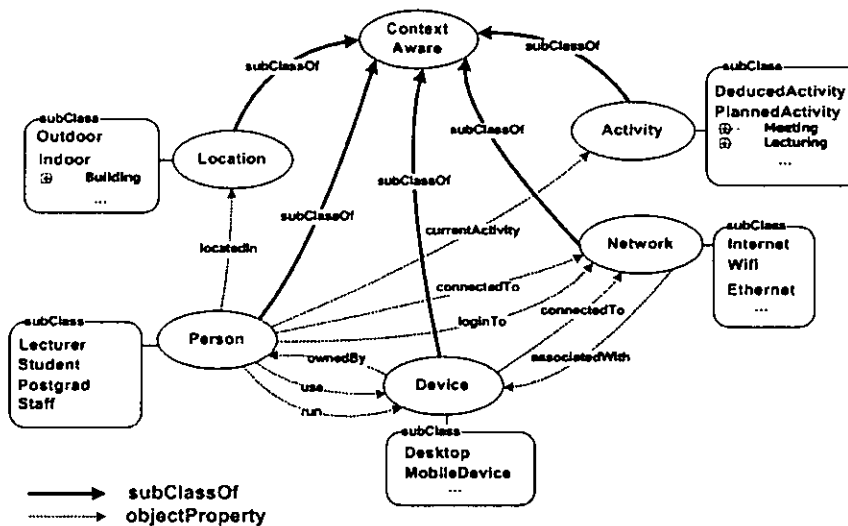


Figure 3.2: Highest Level CIS Context Ontology

The CIS context ontology describes user's environment surrounding Computer and Information Science Department (CIS) at Universiti Teknologi PETRONAS. The concepts involved in CIS context ontology are declared using DLs (Description Logics) notation as follows:

$$(Location, Person, Activity, Device, Network) \sqsubseteq \Delta^{\mathcal{I}}$$

where $\Delta^{\mathcal{I}}$ is CIS context interpretation domain.

CIS context model distinguishes location into Outdoor and Indoor place. Indoor place indicates location inside the CIS building. If the position of a person is outside, it is indicated by longitude and latitude point, which can be acquired from GPS-enabled device.

Indoor location is composed of room, which can be a class room, seminar room,

tutorial room, office room, and laboratory room, as depicted by ontology graphical notation in Figure 3.4. The concept of *Location*, including its sub classes, are composed in DLs notation as follows:

$(Indoor, Outdoor) \sqsubseteq Location$
 $(Longitude, Latitude) \sqsubseteq Outdoor$
 $(Room, Building) \sqsubseteq Indoor$
 $(Room) \sqsubseteq Building$
 $ClassRoom, SeminarRoom, LectureHall, MeetingRoom,$
 $OfficeRoom, Lab) \sqsubseteq Room$

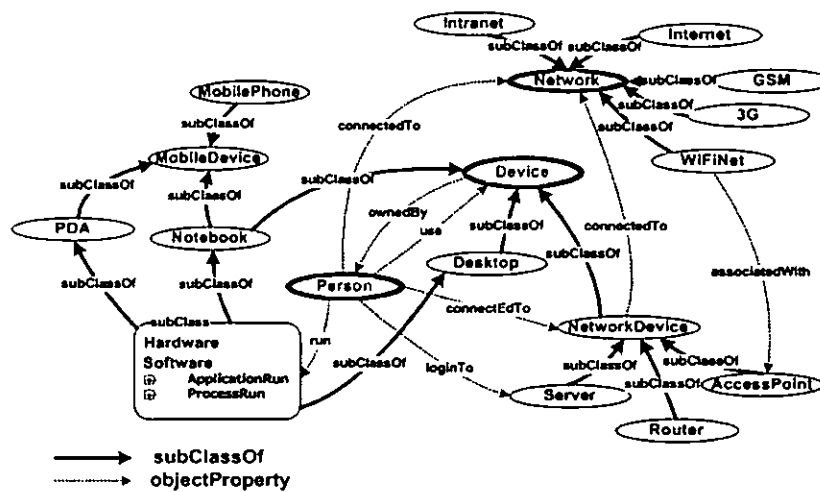


Figure 3.3: Description of Person, Device, and Network Concept

3.2.2 Identify the individuals belong to concept

Once the concepts and their taxonomy have been defined, the individuals belongs to a concept(s) can further be identified. For example, the concept *ClassRoom* describes the class room used by CIS Department for lecturing activity. Following DLs axioms describe the memberships or individuals exist in *ClassRoom* concept.

$ClassRoom \equiv \{C01, C02, C03, C04, C05, C06, D01, D02, D03, D04, D05, D06\}$

$LectureHall \equiv \{LH01, LH02, LH03, LH04, LH04, LH06\}$

$MeetingRoom \equiv \{010310, 010210, 0203010\}$

$OfficeRoom \equiv \{LECTUREROOM, POSTGRADROOM\}$

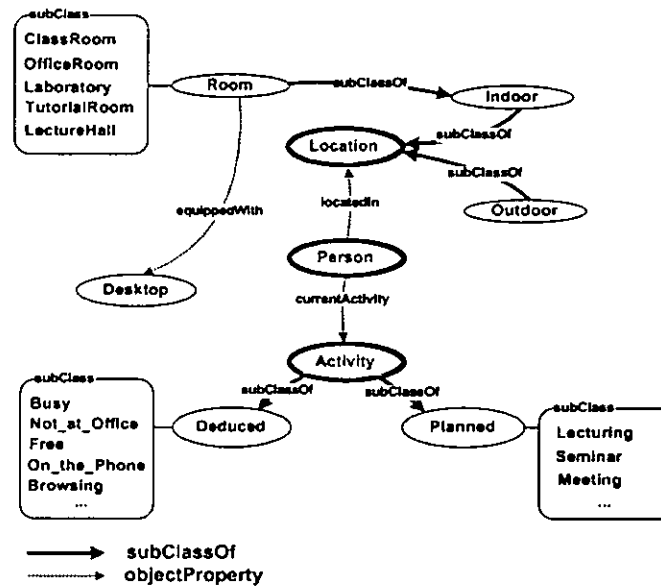


Figure 3.4: Description about Person, Activity, and Location Concept

A small number of existing browsers application are accommodated as individuals in concept *Browser* such as *IE*, *FIREFOX*, *MOZILLA*, *SAFARI*, *OPERA*. Thus, the axiom above can also involve individuals of concept *Browser* to be declared in DLs notation as follow:

$$Browser \equiv \{IE, FIREFOX, MOZILLA, SAFARI, OPERA\}$$

The complete specification of individuals can be seen in the Apendix A.

3.2.3 Distinguish Role to link the concepts

A concept is directed with another concept by means of a role, as depicted by highest level of context ontology in Figure 3.2. In DLs, a role can be distinguished by its domain and range. The description of roles related to the concept of *Person* presented

in the previous subsection are declared in DLs notation as follows.

$$\begin{aligned}
 & Person \sqcap \exists use. Device \\
 & Person \sqcap \forall locatedIn. Location \\
 & Person \sqcap \forall currentActivity. Activity \\
 & Person \sqcap \forall logInto. Server \\
 & Person \sqcap \forall connectedTo. Internet \\
 & Person \sqcap \forall connectedTo. Intranet \\
 & Person \sqcap \forall run. ApplicationRun \\
 & Device \sqcap \forall ownedBy. Person \\
 & Person \sqcap \forall logInto. \{NOVELNETWARE\}
 \end{aligned}$$

Role *use* is declared to describe the relation between concept *Person* and *Device*. For example, to describe there exists a *Desktop* used by a person is reflected by DLs axiom $Person \sqcap \exists use. Desktop$.

In CIS context model, *Profile* is composed of concepts that declare full name, office address, phone number, and email address. Those context information are used to describe person's profile. For example, a role *fullName* is declared, which is to describe person's full name. Actually, the value of this role *fullName* can be related to literal name or data items such as strings. Nevertheless, DLs do not distinguish the role whose value is concept or associated with data type. Therefore, in CIS context model, the *XSD* is introduced as a concept name whose instances are data type definition. This is to describe data type value range. In the implementation of OWL language later, *XSD* can be transformed into data type like string, date, alphanumeric etc. Therefore, it is defined that the role whose value is instance of *XSD* is categorized as data type property.

$$XSD \equiv \{STRING, TIME, DATE, \dots, INTEGER, DECIMAL, BOOLEAN\}$$

In OWL data type role and object role are distinguished and disjoint each other, hence their interpretation domain are also separated. In OWL, object property is subset of $\Delta^{\mathcal{I}}$, while data type property is subset of $\Delta_D^{\mathcal{I}}$. OWL adopts XML Schema Datatype (*XSD*) definition to describe data type used in data type property. Following axioms describe the person's profile declared as role with data type definition.

$$\begin{aligned} \text{Lecturer} &\equiv \text{Person} \sqcap \exists \text{fullName}.\{\text{STRING}\} \\ \text{Staff} &\equiv \text{Person} \sqcap \exists \text{officeAddress}.\{\text{STRING}\} \\ \text{PostGrad} &\equiv \text{Person} \sqcap \exists \text{emailAddress}.\{\text{STRING}\} \end{aligned}$$

3.2.4 Identify sub roles

The role *run* is defined to describe some application software run by a person. The concept of *Software* is previously declared as subclass of *Device*. This role is defined as sub role of *use*. The family of DLs in which role hierarchy is used is specified as \mathcal{H} . In another word, role *run* determines the DLs expressiveness of ontology being specified.

3.2.5 Determine concept and role constraints

Regarding to Figure 3.3, the domain and range of role *ownedBy* is inverse of role *use*. Therefore, it can also be written in DLs notation as $\text{use} \equiv \neg \text{ownedBy}$. The use of inverse role indicates the expressiveness of DLs specification. Thus, for DLs specification that has inverse role is categorized as \mathcal{I} language.

The axiom $\text{Person} \sqcap \forall \text{loginTo}.\{\text{NOVELNETWARE}\}$ relates role *loginTo* with nominal. This axioms describes a condition in which a person has to log in to the *Netware* server prior to accessing the network resource. $\{\text{NOVELNETWARE}\}$ is declared as instance of concept *Server*. This expressiveness reflects the use of nominal in DLs language, expressed with letter \mathcal{O} .

Another role, namely *connectedTo*, is used to describe a person that is connected to a network device. This role also is used to describe concept *Device* that is connected to the Internet, as sub concept of *Network*. Regarding its relation, this role transitive that makes *Person* is connected to *Network*. The characteristic of transitive role makes the minimum \mathcal{ALC} language in our CIS context model become \mathcal{S} .

Number restriction is assigned in axiom $\equiv 1.\text{currentActivity}$ and $\geq 2.\text{run}$. Axiom $\equiv 1.\text{currentActivity}$ restricts role *currentActivity* with one role value (role concerned), meaning that person is restricted with only one possible activity that he can do within a specific time. Meanwhile, $\geq 2.\text{run}$ restricts the role *run* with 2, meaning that a person can run more than two application in his computing devices. The use of number restriction indicates DLs language with \mathcal{N} .

Practically, in CIS context model, activities related to a person is distinguished into scheduled and deduced activities, which are declared as concept *Planned* and *Deduced*, respectively. *Planned* concept is to describe a situation when a person is

doing activities that have been on schedule. Activities like meeting and lecturing are classified as planned activities.

Assume that a user is required to put his schedule into the calendar or organizer application. The context related to user's scheduled activity actually can be acquired by means of the information sent by software agents that are attached to the existing calendar or organizer application software, e.g. Sunbird, Outlook, iCal, etc.

$$\begin{aligned}
 (\textit{Planned}, \textit{Deduced}) &\sqsubseteq \textit{Activity} \\
 (\textit{Meeting}, \textit{Lecturing}, \textit{Seminar}, \textit{LabActivity}, \textit{Tutorial}) &\sqsubseteq \textit{Planned} \\
 (\textit{Busy}, \textit{Free}, \textit{Chatting}, \textit{Browsing}, \textit{Not_At_Office}, \textit{Available}, \textit{On_the_Phone}, \\
 \textit{Opening_Email}) &\sqsubseteq \textit{Deduced} \\
 \textit{Free} &\equiv \neg \textit{Busy}
 \end{aligned}$$

Context information pertaining to deduced activity is obtained by deducing the rules that are already defined in the context model. For example, a person is assumed to be busy if the context-awareness system (including the application) get the information of what is person doing and where. Hence, the context-aware system deduce a person is busy according to the given deduction rule about the person's current activity and the venue of activity to take place.

For example, in deduced activity, the concept of *Browsing* is declared to describe an activity in which a person is running an Internet application, e.g. web browser to surf information throughout the Internet. This activity requires a person that is connected to the Internet. To express this activity, the concept of *Browsing* is restricted as follows.

$$\textit{Browsing} \equiv \textit{Person} \sqcap \forall \textit{connectedTo}.\textit{Internet} \sqcap \exists \textit{run}.\textit{Browser}$$

Several existing browser applications are accommodated as individuals in concept *Browser*, declared as $\textit{Browsing} \equiv \textit{IE}, \textit{FIREFOX}, \textit{MOZILLA}, \textit{SAFARI}, \textit{OPERA}$. Hence, in the axiom above individuals of concept *Browser* could be declared in the DLs axiom as follows:

$$\textit{Browsing} \equiv \textit{Person} \sqcap \forall \textit{connectedTo}.\textit{Internet} \sqcap \exists \textit{run}.\{ \textit{IE}, \textit{FIREFOX}, \textit{MOZILLA}, \textit{SAFARI}, \textit{OPERA} \}$$

In CIS context, the concept of *Not_At_Office* is to describe a person where he/she is not in the office room. At CIS Department, assumed that all of lecturer room and postgraduate room are categorized as office room. Therefore, $\textit{OfficeRoom} \equiv$

$\{POSTGRADROOM, LECTUREROOM\}$. In DLs, the *Not_At_Office* situation is described as follows.

$$\begin{aligned} Not_At_Office &\equiv Person \sqcap \forall locatedIn. \neg OfficeRoom \\ Not_At_Office &\equiv Person \sqcap \forall locatedIn. \neg (\{POSTGRADROOM\}, \\ &\quad \{LECTUREROOM\}) \end{aligned}$$

3.3 Semantic Web Model

The DLs specification of CIS context model becomes the starting point to generate OWL semantic web model. Actually there are many semantic web tools that can be used to generate semantic web model, either using graphical or non graphical tool. In this thesis, Swoop OWL editor is connected to Pellet OWL DL reasoner to reason the CIS context ontology. Swoop is chosen since it is able to display the source of inconsistency of ontology when reasoning has been performed.

3.3.1 OWL Header Definition

In OWL semantic web document, first of all the *uri* (Uniform Resource Identifier) has to be defined. In CIS context ontology, the *uri* is defined as *cis*, which reflects CIS context ontology model. The *cis* namespace is declared in OWL semantic web header by declaring the *uri* as <http://context.org/cis>.

Another header in OWL semantic web document that should be declared is XML namespaces, because OWL is written in XML document. XML namespaces are used for providing uniquely named elements and attributes in an XML document. They are defined by a W3C recommendation. An XML instance may contain element or attribute names from more than one XML vocabulary. In OWL document, vocabulary such as *owl*, *rdf*, *rdfs*, and *xsd* have to be defined as well. Those vocabularies are used for describing OWL semantic web syntax and language. They are defined in semantic web W3C recommendation (<http://www.w3.org/2004/OWL>). The *xsd* vocabulary is used to support XML Schema Datatype definition (<http://www.w3.org/TR/xmlschema-2>). The following lines describe the header of OWL semantic web of CIS context model.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY cis "http://context.org/cis">
  <!ENTITY owl "http://www.w3.org/2002/07/owl#">
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

```

<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&cis;"
  xmlns:owl="&owl;"
  xmlns:rdf="&rdf;"
  xmlns:rdfs="&rdfs;">

```

With respect to DLs model, the OWL semantic web of CIS context ontology is also composed of 5 main classes: *Person*, *Device*, *Activity*, *Location*, and *Network*. The concept in DLs are implemented as class in semantic web language, while role as property. This section briefly describes all the 5 main class and their related properties. In the following subsection, the description of OWL semantic web model for each particular main class will be discussed. The complete OWL semantic web language model is provided in the Appendix B.

3.3.2 Semantic Web of Class Person

In CIS context ontology, the DLs axioms of *Person* and its sub concepts are defined as follow:

$$\begin{aligned}
 (\textit{Lecturer}, \textit{Staff}, \textit{Postgrad}, \textit{Student}) &\sqsubseteq \textit{Person} \\
 \textit{Profile} &\sqsubseteq (\textit{Lecturer}, \textit{Staff}, \textit{Postgrad}, \textit{Student})
 \end{aligned}$$

From those DLs axioms, the OWL semantic web model can be directly generated. Most of semantic web developers use visual OWL editors, e.g. Protege and Swoop, because those editors are visual and very useful for rapid development with very complex taxonomy and ontology. For that purpose, in this thesis, Swoop OWL editor is also used to generate OWL semantic web of CIS context ontology. Besides the visual interface, Swoop also provides the textual interface to see the XML document of ontology being written. The following Figure 3.5 shows the OWL semantic web notation of class *Person* and its sub classes definition.

Class *Person* also relates some data type properties. *OWL:DatatypeProperty* determines the relation between data type property with XSD data format. Regarding to OWL document specified in [61], the data type uses XML Schema Data type definition. To express identity of a user, person's profile class is created and it requires context information like full name, person's address, person's email address, instant messenger ID, phone number, etc. All of that user's profile information is not declared as sub classes. Instead, they are declared as data type property, which relates class *Profile* with XSD data. The description of data type property related to class

```

460 <owl:Class rdf:about="#Person"/>
461 <owl:Class rdf:about="#Lecturer">
462 <rdfs:subClassOf rdf:resource="#Person"/>
463 </owl:Class>
464 <owl:Class rdf:about="#PostGrad">
465 <rdfs:subClassOf rdf:resource="#Person"/>
466 </owl:Class>
467 <owl:Class rdf:about="#Student">
468 <rdfs:subClassOf rdf:resource="#Person"/>
469 </owl:Class>
470 <owl:Class rdf:about="#Staff">
471 <rdfs:subClassOf rdf:resource="#Person"/>
472 </owl:Class>
473 <owl:Class rdf:about="#Profile">
474 <rdfs:subClassOf rdf:resource="#Lecturer"/>
475 <rdfs:subClassOf rdf:resource="#PostGrad"/>
476 <rdfs:subClassOf rdf:resource="#Staff"/>
477 <rdfs:subClassOf rdf:resource="#Student"/>
478 </owl:Class>

```

Figure 3.5: OWL Notation of Class Person and its Sub Classes

person are depicted in Figure 3.6.

For example, to express information of person's full name, *xsd:string* is used and directed with *fullName owl:DatatypeProperty*. As in Figure 3.6, class *Person* is the domain of this *fullName* data type property, whereby *xsd:string* is the range. The complete OWL code of class *Person* is presented in Appendix B.

```

588 <owl:DatatypeProperty rdf:about="#fullName">
589 <rdfs:domain rdf:resource="#Profile"/>
590 <rdfs:range rdf:resource="#xsd:string"/>
591 </owl:DatatypeProperty>
592 <owl:DatatypeProperty rdf:about="#emailAddress">
593 <rdfs:domain rdf:resource="#Profile"/>
594 <rdfs:range rdf:resource="#xsd:string"/>
595 </owl:DatatypeProperty>
596 <owl:DatatypeProperty rdf:about="#gender">
597 <rdfs:domain rdf:resource="#Profile"/>
598 <rdfs:range rdf:resource="#xsd:string"/>
599 </owl:DatatypeProperty>
600 <owl:DatatypeProperty rdf:about="#homeAddress">
601 <rdfs:domain rdf:resource="#Profile"/>
602 <rdfs:range rdf:resource="#xsd:string"/>
603 </owl:DatatypeProperty>
604 <owl:DatatypeProperty rdf:about="#messengerID">
605 <rdfs:domain rdf:resource="#Profile"/>
606 <rdfs:range rdf:resource="#xsd:string"/>
607 </owl:DatatypeProperty>
608 <owl:DatatypeProperty rdf:about="#officeAddress">
609 <rdfs:domain rdf:resource="#Profile"/>
610 <rdfs:range rdf:resource="#xsd:string"/>
611 </owl:DatatypeProperty>
612 <owl:DatatypeProperty rdf:about="#phoneNumber">
613 <rdfs:domain rdf:resource="#Profile"/>
614 <rdfs:range rdf:resource="#xsd:string"/>
615 </owl:DatatypeProperty>

```

Figure 3.6: owl:DatatypeProperty of class Profile

owl:locatedIn and *owl:currentActivity* connect class *Person* with class *Location* and class *Activity*, respectively. Both properties are defined as *owl:ObjectProperty*.

As can be seen in Figure 3.7, the domain of *owl:locatedIn* is class *Person*, and the range is class *Location*. By observing this example, OWL semantic web language distinguishes ontology properties into data type and object properties. However, as described in the previous section, OWL standard defines both properties have different interpretation domain, and both properties are also disjoint each other. Figure 3.7 shows object properties related to class *Person* in CIS context ontology model.

```

691 <owl:ObjectProperty rdf:about="#connectedTo">
692   <rdf:type rdf:resource="#owl:TransitiveProperty"/>
693   <rdfs:domain rdf:resource="#Device"/>
694   <rdfs:domain rdf:resource="#Person"/>
695   <rdfs:range rdf:resource="#Device"/>
696   <rdfs:range rdf:resource="#Network"/>
697 </owl:ObjectProperty>
698 <owl:ObjectProperty rdf:about="#currentActivity">
699   <rdfs:domain rdf:resource="#Person"/>
700   <rdfs:range rdf:resource="#Activity"/>
701 </owl:ObjectProperty>
702 <owl:ObjectProperty rdf:about="#locatedIn">
703   <rdfs:domain rdf:resource="#Person"/>
704   <rdfs:range rdf:resource="#Location"/>
705 </owl:ObjectProperty>
706 <owl:ObjectProperty rdf:about="#logInto">
707   <rdfs:domain rdf:resource="#Person"/>
708   <rdfs:range rdf:resource="#Server"/>
709 </owl:ObjectProperty>
710 <owl:ObjectProperty rdf:about="#ownedBy">
711   <owl:inverseOf rdf:resource="#use"/>
712 </owl:ObjectProperty>
713 <owl:ObjectProperty rdf:about="#run">
714   <rdfs:range rdf:resource="#Software"/>
715   <rdfs:subPropertyOf rdf:resource="#use"/>
716 </owl:ObjectProperty>
717 <owl:ObjectProperty rdf:about="#use">
718   <rdfs:domain rdf:resource="#Person"/>
719   <rdfs:range rdf:resource="#Device"/>
720   <owl:inverseOf rdf:resource="#ownedBy"/>
721 </owl:ObjectProperty>

```

Figure 3.7: owl:ObjectProperty Related to Class Person

3.3.3 Semantic Web of Class Network

Class *Network* describes the available network resources that a person can exploit and communicate using his/her computer devices, e.g. computer desktop, notebook, and mobile device as well. This class also to describe that a person may initiate a conversation through the existing network resource such as GSM or 3G Network. He/she may access the available Internet (or Intranet) by means of the existing network and Internet resources as well.

As depicted in Figure 3.8, class *Internet* describes a condition in which the class *Network* connects to the Internet. We accommodate this requirement by representing *Proxy* and *Gateway* sub class of *Internet* (see line 336-347). As UTP policy, to uti-

```

317 <owl:Class rdf:about="#Network"/>
318 <owl:Class rdf:about="#Internet">
319 <rdfs:subClassOf rdf:resource="#Network"/>
320 </owl:Class>
321 <owl:Class rdf:about="#IMT5G">
322 <rdfs:subClassOf rdf:resource="#Network"/>
323 </owl:Class>
324 <owl:Class rdf:about="#WifiNetwork">
325 <rdfs:subClassOf rdf:resource="#Network"/>
326 </owl:Class>
327 <owl:Class rdf:about="#GPRS">
328 <rdfs:subClassOf rdf:resource="#Network"/>
329 </owl:Class>
330 <owl:Class rdf:about="#GSM">
331 <rdfs:subClassOf rdf:resource="#Network"/>
332 </owl:Class>
333 <owl:Class rdf:about="#Intranet">
334 <rdfs:subClassOf rdf:resource="#Network"/>
335 </owl:Class>
336 <owl:Class rdf:about="#Proxy">
337 <rdfs:subClassOf rdf:resource="#Internet"/>
338 <owl:oneOf rdf:parseType="Collection">
339 <rdf:Description rdf:about="#160.0.226.206"/>
340 <rdf:Description rdf:about="#160.0.226.207"/>
341 <rdf:Description rdf:about="#160.0.226.208"/>
342 </owl:oneOf>
343 </owl:Class>
344 <owl:Class rdf:about="#Gateway">
345 <rdfs:subClassOf rdf:resource="#Internet"/>
346 </owl:Class>
347 <rdf:Gateway rdf:about="#160.0.226.202"/>

```

Figure 3.8: OW Notation of Class Network and its Sub Classes

lize the Internet resource, a person who uses computer devices should configure the Internet Gateway and Proxy as well.

3.3.4 Semantic Web of Class Device

Class *Device* is composed of *MobileDevice*, *NetworkDevice*, and *Desktop* as its sub classes. Class *Software* is sub class of *Desktop*, *Notebook* and *PDA*. This entity is used to model software used by a person. The software resource is distinguished into process run and application run, which are described by class *ProcessRun* and class *ApplicationRun*, respectively.

Class *ApplicationRun* reflects the applications executed by a person. When deducing CIS context model in the implementation later, context-aware application can deduce the software that is being run by a person. The various applications run are distinguished into *EmailApplication*, *OfficeApplication*, *InternetApplication*, and *IMApplication* (Instant Messenger Application).

Class *NetworkDevice* is to describe computer network devices used to connect to the available network resources. The network devices comprises 3 sub classes, namely *Server*, *Router*, and *AccessPoint*. Figure 3.9 shows OWL semantic web of class *Device* and its sub classes. The complete OWL specification related to class

Device can be seen in the Appendix B.

```

329 ...
330 <owl:Class rdf:about="#Device"/>
331   <owl:Class rdf:about="#Desktop">
332     <rdfs:subClassOf rdf:resource="#Device"/>
333   </owl:Class>
334   <owl:Class rdf:about="#MobileDevice">
335     <rdfs:subClassOf rdf:resource="#Device"/>
336   </owl:Class>
337   <owl:Class rdf:about="#NetworkDevice">
338     <rdfs:subClassOf rdf:resource="#Device"/>
339   </owl:Class>
340     <owl:Class rdf:about="#Router">
341       <rdfs:subClassOf rdf:resource="#NetworkDevice"/>
342     </owl:Class>
343   <owl:Class rdf:about="#PDA">
344     <rdfs:subClassOf rdf:resource="#MobileDevice"/>
345   </owl:Class>
346   <owl:Class rdf:about="#Notebook">
347     <rdfs:subClassOf rdf:resource="#Device"/>
348   </owl:Class>
349   <owl:Class rdf:about="#Software">
350     <rdfs:subClassOf rdf:resource="#Desktop"/>
351     <rdfs:subClassOf rdf:resource="#Notebook"/>
352     <rdfs:subClassOf rdf:resource="#PDA"/>
353     <owl:disjointWith rdf:resource="#Hardware"/>
354   </owl:Class>
355   <owl:Class rdf:about="#Hardware">
356     <rdfs:subClassOf rdf:resource="#Desktop"/>
357     <rdfs:subClassOf rdf:resource="#Notebook"/>
358     <rdfs:subClassOf rdf:resource="#PDA"/>
359     <owl:disjointWith rdf:resource="#Software"/>
360   </owl:Class>
361 ...

```

Figure 3.9: OWL Notation of Class Device and its Sub Classes

The relation between user and computing resources is modeled by object property *use*, which relates class *Person* with class *Device*. Object property *connectedTo* relates *Person* with *Network* resource. The *connectedTo* object property also models a relation between *Network* entity that connects to the *Internet*. This relation makes *connectedTo* property as *Transitive* property.

3.3.5 Semantic Web of Class Location

Class *Location* describes location related to a person. *Outdoor* is a sub class of *Location*. A position of user is indicated by *longitude* and *latitude* values. Class *Indoor*, which is also a sub class of *Location*, describes a user's position related to its geographical position, e.g. in a room when a user or a person is inside a building. In CIS context, indoor location is derived into *Room*, which is to distinguish room functionality used by CIS Department, Universiti Teknologi PETRONAS (UTP). Object property *owl:locatedIn* is used to model a person that exists at a certain location, either at outdoor space or indoor.

Class *Outdoor* reflects a situation where a person exists in outdoor environment surrounding UTP Campus. Assume that the position of latitude and longitude are acquired through a GPS-enabled gizmo. To represent the value of longitude and latitude position, the *DatatypeProperty owl:longitude* and *owl:latitude* are used. Figure 3.10 shows OWL semantic web of class *Location* description. The complete OWL code is provided in the Appendix B.

```

240 ...
241 <owl:Class rdf:about="#Location"/>
242   <owl:Class rdf:about="#Indoor">
243     <rdfs:subClassOf rdf:resource="#Location"/>
244   </owl:Class>
245   <owl:Class rdf:about="#MeetingRoom">
246     <owl:Class rdf:about="#Laboratory">
247       <rdfs:subClassOf rdf:resource="#Indoor"/>
248     <owl:oneOf rdf:parseType="Collection">
249       <rdf:Description rdf:about="#DataCm"/>
250       <rdf:Description rdf:about="#Multimedia"/>
251       <rdf:Description rdf:about="#Programminglab"/>
252       <rdf:Description rdf:about="#VRLab"/>
253     </owl:oneOf>
254   </owl:Class>
255   <owl:Class rdf:about="#ClassRoom">
256     <rdfs:subClassOf rdf:resource="#Indoor"/>
257     <owl:oneOf rdf:parseType="Collection">
258       <rdf:Description rdf:about="#C01"/>
259       <rdf:Description rdf:about="#C02"/>
260       <rdf:Description rdf:about="#C03"/>
261       <rdf:Description rdf:about="#C04"/>
262       <rdf:Description rdf:about="#C05"/>
263       <rdf:Description rdf:about="#C06"/>
264       <rdf:Description rdf:about="#D01"/>
265       <rdf:Description rdf:about="#D02"/>
266       <rdf:Description rdf:about="#D03"/>
267       <rdf:Description rdf:about="#D04"/>
268       <rdf:Description rdf:about="#D05"/>
269       <rdf:Description rdf:about="#D06"/>
270     </owl:oneOf>
271   </owl:Class>
272 ...

```

Figure 3.10: OWL Notation of Class Location and and its Sub Classes

3.3.6 Semantic Web of Class Activity

Like in the DLs model, practically activities related to a person in CIS context model are categorized into scheduled and deduced activities. In this subsection, the OWL semantic web model of class *activity* is briefly discussed. Activities *lecturing* and *meeting* are classified as planned or scheduled activities.

3.3.7 Class Restriction

As described in DLs model of CIS context, some classes are composed and restricted by class axioms. For example, to express that a person can only have one activity at

```

21 ...
22 <owl:Class rdf:about="#Activity">
23   <rdfs:subClassOf rdf:resource="#owl:Thing"/>
24   <owl:equivalentClass>
25     <owl:Restriction>
26       <owl:cardinality rdf:datatype="#xsd:nonNegativeInteger">1</owl:cardinality>
27       <owl:onProperty rdf:resource="#currentActivity"/>
28     </owl:Restriction>
29   </owl:equivalentClass>
30 </owl:Class>
31 <owl:Class rdf:about="#Planned">
32   <rdfs:subClassOf rdf:resource="#Activity"/>
33 <owl:Class rdf:about="#Seminar">
34   <rdfs:subClassOf rdf:resource="#Planned"/>
35 </owl:Class>
36 <owl:Class rdf:about="#Lecturing">
37   <rdfs:subClassOf rdf:resource="#Planned"/>
38 </owl:Class>
39 </owl:Class>
40 <owl:Class rdf:about="#Deduced">
41   <rdfs:subClassOf rdf:resource="#Activity"/>
42 </owl:Class>
43 <owl:Class rdf:about="#Free">
44   <rdfs:subClassOf rdf:resource="#Deduced"/>
45 </owl:Class>
46 ...

```

Figure 3.11: OWL Notation of Class Activity and its Sub Classes

a certain time, the cardinality restriction can be used in axiom $\equiv 1.currentActivity$. The OWL semantic web syntax of this restriction axiom can be seen in Figure 3.11 (see line 26).

For example, class *Busy* is declared to express situation of a user when he/she is busy, i.e. by assuming a user is busy if his/her is doing his daily planned activities or a user is working on his workstation by running some related office application software. OWL semantic web code for class *Busy* axiom is depicted in Figure 3.12. The busy situation could be expressed by means of axioms in DLs syntax as follows.

$$\begin{aligned}
 Busy &\equiv Person \sqcap \exists currentActivity.Planned \\
 Busy &\equiv Person \sqcap \exists run.OfficeApplication \\
 Busy &\equiv Person \sqcap \exists run.(WORDPROCESSOR \sqcup SPREADSHEET \sqcup \\
 &PDFREADER).
 \end{aligned}$$

3.4 OWL Semantic Checking

As discussed in Chapter 2, semantic consistency checking is carried out to detect whether unsatisfiable concepts exist in ontology model. Unsatisfiable concept is equivalent to concepts and axioms that belong (members of) to the empty set (\emptyset). In this

```

101 ...
102 <owl:Class rdf:about="#Busy">
103   <rdf:type rdfs:Class rdfs:source="#Deduced"/>
104   <owl:equivalentClass>
105     <owl:Restriction>
106       <owl:onProperty rdf:resource="#sum"/>
107       <owl:someValuesFrom rdf:resource="#BRIInApplication"/>
108     </owl:Restriction>
109   </owl:equivalentClass>
110   <owl:equivalentClass>
111     <owl:Restriction>
112       <owl:onProperty rdf:resource="#CurrentActivity"/>
113       <owl:someValuesFrom rdf:resource="#Planned"/>
114     </owl:Restriction>
115   </owl:equivalentClass>
116   <owl:equivalentClass>
117     <owl:Restriction>
118       <owl:onProperty rdf:resource="#sum"/>
119       <owl:someValuesFrom
120         <owl:Class>
121           <owl:unionOf (rdf:parType="Collection">
122             <rdf:Description>
123               <owl:oneOf (rdf:parType="Collection">
124                 <rdf:Description rdf:about="#SpreadSheet"/>
125               </owl:oneOf>
126             </rdf:Description>
127             <rdf:Description>
128               <owl:oneOf (rdf:parType="Collection">
129                 <rdf:Description rdf:about="#WordProcessor"/>
130               </owl:oneOf>
131             </rdf:Description>
132             <rdf:Description>
133               <owl:oneOf (rdf:parType="Collection">
134                 <rdf:Description rdf:about="#PDFReader"/>
135               </owl:oneOf>
136             </rdf:Description>
137           </owl:unionOf>
138         </owl:Class>
139       </owl:someValuesFrom>
140     </owl:Restriction>
141   </owl:equivalentClass>
142 </owl:Class>
143 ...

```

Figure 3.12: OWL Notation of Class Restriction on Class Busy

thesis, Pellet reasoner is used for semantic consistency checking, which involves consistency, subsumption, and instance checking. Pellet works based-on Tableau Reasoning Algorithm [48]-[62], to detect any inconsistency of logical axioms in semantic web model.

3.4.1 Consistency checking

The intention is to check whether the knowledge in ontology is consistent or not. Therefore, the ontology \mathcal{O} is consistent such that \mathcal{O} satisfies the interpretation of \mathcal{I} . In other word it can be said that $\mathcal{I} \models \mathcal{O}$. For checking purpose, three examples of checking strategy have been defined to be assigned to context ontology and to be reasoned by Pellet version 1.5.

The first strategy consists of axioms that correspond to the class disjointness and quantifier restriction.

Definition 3.1. Let $c_1, c_2, c_3 \in \mathcal{C}$ be concept name, $r \in \mathcal{R}$ be role name, c_2 is the range of $\forall c_1 \sqcap r.c_2$, whereas $c_2 \sqsubseteq \neg c_3$, such that c_3 cannot be applied for the range of r that causes property concerned of r contradicts each other.

The axioms in Definition 3.1 guard if two classes are disjoint each other, then both class cannot be restricted either by existential or a universal quantifier. For example, class restrictions (and axioms) are defined in our context ontology as follow (using DLs notation).

$Person, Indoor, Outdoor \sqsubseteq Class$
 $Indoor \equiv \neg Outdoor$
 $Person \sqcap \exists locatedIn.Indoor$
 $Person \sqcap \exists locatedIn.Outdoor$

In DLs, a value constraint (value restriction or existential quantifier restriction) puts constraints on the range of the property when applied to a particular class description. Once Pellet reasoned class restriction above, the reasoner discovers inconsistency in the ontology. It is because of the disjointness of the two classes ($Indoor \sqsubseteq \neg OutdoorSpace$) that is used as the range of property *locatedIn*.

Proof. Value restriction defines individual of class *Person* for which holds that if the pair (x, y) is the property concerned of *locatedIn*, then y should be an instance of the class *Indoor*. Since *Indoor* is disjoint with *Outdoor*, hence the the property concerned (value of property) of *locatedIn* is not be an instance of the class *Outdoor* ($Outdoor = \neg Indoor$ or $Indoor = \neg Outdoor$). Given the constraints above, it can be proved by means of Tableaux Reasoning Algorithm [62] that the axioms in Definition 3.1 is clash.

$$\begin{aligned} & (Person \sqcap \exists locatedIn.Outdoor)(x), (Person \sqcap \exists locatedIn.Outdoor)(x) \\ & Person, \exists locatedIn.Indoor, Person, locatedIn.Outdoor \quad | \sqcap rule \\ & locatedIn(x, y), locatedIn(x, y) \quad | \exists rule \\ & Indoor(y), \neg Indoor(y) \\ & \langle CLASH \rangle \end{aligned}$$

The axioms in Definition 3.1 is further addressed into CIS context ontology. The axioms are reasoned by Pellet through Swoop interface. Surprisingly, Pellet cannot detect the inconsistency of the object property *locatedIn* caused of the disjointness of *Outdoor* and *Indoor*. The result of reasoning process (indicated by ellipse line) is further visualized by Swoop ontology editor, as depicted in Figure 3.13.

The second consistency checking corresponds to the consistency of cardinality constraints. A cardinality constraint puts constraints on the number on property concerned, in the context of this particular class description.

Definition 3.2. Let C be concept name, $D \equiv \{d, e\}$ be individuals, $r \in R$ be role name, and $\equiv n.r$ is restricted role with cardinality constraint. As for in restricted role with $\equiv n$, i.e. $n = 1$, such that $d_1 \equiv \forall r.(d \sqcap e)$ does not hold, because the cardinality of property concern is assigned with instances in two classes.

$$\begin{aligned} & Lecturing \sqsubseteq Class \\ & \equiv 1.currentActivity \\ & Lecturing \equiv \{ICIS, CO, DATACOM\} \\ & Lecturer \equiv \forall currentActivity.(ICIS \sqcap CO) \end{aligned}$$

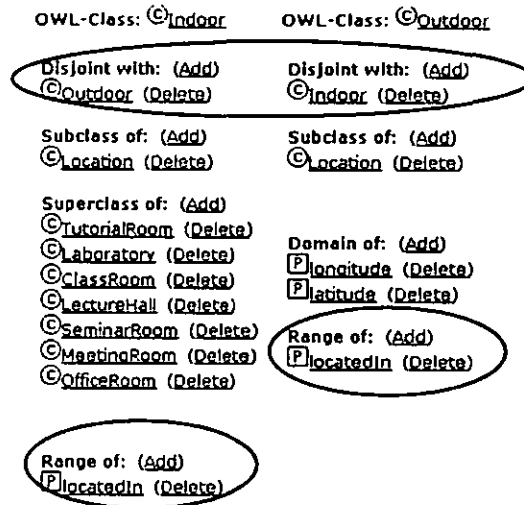


Figure 3.13: Undetected Inconsistency Reasoned by Pellet OWL DL Reasoner

In the above axioms, there exists a case whereby a person has two activities that is impossible to be done at the same time. Once Pellet reasoned the logical restrictions above, this reasoner still returns with inconsistent ontology. A conjunction of individual cardinality value is violated, i.e. restriction equals to 2, not 1 as required above. Such that, cardinality on object property $\equiv 1.currentActivity$ has been violated.

3.4.2 Concept Subsumption

The intention is to check the structure of knowledge in ontology and to obtain the taxonomy of ontology, so that $C \sqsubseteq D$ i.e. $C^I \subseteq D^I$ iff $\mathcal{I} \models \mathcal{O}$. In other words, subsumption checking discovers concept inclusion or sub class definition.

Definition 3.3. Let $c_1, c_2, c_3, c_4 \in \mathcal{C}$ be concept name, $c_2 \sqsubseteq \neg c_4$, $c_1 \sqsubseteq c_2$, $c_3 \sqsubseteq c_4$, such that c_1 cannot be assigned to be equivalent with c_2 .

This definition corresponds to equivalence checking of two subsumed classes. However, the superclasses are disjoint. The intention of this example is to check if two classes or concepts denote the same set of instances, or equivalence, such that $c_1 \equiv c_2$, so that $c_1^I = c_2^I$ iff $\mathcal{I} \models \mathcal{O}$.

As in 3.3, the ontology will be evaluated whether the condition of $c_1 \equiv c_2$ holds, if their super class is disjoint each other. For this purpose, some class axioms that have been generated previously in context ontology are used and the restriction in sub class of *Activity* is given as follows.

Once Pellet reasoned the logical restrictions above, this reasoner can detect and

$$\begin{aligned}
\text{Not_At_Desk} &\sqsubseteq \text{Deduced} \\
\text{Lecturing} &\sqsubseteq \text{Planned} \\
\text{Planned} &\sqsubseteq \neg \text{Deduced} \\
\text{Not_At_Desk} &\equiv \text{Lecturing}
\end{aligned}$$

returns with inconsistent ontology, as can be seen in Figure 3.14.

Proof. Since the superclass of *Not_At_Desk* and *Lecturing* are disjoint each other, i.e. $\text{Planned} \sqsubseteq \neg \text{Deduced}$, when equivalent condition is assigned to *Not_At_Desk* with class *Lecturing*, hence, the context ontology will not be consistent. Pellet will detect inconsistency and it displays the reasoning result as in depicted in Figure 3.14.

<p>OWL-Class: $\text{C}_{\text{Lecturing}}$ Unsatisfiable concept Axioms causing the problem: 1) $\text{C}_{\text{Lecturing}} \sqsubseteq \text{C}_{\text{Planned}}$ 2) $\perp_{\text{C}_{\text{Planned}} \sqsubseteq \neg \text{C}_{\text{Deduced}}}$ 3) $\text{C}_{\text{Not_At_Desk}} \equiv \text{C}_{\text{Lecturing}}$ 4) $\perp_{\text{C}_{\text{Not_At_Desk}} \sqsubseteq \text{C}_{\text{Deduced}}}$</p>	<p>OWL-Class: $\text{C}_{\text{Not_At_Desk}}$ Unsatisfiable concept Axioms causing the problem: 1) $\text{C}_{\text{Not_At_Desk}} \sqsubseteq \text{C}_{\text{Deduced}}$ 2) $\text{C}_{\text{Not_At_Desk}} \equiv \text{C}_{\text{Lecturing}}$ 3) $\perp_{\text{C}_{\text{Lecturing}} \sqsubseteq \text{C}_{\text{Planned}}}$ 4) $\perp_{\text{C}_{\text{Planned}} \sqsubseteq \neg \text{C}_{\text{Deduced}}}$</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.14: Subsumption Checking for Definition 3.3

Subsumption can be performed as *necessary axiom* (\equiv checking, like in the above example) and *sufficient axiom* (\sqsubseteq). Logical constraints can be assigned to a class for subsumption purpose. Depending on the assigned logical constraints, ontology reasoner will classify the result of subsumption checking as intersection, union, or equivalent. Given is an example of subsumption checking, as depicted in Figure 3.15.

In CIS context ontology class *Busy* is restricted with the following axioms:

$$\begin{aligned}
\text{Person} &\sqcap \exists \text{currentActivity.Planned} \\
\text{Person} &\sqcap \exists \text{run.OfficeApplication} \sqcap \forall \text{locatedIn.OfficeRoom}
\end{aligned}$$

The above axioms is to define that a person is assumed to be busy when he/she is doing a planned activity, working with computer by running office applications, e.g. word processor application, reading some paper or journal using PDF viewer in his workstation (at office room). When Pellet reasons those axioms, it concludes that class *Busy* is subsumed as sub class of class *Person*; this is because of the following

axioms:

$$\begin{aligned} &\equiv 1.\text{currentActivity} \\ &Person \sqcap \exists \text{currentActivity.Planned} \\ &\text{domain}(\text{currentActivity}) = Person \\ &\text{range}(\text{currentActivity}) = Activity \end{aligned}$$

Thus, class *Busy* and *Activity* is subsumed by class *Person*:

$$Busy \sqsubseteq Activity \sqsubseteq Person$$

The result of subsumption checking through Pellet reasoner is visualized by Swoop editor as depicted in Figure 3.15

<p>OWL-Class: \textcircled{C}Activity</p> <p>Equivalent to: (Add) $(= 1 \text{ P}_{\text{currentActivity}})$ (Delete)</p> <p>Subclass of: (Add) \textcircled{C}Person (Why?)</p> <p>Superclass of: (Add) \textcircled{C}Planned (Delete) \textcircled{C}Deduced (Delete)</p>	<p>Axioms causing the inference</p> <p>Activity \sqsubseteq Person:</p> <p>1) $(\textcircled{C}Activity \equiv (= 1 \text{ P}_{\text{currentActivity}}))$</p> <p>2) $1_{-}(\text{P}_{\text{currentActivity}} \text{ domain } \textcircled{C}Person)$</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 3.15: *Busy* and *Activity* is subsumed by class *Person*, Visualized by Swoop

3.4.3 Instantiation Checking

Instantiation checking is performed to check if individual i is instance of concept C , i.e. $i \in C^{\mathcal{I}}$ iff $\mathcal{I} \models \mathcal{O}$. In CIS context model, some individuals belong to two classes have been declared. For example, Figure 3.16 shows some instances that are assigned to two class, i.e. to class *LectureHall* and class *ClassRoom*. Previously both classes are defined disjoint each other. Once Pellet reasoned this instance assignment, it returns with inconsistent individuals. This is because an instance cannot belong to two or more disjoint classes.

Therefore, if the instance of class room would be assigned similar to the instances in lecture hall, thus the disjointness of two classes should be removed. This is to reflect the situation at CIS Department that both classes room and lecture hall are allocated for lecturing. The result of instantiation checking is visualized by Swoop as depicted in Figure 3.16.

OWL-Class: ClassRoom	OWL-Class: LectureHall	OWL Ontology: cdf (Edit URI)
Subclass of: (Add)	Subclass of: (Add)	Annotations: (Add)
Room (Delete)	Room (Delete)	Imports: (Add)
Instances: (Add)	Instances: (Add)	Inconsistent ontology
C06 (Delete)	LH4 (Delete)	Reason: Individual D04 is forced to belong to class
D01 (Delete)	LH3 (Delete)	LectureHall
C05 (Delete)	C05 (Delete)	and its complement
D04 (Delete)	LH5 (Delete)	Axioms causing the problem:
D05 (Delete)	D04 (Delete)	1) (D04 rdf:type ClassRoom)
C02 (Delete)	D02 (Delete)	2) $\exists (ClassRoom \sqsubseteq \neg LectureHall)$
C03 (Delete)	C01 (Delete)	3) $\exists (D04 \text{ rdf:type } LectureHall)$
D02 (Delete)	D03 (Delete)	
C01 (Delete)	LH2 (Delete)	
D03 (Delete)	LH1 (Delete)	
C04 (Delete)	C06 (Delete)	
D06 (Delete)	D01 (Delete)	
	D05 (Delete)	
	C03 (Delete)	
	C02 (Delete)	
	LH6 (Delete)	
	C04 (Delete)	
	D06 (Delete)	

Figure 3.16: Instance Definition (left). Inconsistency Detected (right)

From the modeling point of view, nominal is used to describe enumeration of membership of a class. Peter F. Patel-Schneider et al. in OWL DL W3C Reference Standard [33] define that the OWL DL or *SHOIN* contains two modeling constructs specific for nominal, namely *owl:oneOf* and *owl:hasValue*. The *owl:oneOf* construct allows defining finite enumeration of elements in a concept or class. In this case, the individuals of class *Browser* is declared with the type of browser applications. By using DLs notation, individuals in class *Browser* can be written as follow: $Browser \equiv \{FIREFOX, MOZILLA, IE, SAFARI\}$. The OWL semantic web notation to express the same enumeration above is presented follows.

```
<owl:Class rdf:about="#Browser">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#InternetApplication"/>
  </rdfs:subClassOf>
  <owl:oneOf rdf:parseType="Collection">
    <cdf:Browser rdf:about="#Firefox"/>
    <cdf:Browser rdf:about="#IE"/>
    <cdf:Browser rdf:about="#Mozilla"/>
    <cdf:Browser rdf:about="#Safari"/>
  </owl:oneOf>
</owl:Class>
```

The *owl:hasValue* is OWL construct used in an existential restriction on a nominal concept. Regarding the CIS context ontology, we define a class *Server* in such a way to restrict a person that has to login to Novel Netware server prior to use network

resources. This situation in which a *Person* must login to the *NovelNetware* server as the individual of class *Server* is declared in OWL as follow.

```
<owl:Class rdf:about="#Server">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Network"/>
  </rdfs:subClassOf>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:hasValue>
        <cdf:Server rdf:about="#NovelNetware"/>
      </owl:hasValue>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#login"/>
      </owl:onProperty>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

3.5 Chapter Summary

The main issues in this chapter are summarized as follows.

1. This chapter explains the modeling of context ontology using Description Logics notation and OWL semantic web language. It shows that DLs notation are more expressive than OWL semantic web model, context model in DLs notation can directly be generated for implementation language, like OWL semantic web. It is because OWL semantic web is fully supported by DLs semantics. Our OWL context ontology is generic; hence it can be modified or adjusted depending on the user's needs.
2. It is shown that DLs notation of context ontology is built on top of formal or mathematical model. By describing context ontology in DLs notation, we actually provide a context specification that is independently from the implementation language level. Nevertheless, many researchers are concerned with OWL, therefore they are focusing on developing DL reasoner that is based-on OWL semantic web language instead of developing automated reasoning tool based-on DLs notation.

In the next chapter, the use of Z formal specification to construct context ontology will be presented. By using formal specification, hopefully the context ontology can be expressed independently from OWL semantic web format. The Z specification is fully supported by Z/EVES automatic theorem prover.

Chapter 4

Z Specification of Context Model

This chapter begins with the description of mapping process to generate context ontology in Z formal specification. Thereafter, a process of how to express OWL semantics in Z semantics, how to map OWL context ontology onto Z notation, and how to perform semantic checking of context ontology in Z environment are presented, respectively.

4.1 Mapping Process

In the previous chapter, context ontology model is prepared in OWL semantic web language. In this chapter, the use Z specification language to address the formal specification of context ontology will be presented. The process of mapping OWL semantic web of context ontology onto Z specification is illustrated in Figure 4.1.

The Z syntaxes and semantics for OWL semantic web have been defined in [13]-[15]. In this thesis, the semantics are rewritten by taking from OWL W3C semantic theoretic [33], which are to define the semantics of Z syntax for each particular OWL language. For this purpose, this thesis use the term of OWL-Z to express the Z syntaxes and semantics for OWL language. Either Z syntax or Z semantics are prepared in \LaTeX format(see box no ①) in order to be parsed by Z/EVES tool. This is because Z/EVES read \LaTeX format as input for specification and proofing process. The following Table 4.1 briefly describes the OWL W3C abstract syntax and its corresponding Z syntax used to define ontology in Z specification.

Once the OWL-Z notation has been type-checked and semantically proved, thus, the semantic web of context ontology which has been prepared in OWL can then be mapped onto Z specification by referring to OWL-Z syntax. Now, the context ontology structure is presented in Z. As the result of mapping process, the Z notation

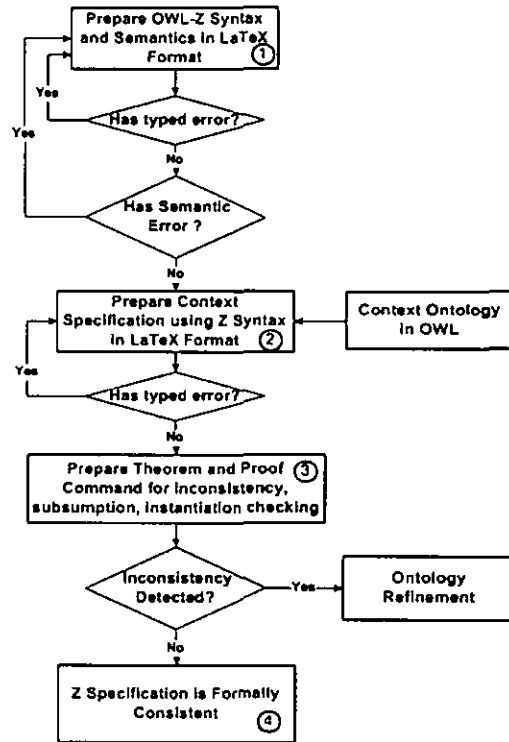


Figure 4.1: Process of Generating and Checking of Context Model in Z Formal Specification

of context ontology should be prepared in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ format (see box no ②). Once the context ontology has been written in Z, the type checking to detect the trivial syntax error should then be prepared.

The further step is to prepare the rule and proof/test command, i.e for inconsistency checking purpose (see step ③). In this step, some assumption rule and defined theorem will be used to prove the Z specification. Once the specification of context ontology is proved by Z/EVES, and it returns with true, it means that our context Z specification of context ontology is formally consistent (see box no④). Otherwise, once the inconsistency source has been discovered, it means that the specification of context ontology in OWL semantic web has to be redefined to remove errors that have been detected by Z/EVES. The inconsistency is detected because the current OWL DL reasoner previously might not able to detect the logical inconsistency in the semantic web model. Thus, to conclude, by mapping OWL definition of context ontology and performing semantic checking in Z/EVES, this thesis has use formal specification technique as the complementary approach to design and verify context ontology.

Table 4.1: OWL Syntax and Z Syntax

OWL Abstract Syntax	Z Syntax
<i>subClassOf</i> (C_1, C_2)	<i>subClassOf</i> (c_1, c_2)
<i>disjointWith</i>	<i>disjointWith</i> (c_1, c_2)
<i>intersectionOf</i> (C_1, C_2)	<i>intersectionOf</i> (c_1, c_2)
<i>unionOf</i> (C_1, C_n)	<i>unionOf</i> (c_1, c_2)
<i>complementOf</i> (C)	$(c_1, c_2) \in \text{complementOf}$
<i>oneOf</i> ($o_1 \dots o_n$)	<i>oneOf</i> (X) = c_1
<i>restriction</i> ($R \text{ allValuesFrom}(C)$)	<i>allValuesFrom</i> (c_1, R) = c_2
<i>restriction</i> ($R \text{ someValuesFrom}(C)$)	<i>someValuesFrom</i> (c_1, R) = c_2
[<i>Transitive</i>]	$(R) \in \text{Transitive}$
[<i>Symmetric</i>]	$(R) \in \text{Symetric}$
[<i>inverseOf</i> (R_o)]	$(R_1, R_2) \in \text{inverseOf}$
<i>restriction</i> ($C \text{ maxCardinality}(n)$)	<i>maxCardinality</i> (n, R) = c
<i>restriction</i> ($C \text{ minCardinality}(n)$)	<i>minCardinality</i> (n, R) = c
<i>restriction</i> ($C \text{ Cardinality}(n)$)	<i>Cardinality</i> (n, R) = c

4.2 Z Syntax and Semantics (OWL-Z)

Regarding the OWL semantics, everything is a model of resource. DLs models this kind of resource as interpretation domain, or Δ^I . To express this interpretation domain, the basic Z type definition is used as follows.

[DELTA]

As in DLs *SHOIN* semantics, the OWL-Z semantics model basically define the meaning and interpretation of concept (*Class*), role (*Property*), and *Individual*.

A class provides a mechanism to group instances with similar characteristics. Therefore, every class is associated with a set of individuals, called the *class extension or class instance*. In DLs, a class is, or atomic class, is a member of domain interpretation. The semantic of an atomic class in DLs is expressed as $C^I \subseteq \Delta^I$.

Role or property is also defined as subset of interpretation domain. In DLs semantics, a property is defined as *cross product of interpretation domain*, expressed as $R^I \subseteq \Delta^I \Delta^I$.

In DLs semantics, individual is also defined as subset of interpretation domain. DLs defines individual as the power set of all instances exist in interpretation domain Δ^I . The semantic of individual is expressed as $a \in C^I$.

Those syntaxes and semantics definition above are prepared in \LaTeX format. This format is further parsed by Z/EVES tool for type and semantics checking. Z/EVES command *prove by reduce* is further defined, which is used to check the semantics of

our Z specification.

```
\begin{axdef}
Class: \power DELTA
Property: \power DELTA
Individual: \power DELTA
\where
Property \cap Class = \emptyset
Property \cap Individual = \emptyset
Individual \cap Class = \emptyset
\end{axdef}
```

proof

prove by reduce



In this thesis, Z/EVES style is used to render the \LaTeX format. Thus, upon rendering the \LaTeX format, Z specification becomes readable for human. For example, the above definition of class, property, and individual in OWL-Z are rendered as follows:

$Class : \mathbb{P} DELTA$ $Property : \mathbb{P} DELTA$ $Individual : \mathbb{P} DELTA$
$Property \cap Class = \emptyset$ $Property \cap Individual = \emptyset$ $Individual \cap Class = \emptyset$

We use *instances* syntax to map a class with class extension (instances).

$instances : Class \rightarrow \mathbb{P} Individual$

To describe a property concerned, or value of a property, either as Object Property or Datatype Property, individual has to be defined by mapping it as a property, either object property (*propval*) or data type property (*propvalD*). For instance, a and b are Individuals, p is a property, and p relates a with b , such that a and b are the property concerned of p , or formally $(a, b) \in R^I$. Further, in Z specification such property values are declared as $(a, b) \in propval(p)$.

$$\left| \begin{array}{l} \text{propval} : \text{ObjectProperty} \rightarrow (\text{Individual} \leftrightarrow \text{Individual}) \end{array} \right.$$

$\left[\begin{array}{l} \text{propvalD} : \text{DatatypeProperty} \rightarrow (\text{Individual} \leftrightarrow \text{XSD}) \end{array} \right]$

4.2.1 Class Description

Class axioms typically contain additional components that state necessary and/or sufficient characteristics of a class. Regarding to OWL W3C Document, there are three syntaxes for combining class descriptions into class axioms as follows:

1. *subClassOf*. If a class description c is defined as a subclass of another class description d , then the set of individuals in the class extension of c should be a subset of the set of individuals in the class extension of d . DLs semantic of this statement is $c^{\mathcal{I}} \subseteq d^{\mathcal{I}}$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for the *subClassOf* statement is declared as follow.

$$\left| \begin{array}{l} \text{subClassOf} : \text{Class} \leftrightarrow \text{Class} \\ \hline \forall c, d : \text{Class} \bullet \\ (c, d) \in \text{subClassOf} \Leftrightarrow \text{instances}(c) \subseteq \text{instances}(d) \end{array} \right.$$

2. *equivalentClass*. The two class descriptions involved have the same set of individuals. DLs semantic of this statement is $c^{\mathcal{I}} \equiv d^{\mathcal{I}}$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for the *equivalentClass* statement is declared as follows.

$$\left| \begin{array}{l} \text{equivalentClass} : \text{Class} \leftrightarrow \text{Class} \\ \hline \forall c, d : \text{Class} \bullet (c, d) \in \text{equivalentClass} \Leftrightarrow \\ \text{instances}(c) = \text{instances}(d) \end{array} \right.$$

3. *disjointWith*. This statement asserts that the class extension of the two class descriptions involved have no individuals in common. OWL abstract syntax of this statement is $disjointWith(c, d)$, and semantic of this statement is $c^I \cap d^I = \emptyset$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for the *disjointOf* class statement is declared as follows.

$$\begin{array}{|l} disjointWith : Class \leftrightarrow Class \\ \hline \forall c, d : Class \bullet \\ (c, d) \in disjointWith \Leftrightarrow \\ instances(c) \cap instances(d) = \emptyset \end{array}$$

4.2.2 Properties

OWL distinguishes between two main categories of properties. First is object property that relates individual of a class with individuals in another class. Second is data type property that relates individual of a class with data values that refers to XML Schema Data type definition (XSD). Object property and data type property are declared in OWL-Z as follows:

$$\begin{array}{|l} ObjectProperty : \mathbb{P} Property \\ DatatypeProperty : \mathbb{P} Property \\ \hline ObjectProperty \cap DatatypeProperty = \emptyset \end{array}$$

In OWL, *subpropertyOf* reflects that a property is a sub property of another property. Formally this means that if p_1 is a subproperty of p_2 , then the property concerned (property value or extension) of p_1 should be a subset of the property concerned p_2 . DLs semantic of sub property statement is $\{a \in \Delta^I | \forall b \in R^I \rightarrow (a, b) \in S^I\}$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for *subpropertyOf* statement is declared as follows.

[XSD]
$subPropertyOf : Property \leftrightarrow Property$
$\forall r, s : Property \bullet (r, s) \in subPropertyOf \Leftrightarrow$ $(r \in ObjectProperty \wedge s \in ObjectProperty \Rightarrow propval(r) \subseteq propval(s)) \wedge$ $(r \in DatatypeProperty \wedge s \in DatatypeProperty \Rightarrow propvalD[XSD](r) \subseteq propvalD[XSD](s))$

Another OWL property statement, i.e. *equivalentProperty*, is used to state that two properties have the same property concerned (property value). OWL syntax of this statement is $equivalentProperty(c, d)$, and DLs semantic of this statement is $\{a \in \Delta^I \mid \forall b \in R^I \Leftrightarrow (a, b) \in S^I\}$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for *equivalentProperty* is declared as follows.

[XSD]
$equivalentProperty : Property \leftrightarrow Property$
$\forall r, s : Property \bullet (r, s) \in equivalentProperty \Leftrightarrow$ $(r \in ObjectProperty \wedge s \in ObjectProperty$ $\Rightarrow propval(r) = propval(s)) \wedge (r \in DatatypeProperty \wedge s$ $\in DatatypeProperty \Rightarrow propvalD[XSD](r) = propvalD[XSD](s))$

Properties have a direction, from domain to range. In practice, people often find it useful to define relations in both directions: persons own cars, cars are owned by persons. Regarding this matter, OWL uses *inverseOf* syntax as an inverse relation function between properties. Formally, it can be said that p_1 is inverse of p_2 , thus it asserts that for every pair (x, y) in the property extension of p_1 , there is a pair (y, x) in the property extension of p_2 , and vice versa. DLs syntax of *inverseOf* statement is $R \equiv R_o^-$. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for *inverseOf* property is declared as follows.

$inverseOf : ObjectProperty \leftrightarrow ObjectProperty$
$\forall p1, p2 : ObjectProperty \bullet (p1, p2) \in inverseOf \Leftrightarrow$ $propval(p1) = (propval(p2))^\sim$

In OWL, a property is defined as being transitive by making use of OWL class *TransitiveProperty* syntax. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for *Transitive* property is declared as follows.

$$\begin{array}{|l}
 \hline
 \textit{Transitive} : \mathbb{P} \textit{ObjectProperty} \\
 \hline
 \forall \textit{prop} : \textit{ObjectProperty} \bullet \textit{prop} \in \textit{Transitive} \Leftrightarrow \\
 (\forall x, y, z : \textit{Individual} \bullet (x, y) \in \textit{propval}(\textit{prop}) \wedge \\
 (y, z) \in \textit{propval}(\textit{prop}) \Rightarrow (x, z) \in \textit{propval}(\textit{prop}))
 \end{array}$$

A symmetric property is a property for which holds that if the pair (x, y) is an instance of property P , then the pair (y, x) is also an instance of P . The domain and range of a symmetric property are the same. From the OWL abstract syntax and DLs semantics, the OWL-Z syntax and semantic for *Symetric* property is declared as follows.

$$\begin{array}{|l}
 \hline
 \textit{Symetric} : \mathbb{P} \textit{ObjectProperty} \\
 \hline
 \forall \textit{prop} : \textit{ObjectProperty} \bullet \textit{prop} \in \textit{Symetric} \Leftrightarrow (\forall x, y : \textit{Individual} \bullet (x, y) \\
 \in \textit{subVal}(\textit{prop}) \Rightarrow (y, x) \in \textit{subVal}(\textit{prop}))
 \end{array}$$

4.2.3 Value Constraint

A property can also be restricted by constraints. OWL distinguishes two kinds of property restrictions: *value constraints* and *cardinality constraints*.

The value constraint *allValuesFrom* is an OWL statement that relates a restriction class to either a class description or a data range. Formally, it defines individual x for which holds that if the pair (x, y) is a value of R (the property concerned), then y should be an instance of the class description (or a value in the data range for data type property). DLs semantics of this value restriction is $a \in \Delta^{\mathcal{I}} | \forall b. (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}$. The OWL-Z syntax and semantics of this *allValuesFrom* property statement are declared as follows.

$$\overline{\text{allValuesFrom} : \text{Class} \times \text{ObjectProperty} \rightarrow \text{Class}}$$

$$\forall c, d : \text{Class}; p : \text{ObjectProperty} \bullet \text{allValuesFrom}(c, p) = d \Leftrightarrow$$

$$\text{instances}(d) = \{a : \text{Individual} \mid \forall b : \text{Individual} \bullet$$

$$(a, b) \in \text{propval}(p) \Rightarrow b \in \text{instances}(c)\}$$

The value constraint *someValuesFrom* is a OWL property that relates a restriction class to a class description (or a data range for data type property). Formally, it defines individual x for which there is at least one y (either an instance of the class description or value of the data range) such that the pair (x, y) is value of R . DLs semantics of this value restriction is $a \in \Delta^{\mathcal{I}} \mid \exists b. (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}$. The following are OWL-Z syntax and semantic for *someValuesFrom* property statement.

$$\overline{\text{someValuesFrom} : \text{Class} \times \text{ObjectProperty} \rightarrow \text{Class}}$$

$$\forall c, d : \text{Class}; p : \text{ObjectProperty} \bullet \text{someValuesFrom}(c, p) = d \Leftrightarrow$$

$$\text{instances}(d) = \{a : \text{Individual} \mid \exists b : \text{Individual} \bullet$$

$$(a, b) \in \text{propval}(p) \wedge b \in \text{instances}(c)\}$$

The value constraint *hasValue* is an OWL property that relates a restriction class to a value V , which can be either an individual or a data value. DLs semantic of this property statement is $a \in \Delta^{\mathcal{I}} \mid \exists b. (a, b) \in R^{\mathcal{I}}$. The following are OWL-Z syntax and semantics for *hasValue* property statement.

$$\overline{\text{hasValue} : (\text{Individual} \times \text{ObjectProperty}) \rightarrow \text{Class}}$$

$$\forall \text{ind} : \text{Individual}; c : \text{Class}; p : \text{ObjectProperty} \bullet$$

$$\text{hasValue}(\text{ind}, p) = c \Leftrightarrow \text{instances}(c) =$$

$$\{a : \text{Individual} \mid \text{ind} \in \text{propval}(p) \mid \{a\} \}$$

The cardinality constraint *maxCardinality* constraint describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. DLs Semantics of this cardinality statement is $a \in \Delta^{\mathcal{I}} \mid \{b \in \Delta^{\mathcal{I}} \mid (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n$. OWL-Z syntax and semantics for this property statement is declared as follows:

$$\begin{array}{|l}
\hline
maxCardinality : (\mathbb{N} \times ObjectProperty) \rightarrow Class \\
\hline
\forall c : Class; n : \mathbb{N}; p : ObjectProperty \bullet maxCardinality(n, p) = c \Leftrightarrow \\
instances(c) = \{x : Individual \mid \#\{(propval(p)\{x\})\} \leq n\}
\end{array}$$

Another cardinality constraints are *minCardinality* and *Cardinality*, which are almost the same meaning (semantics) with *maxCardinality*, except the number of *N* as constraint values.

4.2.4 Individual

The OWL syntax *sameAs* links an individual of a class to an individual of another class. This statement indicates that two individuals have the same identity. OWL-Z syntax and semantic of *sameAs* statement are declared as follows.

$$\begin{array}{|l}
\hline
sameAs : \mathbb{P} Individual \leftrightarrow \mathbb{P} Individual \\
\hline
\forall x, y : \mathbb{P} Individual \bullet (x, y) \in sameAs \Leftrightarrow x = y
\end{array}$$

Like *sameAs*, the OWL *differentFrom* statement links an individual to an individual. However, this statement indicates that two individuals have different identity. OWL-Z syntax and semantic of *differentFrom* statement are declared as follows.

$$\begin{array}{|l}
\hline
differentFrom : \mathbb{P} Individual \leftrightarrow \mathbb{P} Individual \\
\hline
\forall x, y : \mathbb{P} Individual \bullet (x, y) \in differentFrom \\
\Leftrightarrow x \neq y
\end{array}$$

4.3 Mapping Context Ontology onto Z Notation

This section presents the mapping of OWL semantic web context ontology onto Z notation. To generate context ontology in Z notation, this thesis uses the rewritten OWL-Z, which has been defined in the previous section. The overall specification of context ontology will not be discussed in this section, the complete specification is provided in the Appendix D.

As in the OWL semantic web version, context ontology consists of *Person*, *Network*, *Activity*, *Device*, *Network*, and *Location* as main concepts. Every classes defined in OWL semantic web are sub class of *Thing* (or \top in DLs). Those classes are modeled in Z using axiomatic box as follows.

<i>Person, Network,</i> <i>Activity, Location, Device : Class</i>
<i>(Person, Thing) ∈ subClassOf</i>
<i>(Network, Thing) ∈ subClassOf</i>
<i>(Device, Thing) ∈ subClassOf</i>
<i>(Activity, Thing) ∈ subClassOf</i>
<i>(Location, Thing) ∈ subClassOf</i>

4.3.1 Specification of Class Person and Its Related Property

As in OWL semantic web version of CIS context model, class *Person* is composed of lecturer, staff, post graduate student, and undergraduate student. Z axiomatic box is used to declare all classes since the dynamic context model is not to be a concern in this thesis. Some assumption rule labels are defined well, e.g. as indicated by $\langle\langle\textit{grule LecturerInPerson}\rangle\rangle$. The purpose of this assumption rule is to be used (re-called) later with command to test the consistency of the axioms (declared with test command).

Lecturer, Student, Postgrad, Staff, Profile : Class

```
⟨⟨ grule StudentInPerson ⟩⟩  
(Student, Person) ∈ subClassOf  
⟨⟨ grule LecturerInPerson ⟩⟩  
(Lecturer, Person) ∈ subClassOf  
⟨⟨ grule PostgradInPerson ⟩⟩  
(Postgrad, Person) ∈ subClassOf  
⟨⟨ grule StaffInPerson ⟩⟩  
(Staff, Person) ∈ subClassOf  
⟨⟨ grule ProfileofStaff ⟩⟩  
(Profile, Staff) ∈ subClassOf  
⟨⟨ grule ProfileofLecturer ⟩⟩  
⟨⟨ grule ProfileInLecturer ⟩⟩  
(Profile, Lecturer) ∈ subClassOf  
⟨⟨ grule ProfileofStudent ⟩⟩  
(Profile, Student) ∈ subClassOf  
⟨⟨ grule ProfileofPostgrad ⟩⟩  
(Profile, Postgrad) ∈ subClassOf
```

Person's related object properties are declared in Z notation using Z axiomatic box. Because object property links a class with another class, therefore its domain and range have to be determined as well. Some assumption rules are introduced in this specification. The following Z axiomatic box shows a part of specification of object properties related to class *Person*.

```

use, run, connectedTo, currentActivity, locatedIn,
loginTo, locatedIn,
ownedBy : ObjectProperty

domain(use) = Person
range(use) = Device
domain(run) = Person
range(run) = ApplicationRun
domain(connectedTo) = Person
range(connectedTo) = Device
domain(connectedTo) = Device
range(connectedTo) = Network
domain(connectedTo) = Person
range(connectedTo) = Network
...
« grule runSubProp »
(run, use) ∈ subPropertyOf
« grule usesIsTransitive »
(connectedTo) ∈ Transitive
« grule ownedByIsInverse »
(use, ownedBy) ∈ inverseOf
...

```

Regarding the specification of class *Person* related properties, three properties that determine the expressiveness of Z specification of context ontology model have been declared. Axiom $(run, use) \in subPropertyOf$ determines the hierarchy of properties, or labeled with \mathcal{H} in DLs. Axiom $(connectedTo) \in Transitive$ determines that this property is transitive, or labeled with \mathcal{S} in DLs. The label \mathcal{I} in DLs language is determined by inverse role axiom $(use, ownedBy) \in inverseOf$.

Data type properties related to class *Person* can also be specified in Z notation. Actually Z has no specific data type definition, such as to express string, date, integer, etc. By referring to OWL definition of XSD data type for semantic web, a new free type definition, i.e. [XSD], is issued to express data type in Z specification of context model ontology. Data type property is used to relate instances of a class with literal.

For example, the Z specification to relate data type properties in class *Profile* with a data type is written as follows.

```

[XSD]
-----
fullName, officeAddress, phoneNumber,
emailAddress, imAddress : DatatypeProperty
name, office, phone, email, im : P XSD

domain(fullName) = Profile
rangeD(fullName) = name
domain(officeAddress) = Profile
rangeD(officeAddress) = office
domain(phoneNumber) = Profile
rangeD(phoneNumber) = phone
domain(emailAddress) = Profile
rangeD(emailAddress) = email
domain(imAddress) = Profile
rangeD(imAddress) = im

```

Let us take an example. Axiom $\text{domain}(imAddress) = Profile$ determines the domain of $imAddress$ property. This property is used to relate class *Profile* with the literal of person instant messenger address, e.g. anybody@yahoo.com. The axiom $\text{rangeD}(imAddress) = im$ describes that the range of property $imAddress$ is literal im with common data type namely XSD. As in the implementation language, such as in OWL, the XSD can further be defined as string, or character. However, in this formal specification of context model, there is no need a detail or specific of data type in the property value, since data type is considered not to affect the whole consistency of context ontology model.

4.3.2 Specification of Class Device

A part of Z specification of class device is discussed in this subsection. As in OWL semantic web model, the three distinct devices used by a person in CIS context model are declared as well. Subclasses of device are also declared in this axiomatic box. Assumption rule ($\langle\langle\text{grule HardwareSoftwareDisjoint}\rangle\rangle$) is declared to assert class disjointness definition $(Hardware, Software) \in disjointWith$ in the command for testing consistency of axioms.

```

Desktop, MobileDevice, NetworkDevice,
Hardware, Software, MobilePhone, Notebook, PDA,
AccessPoint, Router, Server, ... : Class


---


(Desktop, Device) ∈ subClassOf
(MobileDevice, Device) ∈ subClassOf
(NetworkDevice, Device) ∈ subClassOf
(Notebook, MobileDevice) ∈ subClassOf
(PDA, MobileDevice) ∈ subClassOf
(MobilePhone, MobileDevice) ∈ subClassOf
(AccessPoint, NetworkDevice) ∈ subClassOf
(Server, NetworkDevice) ∈ subClassOf
(Router, NetworkDevice) ∈ subClassOf
...
⟨⟨ grule HardwareSoftwareDisjoint ⟩⟩
(Hardware, Software) ∈ disjointWith...

```

4.3.3 Specification of Class Activity

Like in the OWL semantic web of context model, activities related to a person are declared as *Planned* and *Deduced*. The specification of both deduced and planned activities are declared using Z axiomatic box. In Chapter 3 Figure 3.11, Planned and Deduced have been defined to be disjoint each other.

```

Planned, Deduced, Available, Busy, Free, ... : Class


---


(Planned, Activity) ∈ subClassOf
(Deduced, Activity) ∈ subClassOf
⟨⟨ grule PlannedRule ⟩⟩
(Deduced, Planned) ∈ disjointWith
...
(Available, Deduced) ∈ subClassOf
(Free, Deduced) ∈ subClassOf
(Busy, Deduced) ∈ subClassOf
⟨⟨ grule BusyFreedisjointWith ⟩⟩
(Busy, Free) ∈ disjointWith
...

```


Disjointness restriction is also used during the consistency checking in Chapter 3 Section 3.4.1. For the purpose of testing the class disjointness between *Planned* and *Deduced* in Z specification, the assumption rule label $\langle\langle\text{grule PlannedRule}\rangle\rangle$ is defined. Another assumption rule is also defined, i.e. $\langle\langle\text{grule BusyFreedisjointWith}\rangle\rangle$ that is to test the disjointness between class *Busy* and *Free*.

4.3.4 Specification of Class Location

Location context model are declared in Z specification by distinguishing indoor location and outdoor location, as the with OWL semantic web model. Class *Outdoor* and *Indoor* is also declared disjointness each other. For the purpose of testing the class disjointness between *Outdoor* and *Indoor* in Z specification, the assumption rule label $\langle\langle\text{grule OutDoorIndoorDisjoint}\rangle\rangle$ is issued to test the disjointness between class *Indoor* and *Outdoor*.

<i>Indoor, Outdoor, Building, Room, Classroom, LectureHall, OfficeRoom, ... : Class</i>
<i>(Indoor, Location) ∈ subClassOf</i> <i>(Outdoor, Location) ∈ subClassOf</i> $\langle\langle\text{grule OutDoorIndoorDisjoint}\rangle\rangle$ <i>(Indoor, Outdoor) ∈ disjointWith</i> <i>(Building, Indoor) ∈ subClassOf</i> <i>(Room, Building) ∈ subClassOf</i> <i>(Lab, Room) ∈ subClassOf</i> <i>(ClassRoom, Room) ∈ subClassOf</i> <i>(LectureHall, Room) ∈ subClassOf</i> ...

The complete mapping from OWL semantic theoretic onto Z syntax and semantics is provided in the Appendix C.

4.3.5 Specification of Class and Property Constraint

In Chapter 3, the activity of *Busy* is sub classes of *Deduced*. This class is declared to describe an activity in which a person is busy, by assuming he is running the office application, e.g. word processor application while he is located at his office room, or

he is doing a planned activity. This class *Busy* is restricted with axioms:

$$\begin{aligned} \text{Busy} &\equiv \text{Person} \sqcap \forall \text{run}. \text{OfficeApplication} \sqcap \forall \text{located}. \text{OfficeRoom} \\ \text{Busy} &\equiv \text{Person} \sqcap \forall \text{currentActivity}. \text{Planned} \end{aligned}$$

To express class *Busy* restriction in Z specification, the Z axiomatic box can be issued as follow.

$$\left| \begin{array}{l} \dots \\ \text{Busy} = \text{someValuesFrom}(\text{Person}, \text{run}) = \text{OfficeApplication} \wedge \\ \text{allValuesForm}(\text{Person}, \text{loctedIn}) = \text{OfficeRoom} \\ \text{Busy} = \text{someValuesFrom}(\text{Person}, \text{currentActivity}) = \text{Planned} \\ \dots \end{array} \right.$$

Another restriction that are defined in OWL semantic web language of the context ontology is cardinality restriction, which describes a class of all individuals that have at most N semantically distinct values (individuals or data values) for the property concerned.

As defined in OWL semantic web language in Chapter 3, for example, a person can only have one activity at a certain time (either doing planned activity or deduced activity), cardinality restriction $\equiv 1.\text{currentActivity}$ is used to restrict property *currentActivity*. The Z specification of this property restriction is declared in Z axiomatic box as follow.

$$\left| \begin{array}{l} \dots \\ \text{Cardinality}(1, \text{currentActivity}) = \text{Person} \\ \text{maxCardinality}(1, \text{run}) = \text{ApplicationRun} \\ \dots \end{array} \right.$$

The above Z axiomatic box also defines a cardinality restriction on property *run*, that restricts a person is able to run at least 2 application software on that computer (including operating system).

4.3.6 Specification of Individuals

In OWL semantic web language, *owl:OneOf* is a syntax used to define enumerated instances of a class. In OWL-Z specification, *oneOf* can also be issued to define the memberships of a concept or a class. For example, class *ClassRoom* is defined to describe room entities used in lecturing activity. CIS context model define a classroom into class name, e.g. C01,C02. Name of the classes also describes a room located in Block C and D in our university. Some related instances of *ClassRoom* are defined in Z specification as follows.

```

C01, C02, C03, C04, C05, C06, D01, D02, D03, D04, D05, D06 : Individual
-----
C01 ∈ instances(ClassRoom);
C02 ∈ instances(ClassRoom);
C03 ∈ instances(ClassRoom);
...
D05 ∈ instances(ClassRoom); D06 ∈ instances(ClassRoom);

```

4.4 Checking Z Specification of Context Ontology

Once the ontology has been written in formal specification language, there is a need to verify such specification whether conform to a given property. Further, this thesis follows the previous works the way how to reason the ontology beyond the existing semantic web reasoner, as described in [14]-[17].

4.4.1 Consistency Checking

In this section, the demonstration of verification of context ontology model beyond the semantic web reasoner is presented. The intention of verification is to explore the undetected inconsistent class with respect to Definition 3.1 in Chapter 3. In Chapter 3, Pellet OWL DL reasoner is already used to detect the unsatisfiable concepts of contest ontology model. The reasoner concludes that the OWL version of context ontology model is consistent, though it does not satisfies the Definition 3.1.

After declaring the Z specification of class *Indoor* and *Outdoor*, a rule label in the specification, i.e. $\langle\langle\text{gruleOutDoorIndoorDisjoint}\rangle\rangle$, is issued to be used by Z/EVES during the proof process. Following is Z specification of class *Indoor* and *Outdoor*.

$Indoor, Outdoor : Class$ <hr style="width: 100%;"/> $(Indoor, Location) \in subClassOf$ $(Outdoor, Location) \in subClassOf$ $\langle\langle \text{grule } OutDoorIndoorDisjoint \rangle\rangle$ $(Indoor, Outdoor) \in disjointWith$	
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

The rule label is also put in the specification of class *Person* and value restriction of property *locatedIn*. The rule label is declared as follows.

$\dots, Person, \dots : Class$ <hr style="width: 100%;"/> $\dots, locatedIn, \dots : ObjectProperty$ \dots $\langle\langle \text{grule } PersonLocatedInIndoor \rangle\rangle$ $allValuesFrom(Person, locatedIn) = Indoor$	
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--

The Definition 3.1 in Chapter 3 is expressed in Z theorem that will be used to guard Z axioms used during proof process. The Definition 3.1 is written in Z theorem as follows.

theorem grule allvaluedisjointrule

$$\forall c, d, e : Class; p : Property \bullet (d, e) \in disjointWith \wedge$$

$$allValuesFrom(c, p) = d \Rightarrow \neg (allValuesFrom(c, p) = e)$$

To test the inconsistency of the above definition, the following goal should be issued as follows: $try((allValuesFrom(Person, locatedIn) = Indoor) \Rightarrow (allValuesFrom(Person, locatedIn) = Outdoor))$. Our goal is to prove that property *locatedIn* will be applied in the disjoint class that are in the range property concerned. The proof command to test the axiom should be prepared in \LaTeX script, and the sequence of Z proof command are issued as follows:

proof

```

try allValuesFrom(Person, locatedIn) = Outdoor;
use OutDoorIndoorDisjoint;
use PersonLocatedInIndoor;
use allvaluedisjointrule
[c := Person, d := Indoor, e := Outdoor, p := locatedIn];
prove by reduce;

```

■

The first command (*try*) is the goal to test, second command (*use*) is to recall the assumption rule to assert that class *Indoor* and *Outdoor* are disjoint each other, the last command *reduce* is to let Z/EVES to perform simplification, rewriting, and reduce the goal. The testing result of Z/EVES in L^AT_EX mode interface has also been captured and provided in the Appendix. The result of testing (or reasoning) of consistency is presented in the following lines (non rendered L^AT_EX scripts).

Beginning proof of ...

```

allValuesFrom(Person, locatedIn) = Indoor
⇒ allValuesFrom(Person, locatedIn) = Outdoor
Assuming OutDoorIndoorDisjoint generates...
(Indoor, Outdoor) ∈ disjointWith
∧ allValuesFrom(Person, locatedIn) = Indoor
⇒ allValuesFrom(Person, locatedIn) = Outdoor
Substituting allValuesFrom(Person, locatedIn) = Indoor produces...
(Indoor, Outdoor) ∈ disjointWith
∧ allValuesFrom(Person, locatedIn) = Indoor
⇒ Indoor = Outdoor

```

...

Proving gives...

```

Location = Indoor
⇒ Indoor = Outdoor

```

Z/EVES returns with *Indoor = Outdoor* (see Appendix E Figure E.1). This means that the goal contains a contradiction. This is because previously Planned and deduced are defined to be disjoint each other. Regarding to DLs semantics, value restriction defines individual of a class *Indoor* for which holds that if the pair (x, y) is the value of property *locatedIn* (property concerned), then y should be an instance of the class *Indoor*. Since *Indoor* is disjoint with *Outdoor*, hence the the value of property *locatedIn*, should not be an instance of the class *Outdoor*. Regarding to

proof given by Z/EVES, our context model contains inconsistent class, hence class disjointness should be removed between class *Indoor* and *Outdoor* with respect to the property concerned of *locatedIn*. The preparation and process in Figure 5.1 can be repeated again.

4.4.2 Subsumption Checking

The task of subsumption checking is to infer that a class definition is sub class of another class, or to obtain the taxonomy of knowledge, such that $C \sqsubseteq D$ i.e. $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ iff $\mathcal{I} \models \mathcal{O}$, where \mathcal{O} is the ontology. In other words, subsumption checking discovers concept inclusion.

Previously, an entity *Person* is defined as a sub class of *Class*:

<i>Person, Network,</i> <i>Activity, Location, Device : Class</i>
<hr/> <i>(Person, Thing) ∈ subClassOf</i> ...

and a *Profile* entity is also declared as a sub class of *Lecturer*:

<i>Lecturer, Student, Postgrad, Staff, Profile : Class</i>
<hr/> <i>(Student, Person) ∈ subClassOf</i> <i>⟨⟨ rule LecturerInPerson ⟩⟩</i> <i>(Lecturer, Person) ∈ subClassOf</i> ... <i>⟨⟨ grule ProfileInLecturer ⟩⟩</i> <i>(Profile, Lecturer) ∈ subClassOf</i> ...

Thus, the goal is defined, i.e. to prove the inclusion that the class *Person* is super-class of class *Profile*. The two assumption rules are then recalled, and the command *prove by reduce* are then recalled as well to find out the solution.

proof

```

try (Profile, Person) ∈ subClassOf;
use LecturerInPerson;
use ProfileInLecturer;
reduce;

```

■

Having executed the prover command, Z/EVES concludes that $(Profile, Person) \in subClassOf$ (see Appendix E Figure E.2).

4.4.3 Instantiation Checking

Instantiation checking asserts that an individual is an instance of a class. It is demonstrated through an example that Z/EVES can also perform instantiation checking in Z specification of context model.

In the Z notation of context ontology specification, *NOVELNETWARE* is declared as an instance of class *Server*. This is to describe the situation in which a person has to login to this server first prior to using network resource in our department, such as accessing Intranet or Internet resource. Thus, the instance of *Server* is specified as follows:

<pre> NOVELNETWARE : Individual ----- <<grule ServerInstance >> NOVELNETWARE ∈ instances(Server) ... </pre>

To test the instance assignment of a class, the *try* command of Z/EVES is used, followed by *provebyreduce* command. Upon running Z/EVES to test this instance assignment, Z/EVES is able to detect that *NOVELNETWARE* is instance of *Server* concept, and it returns true (see Appendix E Figure E.3).

proof

```

try NOVELNETWARE ∈ instances(Server);
prove by reduce;

```

■

Another proof of instantiation reasoning will be presented as well. From the

given individual $D01$ is the instance of class $ClassRoom$. The assumption label rule, $\langle\langle D01 \text{ in } ClassRoom \rangle\rangle$ is also defined to test the consistency of the axiom later on during the proofing process. The Z specification of this instance $D01$ is given as follow:

$\dots, D01, \dots : Individual$ <hr style="width: 100%;"/> \dots $\langle\langle \text{grule } D01 \text{ in } ClassRoom \rangle\rangle$ $D01 \in instances(ClassRoom);$ $D02 \in instances(ClassRoom);$ \dots

Previously, specification of $ClassRoom$ and $Room$ entity should also be declared, and the rule label $\langle\langle ClassRoom \text{ in } Room \rangle\rangle$ is used to test the axiom during the proof process.

$\dots, ClassRoom, Room, \dots : Class$ <hr style="width: 100%;"/> $\langle\langle \text{grule } ClassRoom \text{ in } Room \rangle\rangle$ $(ClassRoom, Room) \in subClassOf$ $(OfficeRoom, Room) \in subClassOf$ \dots

Another definition to be used during the proof process needs to be issued as follows:

Definition 4.1. Let $c, d \in C$ be class name, $c \sqsubseteq d$, and $i \in \Delta^I$ be individual. If i is instance of c it implies that i is also instance of d or $i : d$.

The Definition 4.1 is then written in Z specification as Z theorem as follow.

theorem grule instancesubclass

$$\forall c, d : Class; ind : Individual \bullet (c, d) \in subClassOf \wedge ind \in instances(c) \Rightarrow ind \in instances(d)$$

The following goal needs to be issued as well: $D01 \in instances(Room)$. The intention is to test the inconsistency of the above (Z specification) definitions. The goal issued is to prove that if $D01$ belongs to $ClassRoom$, then it also belongs to its superclass, i.e. $Room$. The proof command, including sequence of Z proof command, to

test the axiom should be prepared in \LaTeX as follows:

proof

```

try D01 ∈ instances(Room);
use D01inClassRoom;
use ClassRoominRoom;
use instancesubclass[c := ClassRoom, d := Room, ind := D01];
prove by reduce;

```

■

and having proved such commands, Z/EVES returns true (see Appendix E Figure E.4)

Another example is also given i.e. to address the process of individual property reasoning with *hasValue* syntax. In the beginning of this section, it is known that *NOVELNETWARE* is the server than a person has to login prior to using the Network resource. The OWL semantic model to express such condition is declared as follows:

```

<owl:Class rdf:about="#Server">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NetworkDevice"/>
  </rdfs:subClassOf>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:hasValue>
        <cdf:Server rdf:about="#Netware"/>
      </owl:hasValue>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#loginTo"/>
      </owl:onProperty>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>

```

Either in the DLs model or OWL model of CIS context ontology, it is already defined that the class *Lecturer* is sub classes of class *Person*. It is required to know whether a lecturer has to login to the novel netware server if she/he wants to use the network resource. In Z specification, the above goal is established as follows: *try hasValue(Lecturer,loginTo) = NOVELNETWARE*. By issuing the test commands for

proofing purpose, Z/EVES returns the above goal to be true. The screenshot of proofing process is provided in Appendix E Figure E.5.

proof

```
try hasValue(Lecturer, loginTo) = NOVELNETWARE;
use LecturerInPerson;
use subclassHasValue;
[c := Lecturer, d := Person, p := loginTo, ind := NOVELNETWARE];
prove by reduce;
```

■

It is demonstrated that Z/EVES is able to check and reason instantiation, which means that instantiation checking of context ontology model have been performed beyond the semantic web reasoner, and all individuals are proved to be the membership of a class.

4.5 Chapter Summary

The main contribution of this chapter can be summarized as follows:

1. This chapter addresses the development of context ontology using Z notation. The context ontology is taken from the previous Chapter 3, and mapped onto Z notation by using the OWL-Z syntax and semantics (OWL-Z).
2. It is shown that the separation of modeling language to develop context ontology model has been addressed in this thesis. Modeling language for design / specification is distinguished from the modeling language for application run-time (or implementation) purpose. In another word, separation of modeling language also requires an alternative method to check / validate the model. Context ontology checking in this chapter has been performed beyond the current semantic web reasoning tool.
3. Previously, in Chapter 3, ontology checking is carried out in OWL semantic web environment. For the context ontology which is prepared in OWL format, semantic checking is carried out using Pellet, the OWL DL reasoner. For context ontology in Z formal notation, therefore, to validate the correctness of ontology, Z/EVES tool is used. The undetected error of concept in Chapter 3 could be discovered in Z/EVES environment, and the source of error could also been

displayed. It shows that Z/EVES (Z theorem prover) has the ability to perform ontology checking, the task that is usually done by semantic web reasoner.

The next chapter will be presenting the discussion on the process of developing context ontology using semantic web language and formal specification. It will be shown that formal specification technique is proposed as complementary technique to detect inconsistency of context ontology, thus the refinement process could take place upon detecting the inconsistency.

Chapter 5

Discussion

This chapter presents the discussion on the overall process of developing context ontology using semantic web language and formal specification. The reflection on the proposed methods ends the discussion on this chapter.

5.1 Context Development Process

As with the conducted survey in [1], the list of context modeling approaches are quite comprehensive. It was also observed that the further emerging approaches might exist in the following decades. To date, it can be concluded that the most promising method for context modeling is using ontology. However, this does not mean that the other approaches are unsuitable for ubiquitous computing environments.

In the previous context ontology modeling approach, as proposed in [4]-[3]-[5]-[7], they defined semantic web as the executable format or to be executed directly by application run-time (or for implementation level purpose). During the ontology development, they rely on the semantic web reasoner to check the correctness of context ontology being designed.

This thesis proposes a formal specification technique as a complementary approach to the semantic web ontology modeling. Figure 5.1 shows the context ontology development process presented in this thesis. Context requirement capturing (process ① in Figure 5.1), DLs representation and OWL semantic web definition (process ② in Figure 5.1) are presented in Chapter 3.

The context ontology development approach in this thesis leads to the use of formal specification technique (process ③ in Figure 5.1) that suits to check the correctness of ontology beyond the semantic web model. Mapping process to generate Z specification from OWL context ontology is presented in Chapter 4. The prepared

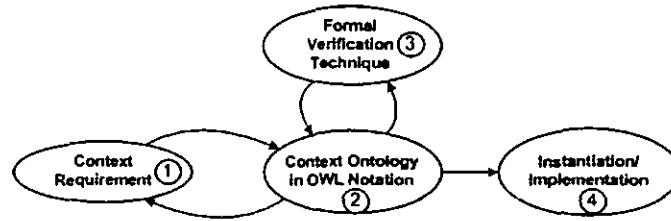


Figure 5.1: Process of developing context ontology in this thesis

OWL semantic web format is mapped to formal specification to enable reasoning process using formal verification technique, e.g. in Z/EVES environment. Refinement of context ontology will take place once the formal verification process discovered inconsistency concept. Afterward, the refined semantic web model of context ontology can then be prepared for instantiation process, or to be used directly by application run-time (process ④ in Figure 5.1). To conclude, this thesis proposes formal verification technique as a complementary step to develop context ontology.

5.2 Context Modeling Using OWL

Formalizing context ontology in OWL not only contains the vocabularies of concepts, but involving relationships among them as well. OWL semantic web allows us to achieve this goal in two steps. First, it allows us to define concepts and their inter-relationships, e.g. describing person, location, devices. etc in our context ontology. Second, it allows us to define instance data pertaining to some specific class.

The strengths of visual ontology modeling as used in Chapter 3 are definitely helpful on the modeling context ontology. To date, Swoop version 2.3, as well as Protege version 4, is connected to Pellet OWL DL semantic web reasoner. The feature to visualize context ontology in OWL semantic web language could assist the context-aware designer to define context ontology along with checking process, hence the inconsistency can be detected at the early modeling process.

By benchmarking both ontology editor mentioned above, Swoop has more strength point in modeling and evaluating the ontology.

1. Swoop has the interface to show the axiom causing the inference result after reasoning process, such for subsumption and instantiation checking. The example of this feature is depicted in the Figure 5.2.
2. Upon detecting the inconsistency, Swoop can show the source of inconsistency and come up a proposed options to fix the inconsistency (see Figure 5.3). This

OWL-Class: \textcircled{C} Free

Intersection of: (Add)
 \textcircled{F} CurrentActivity . (¬ \textcircled{C} Busy) (Delete)
 \textcircled{C} Person (Delete)

Subclass of: (Add)
 \textcircled{C} Deduced (Delete)
 \textcircled{C} Person (Why?)

Explanation

Axioms causing the inference

Free \sqsubseteq Person:

1) \textcircled{C} Free \equiv ((\textcircled{F} CurrentActivity . (¬ \textcircled{C} Busy)) \cap \textcircled{C} Person)

Figure 5.2: Explanation of axioms causing the inference in Swoop

is a very promising feature for rapid context ontology development using OWL semantic web model.

The screenshot shows the Swoop interface with a table of 'Erroneous Axioms'. The table has columns for 'Axiom', 'Arity', 'Impact', 'Usage', 'Rank', and 'Status'. Below the table are buttons for 'Repair All Roots', 'Remove All Erroneous', and 'Repair Selected (above)'. A 'PREVIEW' section at the bottom shows 'Unsettleable Fixed: 0 Remaining: 14' and 'Entailments Lost: 0 Retained: 5'.

Erroneous Axioms	Arity	Impact	Usage	Rank	Status
1) \textcircled{C} Planned \equiv (\textcircled{F} PersonActivity)	2	1	2	1.9	(E) (L)
2) \perp (\textcircled{F} CurrentActivity domain \textcircled{C} Person)	2	1	22	3.7	(E) (L)
3) \textcircled{C} Deduced \equiv (\textcircled{F} CurrentActivity)	2	1	14	2.4	(E) (L)
4) \textcircled{C} Free \equiv (\textcircled{F} CurrentActivity . (¬ \textcircled{C} Busy))	2	2	2	0.49	(E) (L)
5) \perp (\textcircled{C} Busy \equiv ((\textcircled{F} Person . \textcircled{C} OfficeLocation) \cup (\textcircled{F} PersonActivity . \textcircled{C} Stalled)) \cap \textcircled{C} Person)	2	0	30	1.2	(E) (L)
6) \perp (\textcircled{C} Busy \sqsubseteq ¬ \textcircled{C} Free)	2	0	1	-1.7	(Unsettleable) (L)
7) \perp (\textcircled{C} Free \sqsubseteq \textcircled{C} Deduced)	2	2	2	0.29	(E) (L)
8) \textcircled{C} LabWork \sqsubseteq \textcircled{C} Planned	2	0	6	-1.2	(E) (L)
9) \textcircled{C} Lecturing \sqsubseteq \textcircled{C} Planned	2	0	2	-1.2	(E) (L)

Figure 5.3: Depicting the source of error and option to fix

To conclude, OWL semantic web language provide a standard representation to structure contextual information. OWL can associate semantics to represent concepts like class hierarchy, sets, restriction on class, etc. Using this semantics, the inference engine application can act upon OWL document to derive fact, to answer the query about semantic entity, and to deduce the context upon the reasoning process.

The OWL Web semantic web language is designed to be used directly by application entity that needs to process the information instead of just presenting information to human. For this purpose, OWL facilitates machine *interpretability* of document content than that supported by XML, RDF, and RDF Schema (RDF-S)[63].

5.3 Ontology Expressiveness

Practically, the existing OWL semantic web editor and OWL reasoner could be used to get the statistic and expressiveness of ontology being modeled. To do so, the modeling approach provided in this thesis, Swoop and Protege editor are connected to Pellet OWL DL reasoner. Those tool could reflect the statistic and expressiveness of our semantic web model, as depicted in Figure 5.4.

Metrics	
Class count	60
Object property count	9
Data property count	26
Individual count	63
DL expressivity	SHOIN(D)

Class axioms	
SubClass axioms count	62
Equivalent classes axioms count	25
Disjoint classes axioms count	1
GCI count	0
Hidden GCI Count	20

Object property axioms	
Sub object property axioms count	11
Equivalent object properties axioms count	0
Inverse object properties axioms count	1
Disjoint object properties axioms count	0
Functional object property axioms count	0
Inverse functional object property axioms count	0
Transitive object property axioms count	11
Symmetric object property axioms count	0
Anti-symmetric object property axioms count	0
Reflexive object property axioms count	0
Irrreflexive object property axioms count	0

Figure 5.4: Semantic Web Statistic of CIS Context Model, rendered by Pellet OWL DL Reasoner through Protege Editor

As can be seen in Figure 5.4, OWL semantic web model of CIS context ontology conforms to $SHOIN(D)$ family. The language family or expressiveness of DLs are determined by language constructors and axioms we use, as described in Chapter 3. The summary of axioms that form expressiveness in CIS context model are in the following table.

5.4 Reflection on the Proposed Method

Compared to the semantic web reasoning tool, the apparent disadvantage of Z/EVES is that it has a lower degree of automation and can only perform reasoning tasks interactively.

Prior to verify the Z specification of context ontology, some assumption rule labels have to be defined, including the theorem, and calling all relevant assumption rule and

Table 5.1: Context Ontology Statistic

Name	DLs Syntax	Axiom Example	Language
Top	\top	Δ^I	\mathcal{AL}
Bottom	\perp	\emptyset	\mathcal{AL}
Atomic Concept	A	<i>Location</i>	\mathcal{AL}
Atomic Role	R	<i>currentActivity</i>	\mathcal{AL}
Disjoint	$\neg C$	<i>Hardware</i> $\equiv \neg$ <i>Software</i>	\mathcal{C}
Intersection	$C \sqcap D$	\exists <i>run.Browser</i> \exists <i>connectedTo.Internet</i>	\sqcap \mathcal{AL}
Value Restriction	$\forall R.C$	\forall <i>currentActivity.Planned</i>	\mathcal{AL}
Existential Quant.	$\exists R.C$	\exists <i>locatedIn.Location</i>	\mathcal{AL}
number restriction	$\geq nR$	$\geq 2.run$	\mathcal{N}
Role-value	$R \sqsubseteq S$	<i>run</i> \sqsubseteq <i>use</i>	\mathcal{H}
Nominal	I	\forall <i>loginTo</i> .{ <i>Netware</i> }	\mathcal{O}
Inverse Role	$(^-)R$	<i>use</i> \equiv $(^-)$ <i>ownedBy</i>	\mathcal{I}
Transitive Role	$(^+)R$	$(^+)$ <i>connectedTo</i>	$\mathcal{ALC}+$ Transitive Role = \mathcal{S}

theorem for the proofing process. It is because of Z/EVES is general theorem prover, not only intended to check the conceptual specification like ontology, but can also be used to check another logical theorem. With regard to semantic web checking, the overall checking process are automatically performed by OWL reasoner, hence the designers no need to prepare assumption rule like in Z/EVES tool.

As can be seen from the last section in Chapter 4, the proof process using Z/EVES approach is very interactive and it requires substantial user expertise in interacting with the theorem prover. Although Semantic Web reasoners such as FaCT++ and Pellet can only carry out with a limited number of reasoning tasks (concept consistency, subsumption and instantiation reasoning), due to the expressivity limitation of the ontology languages, they are fully automated reasoners. It is advantageous to use semantic web reasoners to perform reasoning tasks that can be automated.

However, the high degree of expressiveness of Z language implies that it can capture properties beyond the OWL ontology languages and applying Z/EVES to check ontologies will give us more confidence on the correctness of ontology. Moreover, since ontology languages are based on description logics, certain complex properties cannot be represented in the semantic web language. It is required to express and verify the desirable properties, which may be critical to assure the correctness of the ontology.

Comparing the language or notation used to develop context ontology, Z speci-

fication is not intended for application-run time (not executable format). Instead of that, context ontology in Z is designed to be expressive and human understandable for formal specification purpose. Due to its feature, Z formal specification is suitable for complementary approach to specify and check ontology beyond the OWL semantic web modeling. On the contrary, OWL notation is intended to be executable format, because it is written on top of XML notation. Hence, during the implementation phase, the context-aware developer can directly execute ontology in OWL format by using the available OWL APIs.

5.5 Chapter Summary

In the previous context ontology modeling approach, during the ontology development, the semantic web reasoner is used to check the correctness of context ontology being modeled. In Chapter 4, the Z notation of context ontology model has been specified, which is generated by mapping from the OWL semantic web context ontology version.

Some limitations have also been identified, where the complementary checking still needs more user interaction in term of defining rule, theorem and command to perform semantic checking in Z/EVES environment. Comparing to semantic web reasoner, all semantic checking process are performed automatically once the ontology has been written completely. In the next chapter, the conclusion and future research direction will be presented, which formally conclude the research work presented in this thesis.

Chapter 6

Conclusion and Future Works

This final chapter presents a conclusion of the whole thesis, including the summary of contributions, followed by recommendations on future work, including limitation of our research work.

This thesis concludes that the method of context modeling approach for distributed and ubiquitous computing environments with respect to the requirements listed in Chapter 3 can be accommodated by ontology model. However, this does not mean that the other approaches are unsuitable for ubiquitous computing environments.

To develop context ontology model, OWL Semantic Web Language has been defined that was derived from DLs conceptual model. Semantic web is chosen since it is currently promising context model for the implementation or application run-time purpose.

The syntax and semantic of OWL-Z is used to map semantic web version of CIS context ontology onto Z formal specification. Z notation was chosen as a formal specification language, since the semantics of OWL language could be expressed in Z specification language.

Current version of Swoop editor is combined with Pellet OWL DL reasoner to carry out semantic checking of OWL context ontology. It was demonstrated that Swoop OWL editor is a very helpful to since it provides features to quickly model a very complex ontology. Swoop is connected to Pellet reasoner, therefore, the correctness of OWL context ontology can be carried out on the fly. During the modeling process, context ontology needs to be refined to achieve the consistent ontology model.

In this thesis, Z/EVES theorem prover is used to carry out semantic checking of context ontology model in Z notation. It was demonstrated that validation of Z specification of context ontology surprisingly could be performed beyond the semantic web reasoner. It was also demonstrated that Z/EVES theorem prover was able to detect

the inconsistency error that was present in the previous OWL version of context ontology.

6.1 Thesis contribution

The contributions of this thesis can be summarized as follows:

1. Thesis address context ontology development approach by employing formal specification as a complementary technique to specify and verify context ontology. By defining this context ontology development process, the refinement of context ontology is performed by utilizing formal specification technique. Thus, any inconsistency error that was undetected by semantic web reasoner is hopefully to be discovered by means of this formal specification technique.
2. The use of Z formal notation is proposed in this thesis as the complementary technique to specify context ontology (see Chapter 4). By mapping semantic web ontology onto Z notation, this has enabled formal methods tool (theorem prover tool such as Z/EVES) to perform semantic checking and reasoning beyond semantic web reasoner. The use of formal specification language also affects to the separation of modeling language. Modeling language used by context developer for application run-time (implementation purpose) is distinguished from language used by context designer for specification/design concern. Well defined context ontology in semantic web language (after refined) is then prepared for the context developer to further develop context-aware application. Meanwhile, the Z specification of context ontology model is prepared for the refinement process of ontology model using formal specification technique.
3. It was demonstrated in this thesis that the validity of context ontology model can be checked by means of Z/EVES tools. It was shown in Chapter 3 that the inconsistency of context ontology cannot be detected by current Pellet OWL DL reasoner. Having mapped onto Z notation and performed semantic checking in Z/EVES tool, this tool could discover inconsistency in context ontology, such as explained in Chapter 4. Z/EVES could also display the source of inconsistency in context ontology definition.

6.2 Future Work Directions

Based on the works in this thesis, there are a number of directions of future research that may be beneficial to the Context-Aware Community and Semantic Web Communities.

1. In this thesis, context ontology is constructed which conforms to $SHOIN(\mathcal{D})$ family. For further research, it is feasible to construct more expressive context ontology. Consequently, the OWL-Z syntax and semantics have to be redefined to accommodate the expressiveness of ontology language (beyond $SHOIN(\mathcal{D})$, or using OWL 2 language construct $SHROIQ(\mathcal{D})$)
2. Another interesting follow-up is how to model ontology that will involve in dynamic context-aware interaction system. The interaction system, including its ontology, should be prepared in formal specification manner. Further, the mapping onto implementation language can then be provided as well.
3. This thesis excluded an automatic tool to map context ontology in OWL semantic web onto Z specification. This transformation tool is another research interests that can be addressed in the future work, such as by utilizing XSLT technology.
4. This thesis excluded the implementation or development of context-aware system. For further implementation, many of context-aware application frameworks are available for free and our context ontology model can be attached after doing some modifications / adjustments.

References

- [1] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, September 2004.
- [2] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, "A data-oriented survey of context models," *Journal of ACM SIGMOD*, vol. 12, no. 36, pp. 19–26, December 2007.
- [3] H. Chen, T. Finin, and A. Joshi, "An ontology for context-aware pervasive computing environments," *Journal of The Knowledge Engineering Review*, vol. 18, no. 13, pp. 197–207, September 2003.
- [4] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung, "Ontology based context modeling and reasoning using owl," in *Proceeding of the second IEEE Annual Conference on Pervasive Computing and Communication Workshops (PERCOMW 04)*, 2004.
- [5] T. Gu, X. H. Wang, H. K. Pung, and D. Q. Zhang, "An ontology-based context model in intelligent environments," Department of Computer Science, National University of Singapore, Singapore, Tech. Rep., 2005.
- [6] T. Gu, H. K. Pung, and D. Zhang, "A service-oriented middleware for building context-aware services," *Elsevier Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, 2005.
- [7] D. R. de Almeida, C. de Souza Baptista, and F. G. de Andrade, "Using ontologies in context-aware application," in *Proceeding of the 17th International Conference on Database and Expert System Applications (IEEE DEXA 06)*, 2006.
- [8] K. Jensen, *Coloured Petri Net. basic Concept, Analysis Methods and Practical Use*. Springer, 1997.

-
- [9] R. Sharp, *Principle of Protocol Design*. Springer-Verlag, 2008.
- [10] D. Bjørner and M. C. Henson, "An overview of specification language," in *Logics of Specification Languages*. Springer, 2008, pp. 3–12.
- [11] N. Nissanke, *Formal Specification: Technique and Applications*. Springer-Verlag, London, 1999.
- [12] J. Bowen, *Formal Specification and Documentation using Z: A Case Study Approach*. Open University, UK, 2003.
- [13] J. S. Dong, C. H. Lee, H. B. Lee, Y. F. Li, and H. Wang, "A combined approach to checking web ontologies," in *Proceeding of 13th ACM International World Wide Web Conference (WWW'04)*. New York, USA: ACM Press, May 2004, pp. 714–722.
- [14] J. S. Dong, Y. Feng, and Y. F. Li, "Verifying owl and orl ontologies in pvs," in *Proceeding of 1st International Colloquium on Theoretical Aspects of Computing (ICTAC'04)*, vol. 3407. Guiyang, China: Springer-Verlag, 2005, pp. 265–279.
- [15] J. S. Dong, J. Sun, and H. Wang, "Z approach to semantic web," in *ICFEM 2002*, C. George and H. Miao, Eds., vol. LNCS 2495. Berlin: Springer-Verlag, 2002, pp. 156–167.
- [16] J. S. Dong, C. H. Lee, Y. F. Li, and H. Wang, "Verifying daml+oil and beyond in z/leves," in *Proceeding of 26th International Conference on Software Engineering (ICSE'04)*. Edinburgh, Scotland, UK: ACM/IEEE Press, 2004, pp. 201–210.
- [17] H. Wang, J. S. Dong, J. Sun, and J. Sun, "Reasoning support for semantic web ontology family languages using alloy," *International Journal of Multiagent and Grid Systems, Special issue on Agent-Oriented Software Development Methodologies*, vol. 2, no. 4, pp. 455–471, December 2006.
- [18] B. N. Schilit, N. I. Adams, and R. Want., "Context-aware computing applications," in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, December 1994, pp. 85–90.
- [19] A. K. Dey, "Understanding and using context," *Journal of Personal Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, February 2001.

- [20] S. Loke, *Context-Aware Pervasive System: A New Breed of Applications*. Auerbach Publication, 2006.
- [21] H. V. Kranenburg, A. H. Salden, H. Eertink, R. van Eijk, and J. de Heer, "Ubiquitous attentiveness - enabling context-aware mobile applications and services," in *EUSAI*, 2003, pp. 76–87.
- [22] Q. Weijun, S. Yuanchun, and S. Yue, "Ontology-based context-aware middleware for smart spaces," *Journal of Tsinghua Science and Technology*, vol. 12, no. 6, pp. 703–711, December 2007.
- [23] C. B. Anagnostopoulos, A. Tsounis, and S. Hadjiefthymiades¹, "Context-awareness in mobile computing environments," *Journal of Wireless Personal Communications*, vol. Volume 42, no. Number 3, pp. 445–464, August 2007.
- [24] E. Kim and J. Choi, "An ontology-based context model in a smart home," in *Proceedings of International Conference in Computational Science and Its Applications - ICCSA 2006*, vol. 3983. Springer, May 8-11 2006, pp. 11–20.
- [25] R. Krummenacher and T. Strang, "Ontology-based context modeling," in *Proceedings of Context Awareness for Proactive Systems (CAPS 2007)*, Guildford, United Kingdom, 2007.
- [26] W. Hai, "Semantic web and formal design methods," Ph.D. dissertation, Department of Computer Science, National University of Singapore, 2004.
- [27] Y. F. Li, "A formal modeling approach to ontology engineering," Ph.D. dissertation, Department of Computer Science, National University of Singapore, 2006.
- [28] D. Lucanu, Y. F. Li, and J. S. Dong, "Soundness proof of z semantics of owl using institutions," in *Proceeding of 14th International World Wide Web Conference (WWW'05)*. Chiba, Japan: ACM Press, 2005.
- [29] C. Lucanu, Y. F. Li, and J. S. Dong, "Semantic web languages toward an institutional perspective," *Journal of Algebra, Meaning, and Computation*, vol. LNCS 4060, pp. 99–123, 2006.
- [30] K. Henriksen, J. Indulska, and T. McFadden, "Modelling context information with orm," *On the Move to Meaningful Internet Systems Journal*, pp. 626–635, 2005.

- [31] K. Henricksena and J. Indulska, "Developing context-aware pervasive computing applications: Models and approach," *Journal of Pervasive and Mobile Computing*, vol. 2, pp. 37–64, 2006.
- [32] A. Borgida and R. J. Brachman, "Conceptual modeling with description logics," in *The Description Logics Handbook: Theory, Implementation and Application*. Cambridge University Press, 2004, ch. 10.
- [33] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "Web ontology language (owl) abstract syntax and semantics section 3. direct model-theoretic semantics," W3C, W3C, W3C Recommendation, 2004, <http://www.w3c.org/TR/owl-semantics/>.
- [34] B. Remmache, "Specification and analysis of context-aware system," Ph.D. dissertation, Dependable Systems and Software Engineering, School of Electronics and Computer Science, University of Southampton, October 2007.
- [35] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceeding 1999 1st International Symposium on Handheld and Ubiquitous Computing*, 1999, pp. 304–307.
- [36] S. Domnitcheva, "Location modeling: State of the art and challenges," in *Workshop on Location Modeling for Ubiquitous Computing Ubicomp 2001*, 2001.
- [37] T. Strang, C. Linnhoff-Popien, and K. Frank, "Integration issues of an ontology based context approach," in *Proceedings of the IADIS International Conference WWW/Internet 2003, ICWI 2003*, Portugal, 2003, pp. 361–368.
- [38] D. Zhang, T. Gu, and X. Wang, "Enabling context-aware smart home with semantic technology," *International Journal of Human-friendly Welfare Robotic Systems*, vol. 6, no. 4, pp. 233–248, 2005.
- [39] T. Gu, H. K. Pung, and D. Zhang, "Towards an osgi-based infrastructure for context-aware applications in smart homes," *Journal of IEEE Pervasive Computing*, vol. 3, no. 4, 2004.
- [40] J. C. Augusto and C. D. Nugent, *Smart Homes Can Be Smarter*. School of Computing and Mathematics, University of Ulster at Jordanstown, UK: Springer Verlag, 2006, ch. 1, pp. 1–15.
- [41] C. L. Gal, *Smart Office*. Wiley, 2005, ch. Chapter 14, p. 323.

- [42] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Journal of Artificial Intelligent*, vol. 48, no. 1, pp. 1–26, 1991.
- [43] F. Baader, R. Kusters, and F. Wolter, "Extension to description logics," in *The Description Logics Handbook: Theory, Implementation and Application*. Cambridge University Press, 2004, ch. 6.
- [44] J. de Bruijn, A. Polleres, R. Lara, and D. Fensel, "Owl-dl vs. owl flight: Conceptual modeling and reasoning for the semantic web," in *Proceedings of the 14th International conference on World Wide Web*, November 2005, pp. 623 – 632. [Online]. Available: <http://www.deri.ie>
- [45] S. Decker, F. van Harmelen, J. Broekstra, M. Erdmann, D. Fensel, I. Horrocks, M. Klein, and S. Melnik, "The semantic web: The roles of XML and RDF," *IEEE Internet Computing Journal*, vol. 4, no. 5, pp. 63–74, 2000. [Online]. Available: <http://www.computer.org/internet/>
- [46] A. Kalyanpur, B. Parsia, E. Sirin, and B. C. Grau, "Swoop: A web ontology editing browser," *Journal of Web Semantics*, vol. 2, p. 144–153, 2006.
- [47] K. K. Breitman, K. K. Breitman, and W. Truszkowski, *Semantic Web: Concepts, Technologies and Applications*, M. Hinchey, Ed. Springer Verlag, 2007.
- [48] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf, "Reasoning in description logics," *Journal of Principles of Knowledge Representation and Reasoning*, pp. 193–238, 1996.
- [49] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From SHIQ and RDF to OWL: The making of a web ontology language," *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, 2003. [Online]. Available: <download/2003/HoPH03a.pdf>
- [50] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. Elsevier B.V, 2004.
- [51] Ó. Corcho, A. Gómez-Pérez, R. González-Cabero, and M. del Carmen Suárez-Figueroa, "Odeval: A tool for evaluating rdf(s), daml+oil and owl concept taxonomies," in *Proceeding of Artificial Intelligence Applications and Innovations, IFIP 18th World Computer Congress(AIAI-2004)*, 2004, pp. 369–382.
- [52] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl dl reasoner," *Journal of Web Semantics*, vol. 5, no. 2, p. 51–73, 2007.

- [53] D. Tsarkov and I. Horrocks, "Fact++ description logic reasoner: System description," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4130 LNAI, pp. 292–297, 2006.
- [54] "Renamed abox and concept expression reasoner," August 2008. [Online]. Available: <http://www.sts.tu-harburg.de/~r.f.moeller/racer>
- [55] "Pellet owl -dl reasoner," August 2008. [Online]. Available: <http://www.pellet.org>
- [56] "Protege owl editor," August 2008. [Online]. Available: <http://protege.stanford.edu/>
- [57] "Swoop owl dl editor," August 2008. [Online]. Available: <http://www.mindswap.org/2004/SWOOP/>
- [58] J. Jacky, *The Way of Z: Pracical Programming with Formal Methods*. Cambridge University Press, 1997.
- [59] J. M. Spivey, *The Z Notation: A Reference Manual*, 2nd ed., ser. International Series in Computer Science. Prentice-Hall, 1992.
- [60] I. Meisels and M. Saaltink, "The z/eves reference manual," ORA Canada, One Nicholas Street, Suite 1208 - Ottawa, Ontario K1N 7B7 - CANADA, Technical Report TR-97-5493-03d, September 1997.
- [61] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language reference," W3C Recommendation, 10 February 2004, <http://www.w3.org/TR/owl-ref>.
- [62] F. Baader and U. Sattler, "An overview of tableau algorithms for description logics," *Journal of Studia Logica*, pp. 5–40, 2001.
- [63] D. L. McGuinness and F. van Harmelen, "Owl web ontology language overview," W3C, Tech. Rep., 2004. [Online]. Available: <http://www.w3c.org/TR/owl-features/>

Appendix A

DLs Specification of CIS Context Model

Domain interpretation = Δ^I .

Following is high level concept of CIS Context Model

$$(Location, Person, Activity, Device, Network) \sqsubseteq \Delta^I$$

A.1 Person Conceptual Model

Following is definition of *Person* concept and its related roles restriction.

$$(Lecturer, Staff, Postgrad, Student) \sqsubseteq Person$$

$$Person \sqcap \forall use.Device$$

$$Device \sqcap \forall ownedBy.Person$$

$$Person \sqcap \forall locatedIn.Location$$

$$Person \sqcap \forall currentActivity.Activity$$

$$Person \sqcap \forall logInto.Server$$

$$Server \equiv \forall loginTo(\{NETWARE\})$$

$$\geq 2.run$$

$$\equiv 1.currentActivity$$

$$Person \sqcap \forall connectedTo.Device$$

$$Person \sqcap \forall connectedTo.Network$$

A.2 Location Conceptual Model

Following is definition of *Location* concept and its individuals.

$(Indoor, Outdoor) \sqsubseteq Location$

$(Longitude, Latitude) \sqsubseteq Outdoor$

$(Room, Building) \sqsubseteq Indoor$

$(Room) \sqsubseteq Building$

$ClassRoom, SeminarRoom, LectureHall, MeetingRoom,$

$OfficeRoom, Lab) \sqsubseteq Room$

$ClassRoom \equiv (\{C01\}, \{C02\}, \{C03\}, \{C04\}, \{C05\}, \{C06\}, \{D01\}, \{D02\},$
 $\{D03\}, \{D04\}, \{D05\}, \{D06\})$

$LectureHall \equiv (\{LH01\}, \{LH02\}, \{LH03\}, \{LH04\}, \{LH04\}, \{LH06\})$

$MeetingRoom \equiv (\{010310\}, \{010210\}, \{0203010\})$

$OfficeRoom \equiv (\{LECTUREROOM\}, \{POSTGRADROOM\})$

$Laboratory \equiv (\{DATACOM\}, \{MULTIMEDIA\}, \{PROGRAMMING\}, \{VR\})$

$SeminarRom \equiv (\{010202\}, \{010310\})$

$Indoor \sqcap \forall \text{equiped With. Desktop}$

A.3 Device and Network Conceptual Model

Following is definition of *Device* concept.

$(MobileDevice, Desktop, NetworkDevice) \sqsubseteq Device$
 $(Laptop, PDA, MobilePhone) \sqsubseteq MobileDevice$
 $(Software, Hardware) \sqsubseteq Laptop$
 $Software \sqsubseteq \neg Hardware$
 $(Software, Hardware) \sqsubseteq PDA$
 $(Software, Hardware) \sqsubseteq Desktop$
 $(Software, Hardware) \sqsubseteq MobilePhone$
 $(ApplicationRun, ProcessRun) \sqsubseteq Software$
 $(EmailApp, OfficeApp, InternetApp) \sqsubseteq ApplicationRun$
 $(Browser, EmailClient, IMApplication) \sqsubseteq InternetApplication$
 $Browser \equiv (\{FIREFOX\}, \{MOZILLA\}, \{SAFARI\}, \{IE\}, \{OPERA\})$
 $IMApplication \equiv (\{MSNChat\}, \{YM\}, \{GTALK\}, \{GAIM\})$
 $EmailClient \equiv (\{THUNDERBIRD\}, \{OUTLOOK\}, \{WEBMAIL\})$
 $OfficeApp \equiv (\{WORDPROCESSOR\}, \{PDFREADER\}, \{SPREADSHEET\})$
 $ProcessRun \equiv (\{ANTIVIRUS\}, \{SERVICE\}, \{TRAY\})$
 $(AccessPoint, Server, Router) \sqsubseteq NetworkDevice$
 $Device \sqcap \forall connectedTo. Network$

$(WifiNetwork, Server, GSM, 3G) \sqsubseteq Network$
 $GSM \sqcup 3G \equiv (\{DIGI\}, \{MAXIS\}, \{CELCOM\})$
 $WifiNetwork \sqcap \exists SSIDName.XSD$
 $Gateway \equiv (\{160.0.226.202\})$
 $Proxy \equiv (\{160.0.226.206\}, \{160.0.226.207\}, \{160.0.226.208\},$
 $\{160.0.226.19\})$

A.4 Activity Conceptual Model

Following is a definition of *Activity* concept and its sub classes.

$$\begin{aligned} (PlannedActivity, DeducedActivity) &\sqsubseteq Activity \\ (Meeting, Lecturing, Seminar, LabActivity, Tutotial) &\sqsubseteq PlannedActivity \\ (Busy, Free, Chating, Browsing, Not_At_Office, Available, On_the_Phone, \\ Opening_Email) &\sqsubseteq DeducedActivity \end{aligned}$$

A.5 Axioms of Restriction

Following is definition of axioms of class and property restrictions.

Class *Browsing* Restriction

$$\begin{aligned} Browsing &\equiv Person \sqcap \forall connectedTo. Internet \sqcap \forall run. Browser \sqcap \\ &\forall connectedTo. Internet \\ Browsing &\equiv \forall run. (\{IE\}, \{FIREFOX\}, \\ &\{MOZILLA\}, \{SAFARI\}, \{OPERA\}) \sqcap Person \sqcap \forall connectedTo. Internet \end{aligned}$$

Class *Busy* Restriction

$$\begin{aligned} Busy &\equiv Person \sqcap \forall run. OfficeApplication \sqcap \forall located. OfficeRoom \\ Busy &\equiv Person \sqcap \exists currentActivity. Planned \\ Free &\sqsubseteq \neg Busy \end{aligned}$$

Class *Chatting* Restriction

$$\begin{aligned} Chatting &\equiv Person \sqcap (\forall conectedTo. Internet) \sqcap (\exists run. IMApplication) \\ Chatting &\equiv Person \sqcap (\forall conectedTo. Internet) \sqcap (\exists .run(\{YM\} \sqcup \{GAIM\} \\ &\sqcup \{MSN\} \sqcup \{GTALK\})) \end{aligned}$$

Class *Not_At_Office* Restriction

$$\begin{aligned} Not_At_Office &\equiv Person \sqcap \forall locatedIn. \neg OfficeRoom \\ Not_At_Office &\equiv Person \sqcap \forall locatedIn. \neg (\{PostgradRoom\}, \{LectureRoom\}) \end{aligned}$$

Class *On_the_Phone* Restriction

$$\text{On_the_Phone} \equiv \text{Person} \sqcap \forall \text{use.MobilePhone} \sqcap \exists \text{connectedTo}(\text{GSM} \sqcup \text{3G})$$

Class *Opening_Email*

$$\text{Opening_Email} \equiv \text{Person} \sqcap (\forall \text{connectedTo.Internet}) \sqcap (\exists \text{run.EmailApplication})$$

$$\text{Opening_Email} \equiv \text{Person} \sqcap (\forall \text{connectedTo.Internet}) \sqcap (\exists \text{run}(\{\text{OUTLOOK}\} \sqcup \{\text{THUNDERBIRD}\} \sqcup \{\text{WEBMAIL}\}))$$

A.6 Class and Role Data Type

We assume that XSD is a class of data containing data type, because DLs notation has no definition of data type role (for implementation modeling like OWL, roles are distinguished for object and data type). Following is a definition of role restricted with XSD, which is used to describe a data type definition.

$$(\text{Lecturer} \sqcup \text{Staff} \sqcup \text{PostGrad} \sqcup \text{Student}) \equiv \text{Person} \sqcap \exists \text{fullName.XSD}$$

$$(\text{Lecturer} \sqcup \text{Staff} \sqcup \text{PostGrad}) \equiv \text{Person} \sqcap \exists \text{officeAddress.XSD}$$

$$(\text{Lecturer} \sqcup \text{Staff} \sqcup \text{PostGrad} \sqcup \text{Student}) \equiv \text{Person} \sqcap \exists \text{emailAddress.XSD}$$

$$(\text{Lecturer} \sqcup \text{Staff} \sqcup \text{PostGrad} \sqcup \text{Student}) \equiv \text{Person} \sqcap \exists \text{imAddress.XSD}$$

$$(\text{Lecturer} \sqcup \text{Staff} \sqcup \text{PostGrad} \sqcup \text{Student}) \equiv \text{Person} \sqcap \exists \text{phoneNumber.XSD}$$

$$(\text{GSM} \sqcup \text{3G}) \sqcap \forall \text{cellID.XSD}$$

$$\text{Indoor} \sqcap \forall \text{buildingName.XSD}$$

$$\text{Indoor} \sqcap \forall \text{roomNumber.XSD}$$

$$\text{Planned} \sqcap \forall \text{startTime.XSD}$$

$$\text{Planned} \sqcap \forall \text{endTime.XSD}$$

$$\text{Gateway} \sqcap \forall \text{gatewayIP.XSD}$$

$$\text{Proxy} \sqcap \forall \text{proxyIP.XSD}$$

$$\text{Outdoor} \sqcap \forall \text{latitude.XSD}$$

$$\text{Outdoor} \sqcap \forall \text{longitude.XSD}$$

$$\text{Hardware} \sqcap \forall \text{memorySize.XSD}$$

$$\text{Software} \sqcap \forall \text{operatingSystem.XSD}$$

$$\text{Hardware} \sqcap \forall \text{processorType.XSD}$$

$$(\text{GSM} \sqcup \text{3G}) \sqcap \forall \text{signalStrength.XSD}$$

$$\text{Device} \sqcap \forall \text{ipAddress.XSD}$$

Appendix B

Context-Aware Ontology Specification

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY cis "http://context.org/cis#" >
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY owl11 "http://www.w3.org/2006/12/owl11#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY owl11xml "http://www.w3.org/2006/12/owl11-xml#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<rdf:RDF xmlns="http://context.org/cis#"
  xml:base="http://context.org/cis"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl11="http://www.w3.org/2006/12/owl11#"
  xmlns:owl11xml="http://www.w3.org/2006/12/owl11-xml#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cis="http://context.org/cis#">
  <owl:Ontology rdf:about=""/>

<!--
  ////////////////////////////////////////////////////////////////////
  // Object Properties
  //
  ////////////////////////////////////////////////////////////////////
  -->
  <!-- http://context.org/cis#associatedWith -->

  <owl:ObjectProperty rdf:about="&cis;associatedWith">
    <rdfs:range rdf:resource="&cis;AccessPoint"/>
    <rdfs:DELTA rdf:resource="&cis;WifiNetwork"/>
  </owl:ObjectProperty>

  <!-- http://context.org/cis#connectedTo -->

  <owl:ObjectProperty rdf:about="&cis;connectedTo">
```



```
<rdf:type rdf:resource="#owl:TransitiveProperty"/>
<rdfs:DELTA rdf:resource="#cis;Device"/>
<rdfs:range rdf:resource="#cis;Device"/>
<rdfs:range rdf:resource="#cis;Network"/>
<rdfs:DELTA rdf:resource="#cis;Person"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#currentActivity -->
<owl:ObjectProperty rdf:about="#cis;currentActivity">
  <rdfs:range rdf:resource="#cis;Activity"/>
  <rdfs:DELTA rdf:resource="#cis;Person"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#equippedWith -->
<owl:ObjectProperty rdf:about="#cis;equippedWith">
  <rdfs:DELTA rdf:resource="#cis;ClassRoom"/>
  <rdfs:range rdf:resource="#cis;Desktop"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#locatedIn -->
<owl:ObjectProperty rdf:about="#cis;locatedIn">
  <rdfs:range rdf:resource="#cis;Location"/>
  <rdfs:DELTA rdf:resource="#cis;Person"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#logInto -->
<owl:ObjectProperty rdf:about="#cis;logInto">
  <rdfs:DELTA rdf:resource="#cis;Person"/>
  <rdfs:range rdf:resource="#cis;Server"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#ownedBy -->
<owl:ObjectProperty rdf:about="#cis;ownedBy">
  <owl:inverseOf rdf:resource="#cis;use"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#run -->
<owl:ObjectProperty rdf:about="#cis;run">
  <rdfs:range rdf:resource="#cis;Software"/>
  <rdfs:subPropertyOf rdf:resource="#cis;use"/>
</owl:ObjectProperty>
<!-- http://context.org/cis#use -->
<owl:ObjectProperty rdf:about="#cis;use">
  <rdfs:range rdf:resource="#cis;Device"/>
  <rdfs:DELTA rdf:resource="#cis;Person"/>
</owl:ObjectProperty>

<!--
//
// Data properties
//
//
-->
<!-- http://context.org/cis#buildingName -->
<owl:DatatypeProperty rdf:about="#cis;buildingName">
  <rdfs:DELTA rdf:resource="#cis;Indoor"/>
  <rdfs:range rdf:resource="#xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#emailAddress -->
<owl:DatatypeProperty rdf:about="#cis;emailAddress">
  <rdfs:DELTA rdf:resource="#cis;Person"/>
```

```
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#endTime -->
<owl:DatatypeProperty rdf:about="&cis;endTime">
    <rdfs:DELTA rdf:resource="&cis;Activity"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#fullName -->
<owl:DatatypeProperty rdf:about="&cis;fullName">
    <rdfs:DELTA rdf:resource="&cis;Person"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#ipAddress -->
<owl:DatatypeProperty rdf:about="&cis;ipAddress">
    <rdfs:DELTA rdf:resource="&cis;Device"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#latitude -->
<owl:DatatypeProperty rdf:about="&cis;latitude">
    <rdfs:DELTA rdf:resource="&cis;Outdoor"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#longitude -->
<owl:DatatypeProperty rdf:about="&cis;longitude">
    <rdfs:DELTA rdf:resource="&cis;Outdoor"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#memorySize -->
<owl:DatatypeProperty rdf:about="&cis;memorySize">
    <rdfs:DELTA rdf:resource="&cis;Hardware"/>
    <rdfs:range rdf:resource="&xsd;integer"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#messengerID -->
<owl:DatatypeProperty rdf:about="&cis;messengerID">
    <rdfs:DELTA rdf:resource="&cis;Profile"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#officeRoom -->
<owl:DatatypeProperty rdf:about="&cis;officeRoom">
    <rdfs:DELTA rdf:resource="&cis;Profile"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#operatingSystem -->
<owl:DatatypeProperty rdf:about="&cis;operatingSystem">
    <rdfs:DELTA rdf:resource="&cis;Software"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#phoneNumber -->
<owl:DatatypeProperty rdf:about="&cis;phoneNumber">
    <rdfs:DELTA rdf:resource="&cis;MobilePhone"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#processorType -->
<owl:DatatypeProperty rdf:about="&cis;processorType">
    <rdfs:DELTA rdf:resource="&cis;Hardware"/>
    <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>
```

```
<!-- http://context.org/cis#proxNumber -->
<owl:DatatypeProperty rdf:about="%&cis;proxNumber">
  <rdfs:DELTA rdf:resource="%&cis;Proxy"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#roomNumber -->
<owl:DatatypeProperty rdf:about="%&cis;roomNumber">
  <rdfs:DELTA rdf:resource="%&cis;OfficeRoom"/>
  <rdfs:range rdf:resource="%&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#signalStrength -->
<owl:DatatypeProperty rdf:about="%&cis;signalStrength">
  <rdfs:DELTA rdf:resource="%&cis;MobileDevice"/>
  <rdfs:range rdf:resource="%&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#ssidName -->
<owl:DatatypeProperty rdf:about="%&cis;ssidName">
  <rdfs:DELTA rdf:resource="%&cis;AccessPoint"/>
  <rdfs:range rdf:resource="%&xsd:string"/>
</owl:DatatypeProperty>
<!-- http://context.org/cis#startTime -->
<owl:DatatypeProperty rdf:about="%&cis;startTime">
  <rdfs:DELTA rdf:resource="%&cis;Activity"/>
  <rdfs:range rdf:resource="%&xsd:string"/>
</owl:DatatypeProperty>
<!--
//
// Classes
//
//
-->
<!-- http://context.org/cis#AccessPoint -->
<owl:Class rdf:about="%&cis;AccessPoint">
  <rdfs:subClassOf rdf:resource="%&cis;NetworkDevice"/>
</owl:Class>
<!-- http://context.org/cis#Activity -->
<owl:Class rdf:about="%&cis;Activity">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="%&cis;currentActivity"/>
      <owl:cardinality rdf:datatype="%&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="%&owl;Thing"/>
</owl:Class>
<!-- http://context.org/cis#ApplicationRun -->
<owl:Class rdf:about="%&cis;ApplicationRun">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="%&cis;run"/>
      <owl:minCardinality rdf:datatype="%&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="%&cis;Software"/>
</owl:Class>
<!-- http://context.org/cis#Browser -->
```

```

<owl:Class rdf:about="&cis;Browser">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&cis;IE"/>
        <rdf:Description rdf:about="&cis;Safari"/>
        <rdf:Description rdf:about="&cis;Opera"/>
        <rdf:Description rdf:about="&cis;Firefox"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&cis;InternetApplication"/>
</owl:Class>
<!-- http://context.org/cis#Browsing -->
<owl:Class rdf:about="&cis;Browsing">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="&cis;run"/>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&cis;Opera"/>
                  </owl:oneOf>
                </owl:Class>
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&cis;Safari"/>
                  </owl:oneOf>
                </owl:Class>
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&cis;Firefox"/>
                  </owl:oneOf>
                </owl:Class>
                <owl:Class>
                  <owl:oneOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="&cis;IE"/>
                  </owl:oneOf>
                </owl:Class>
              </owl:unionOf>
            </owl:Class>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="&cis;connectedTo"/>
          <owl:allValuesFrom rdf:resource="&cis;Internet"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>

```

```

<!-- http://context.org/cis#Busy -->
<owl:Class rdf:about="%cis;Busy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty rdf:resource="%cis;run"/>
          <owl:someValuesFrom rdf:resource="%cis;OfficeApplication"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty rdf:resource="%cis;locatedIn"/>
          <owl:allValuesFrom rdf:resource="%cis;OfficeRoom"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="%cis;currentActivity"/>
      <owl:someValuesFrom rdf:resource="%cis;Planned"/>
    </owl:Restriction>
  </owl:equivalentClass>
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="%cis;run"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="%cis;SpreadSheet"/>
              </owl:oneOf>
            </owl:Class>
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="%cis;PDFReader"/>
              </owl:oneOf>
            </owl:Class>
            <owl:Class>
              <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="%cis;WordProcessor"/>
              </owl:oneOf>
            </owl:Class>
          </owl:unionOf>
        </owl:Class>
      </owl:someValuesFrom>
    </owl:Restriction>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="%cis;Deduced"/>
</owl:Class>

<!-- http://context.org/cis#Chatting -->

<owl:Class rdf:about="%cis;Chatting">
  <owl:equivalentClass>
    <owl:Restriction>

```

```

<owl:onProperty rdf:resource="&cis;run"/>
<owl:someValuesFrom>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="&cis;connectedTo"/>
        <owl:allValuesFrom rdf:resource="&cis;Internet"/>
      </owl:Restriction>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <rdf:Description rdf:about="&cis;GTalk"/>
            </owl:oneOf>
          </owl:Class>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <rdf:Description rdf:about="&cis;YM"/>
            </owl:oneOf>
          </owl:Class>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <rdf:Description rdf:about="&cis;GAIM"/>
            </owl:oneOf>
          </owl:Class>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <rdf:Description rdf:about="&cis;MSNChat"/>
            </owl:oneOf>
          </owl:Class>
        </owl:unionOf>
      </owl:Class>
    </owl:intersectionOf>
  </owl:Class>
</owl:someValuesFrom>
</owl:Restriction>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>

<!-- http://context.org/cis#ClassRoom -->

<owl:Class rdf:about="&cis;ClassRoom">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="&cis;D05"/>
        <rdf:Description rdf:about="&cis;D04"/>
        <rdf:Description rdf:about="&cis;D03"/>
        <rdf:Description rdf:about="&cis;D02"/>
        <rdf:Description rdf:about="&cis;C01"/>
        <rdf:Description rdf:about="&cis;C05"/>
        <rdf:Description rdf:about="&cis;D06"/>
        <rdf:Description rdf:about="&cis;C06"/>
        <rdf:Description rdf:about="&cis;C03"/>
        <rdf:Description rdf:about="&cis;C02"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

```

```

        <rdf:Description rdf:about="&cis;D01"/>
        <rdf:Description rdf:about="&cis;C04"/>
    </owl:oneOf>
</owl:Class>
</owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#Deduced -->
<owl:Class rdf:about="&cis;Deduced">
    <rdfs:subClassOf rdf:resource="&cis;Activity"/>
</owl:Class>
<!-- http://context.org/cis#Desktop -->
<owl:Class rdf:about="&cis;Desktop">
    <rdfs:subClassOf rdf:resource="&cis;Device"/>
</owl:Class>
<!-- http://context.org/cis#Device -->

<owl:Class rdf:about="&cis;Device"/>
<!-- http://context.org/cis#EmailApplication -->
<owl:Class rdf:about="&cis;EmailApplication">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;Outlook"/>
                <rdf:Description rdf:about="&cis;Thunderbird"/>
                <rdf:Description rdf:about="&cis;MSN"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;InternetApplication"/>
</owl:Class>
<!-- http://context.org/cis#Free -->
<owl:Class rdf:about="&cis;Free">
    <rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>
<!-- http://context.org/cis#GSM -->
<owl:Class rdf:about="&cis;GSM">
    <rdfs:subClassOf rdf:resource="&cis;Network"/>
</owl:Class>
<!-- http://context.org/cis#Gateway -->
<owl:Class rdf:about="&cis;Gateway">
    <rdfs:subClassOf rdf:resource="&cis;Internet"/>
</owl:Class>
<!-- http://context.org/cis#Hardware -->
<owl:Class rdf:about="&cis;Hardware">
    <rdfs:subClassOf rdf:resource="&cis;Desktop"/>
    <rdfs:subClassOf rdf:resource="&cis;Notebook"/>
    <rdfs:subClassOf rdf:resource="&cis;PDA"/>
    <owl:disjointWith rdf:resource="&cis;Software"/>
</owl:Class>
<!-- http://context.org/cis#IMApplication -->
<owl:Class rdf:about="&cis;IMApplication">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;GAIM"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>

```

```

        <rdf:Description rdf:about="&cis;GTalk"/>
        <rdf:Description rdf:about="&cis;YM"/>
        <rdf:Description rdf:about="&cis;MSNChat"/>
        <rdf:Description rdf:about="&cis;WebMessenger"/>
    </owl:oneOf>
</owl:Class>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="&cis;InternetApplication"/>
</owl:Class>
<!-- http://context.org/cis#Indoor -->
<owl:Class rdf:about="&cis;Indoor">
    <rdfs:subClassOf rdf:resource="&cis;Location"/>
</owl:Class>
<!-- http://context.org/cis#Internet -->
<owl:Class rdf:about="&cis;Internet">
    <rdfs:subClassOf rdf:resource="&cis;Network"/>
</owl:Class>
<!-- http://context.org/cis#InternetApplication -->
<owl:Class rdf:about="&cis;InternetApplication">
    <rdfs:subClassOf rdf:resource="&cis;ApplicationRun"/>
</owl:Class>
<!-- http://context.org/cis#Intranet -->
<owl:Class rdf:about="&cis;Intranet">
    <rdfs:subClassOf rdf:resource="&cis;Network"/>
</owl:Class>
<!-- http://context.org/cis#LabWork -->
<owl:Class rdf:about="&cis;LabWork">
    <rdfs:subClassOf rdf:resource="&cis;Planned"/>
</owl:Class>
<!-- http://context.org/cis#Laboratory -->
<owl:Class rdf:about="&cis;Laboratory">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;Multimedia"/>
                <rdf:Description rdf:about="&cis;Programminglab"/>
                <rdf:Description rdf:about="&cis;VRLab"/>
                <rdf:Description rdf:about="&cis;DataCom"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#LectureHall -->
<owl:Class rdf:about="&cis;LectureHall">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;LH5"/>
                <rdf:Description rdf:about="&cis;LH2"/>
                <rdf:Description rdf:about="&cis;LH3"/>
                <rdf:Description rdf:about="&cis;LH6"/>
                <rdf:Description rdf:about="&cis;LH4"/>
                <rdf:Description rdf:about="&cis;LH1"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>

```



```

    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#Lecturer -->
<owl:Class rdf:about="&cis;Lecturer">
    <rdfs:subClassOf rdf:resource="&cis;Person"/>
</owl:Class>
<!-- http://context.org/cis#Lecturing -->
<owl:Class rdf:about="&cis;Lecturing">
    <rdfs:subClassOf rdf:resource="&cis;Planned"/>
</owl:Class>
<!-- http://context.org/cis#Location -->
<owl:Class rdf:about="&cis;Location"/>
<!-- http://context.org/cis#Meeting -->
<owl:Class rdf:about="&cis;Meeting">
    <rdfs:subClassOf rdf:resource="&cis;Planned"/>
</owl:Class>
<!-- http://context.org/cis#MeetingRoom -->
<owl:Class rdf:about="&cis;MeetingRoom">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;010312"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#MobileDevice -->
<owl:Class rdf:about="&cis;MobileDevice">
    <rdfs:subClassOf rdf:resource="&cis;Device"/>
</owl:Class>
<!-- http://context.org/cis#MobilePhone -->
<owl:Class rdf:about="&cis;MobilePhone">
    <rdfs:subClassOf rdf:resource="&cis;MobileDevice"/>
</owl:Class>
<!-- http://context.org/cis#Network -->
<owl:Class rdf:about="&cis;Network"/>
<!-- http://context.org/cis#NetworkDevice -->
<owl:Class rdf:about="&cis;NetworkDevice">
    <rdfs:subClassOf rdf:resource="&cis;Device"/>
</owl:Class>
<!-- http://context.org/cis#Not_At_Desk -->
<owl:Class rdf:about="&cis;Not_At_Desk">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&cis;locatedIn"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:complementOf>
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class>
                                    <owl:oneOf rdf:parseType="Collection">
                                        <rdf:Description rdf:about="&cis;LectureRoom"/>
                                    </owl:oneOf>
                                </owl:Class>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:complementOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </owl:equivalentClass>

```

```

        </owl:Class>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;PGLab"/>
            </owl:oneOf>
        </owl:Class>
    </owl:unionOf>
</owl:Class>
    </owl:complementOf>
</owl:Class>
    <owl:someValuesFrom>
</owl:Restriction>
</owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>
<!-- http://context.org/cis#Notebook -->
<owl:Class rdf:about="&cis;Notebook">
    <rdfs:subClassOf rdf:resource="&cis;MobileDevice"/>
</owl:Class>
<!-- http://context.org/cis#OfficeApplication -->
<owl:Class rdf:about="&cis;OfficeApplication">
    <owl:equivalentClass>
        <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
                <rdf:Description rdf:about="&cis;WordProcessor"/>
                <rdf:Description rdf:about="&cis;SpreadSheet"/>
                <rdf:Description rdf:about="&cis;PDFReader"/>
            </owl:oneOf>
        </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="&cis;ApplicationRun"/>
</owl:Class>
<!-- http://context.org/cis#OfficeRoom -->
<owl:Class rdf:about="&cis;OfficeRoom">
    <rdfs:subClassOf rdf:resource="&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#On_the_Phone -->
<owl:Class rdf:about="&cis;On_the_Phone">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&cis;use"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <rdf:Description rdf:about="&cis;MobilePhone"/>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="&cis;connectedTo"/>
                            <owl:someValuesFrom>
                                <owl:Class>
                                    <owl:unionOf rdf:parseType="Collection">
                                        <rdf:Description rdf:about="&cis;UMTS3G"/>
                                        <rdf:Description rdf:about="&cis;GSM"/>
                                    </owl:unionOf>
                                </owl:Class>
                            </owl:someValuesFrom>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </owl:equivalentClass>

```

```

        </owl:intersectionOf>
        </owl:Class>
    </owl:someValuesFrom>
</owl:Restriction>
</owl:equivalentClass>
<rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>
<!-- http://context.org/cis#OpeningEmail -->
<owl:Class rdf:about="&cis;OpeningEmail">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="&cis;run"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class>
                            <owl:unionOf rdf:parseType="Collection">
                                <owl:Class>
                                    <owl:oneOf rdf:parseType="Collection">
                                        <rdf:Description rdf:about="&cis;Outlook"/>
                                    </owl:oneOf>
                                </owl:Class>
                                <owl:Class>
                                    <owl:oneOf rdf:parseType="Collection">
                                        <rdf:Description rdf:about="&cis;Thunderbird"/>
                                    </owl:oneOf>
                                </owl:Class>
                                <owl:Class>
                                    <owl:oneOf rdf:parseType="Collection">
                                        <rdf:Description rdf:about="&cis;MSN"/>
                                    </owl:oneOf>
                                </owl:Class>
                            </owl:unionOf>
                        </owl:Class>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </owl:equivalentClass>
<rdfs:subClassOf rdf:resource="&cis;Deduced"/>
</owl:Class>
<!-- http://context.org/cis#Outdoor -->
<owl:Class rdf:about="&cis;Outdoor">
    <rdfs:subClassOf rdf:resource="&cis;Location"/>
</owl:Class>
<!-- http://context.org/cis#PDA -->
<owl:Class rdf:about="&cis;PDA">
    <rdfs:subClassOf rdf:resource="&cis;MobileDevice"/>
</owl:Class>
<!-- http://context.org/cis#Person -->
<owl:Class rdf:about="&cis;Person"/>
<!-- http://context.org/cis#Planned -->

```

```

<owl:Class rdf:about="#&cis;Planned">
  <rdfs:subClassOf rdf:resource="#&cis;Activity"/>
</owl:Class>
<!-- http://context.org/cis#PostGrad -->
<owl:Class rdf:about="#&cis;PostGrad">
  <rdfs:subClassOf rdf:resource="#&cis;Person"/>
</owl:Class>
<!-- http://context.org/cis#ProcessRun -->
<owl:Class rdf:about="#&cis;ProcessRun">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#&cis;Service"/>
        <rdf:Description rdf:about="#&cis;IMTray"/>
        <rdf:Description rdf:about="#&cis;VirtualMachine"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="#&cis;Software"/>
</owl:Class>
<!-- http://context.org/cis#Profile -->
<owl:Class rdf:about="#&cis;Profile">
  <rdfs:subClassOf rdf:resource="#&cis;Lecturer"/>
  <rdfs:subClassOf rdf:resource="#&cis;PostGrad"/>
  <rdfs:subClassOf rdf:resource="#&cis;Staff"/>
  <rdfs:subClassOf rdf:resource="#&cis;Student"/>
</owl:Class>
<!-- http://context.org/cis#Proxy -->
<owl:Class rdf:about="#&cis;Proxy">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <rdf:Description rdf:about="#&cis;160.0.226.206"/>
        <rdf:Description rdf:about="#&cis;160.0.226.207"/>
        <rdf:Description rdf:about="#&cis;160.0.226.208"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
  <rdfs:subClassOf rdf:resource="#&cis;Internet"/>
</owl:Class>
<!-- http://context.org/cis#Router -->
<owl:Class rdf:about="#&cis;Router">
  <rdfs:subClassOf rdf:resource="#&cis;NetworkDevice"/>
</owl:Class>
<!-- http://context.org/cis#Seminar -->
<owl:Class rdf:about="#&cis;Seminar">
  <rdfs:subClassOf rdf:resource="#&cis;Planned"/>
</owl:Class>
<!-- http://context.org/cis#SeminarRoom -->
<owl:Class rdf:about="#&cis;SeminarRoom">
  <rdfs:subClassOf rdf:resource="#&cis;Indoor"/>
</owl:Class>
<!-- http://context.org/cis#Server -->
<owl:Class rdf:about="#&cis;Server">
  <owl:equivalentClass>

```



```
////////////////////////////////////
-->

<!-- http://context.org/cis#010202 -->
<cis:SeminarRoom rdf:about="%cis;010202"/>
<!-- http://context.org/cis#010312 -->
<owl:Thing rdf:about="%cis;010312"/>
<!-- http://context.org/cis#020212 -->
<cis:OfficeRoom rdf:about="%cis;020212"/>
<!-- http://context.org/cis#160.0.226.202 -->
<cis:Gateway rdf:about="%cis;160.0.226.202"/>
<!-- http://context.org/cis#160.0.226.206 -->
<owl:Thing rdf:about="%cis;160.0.226.206"/>
<!-- http://context.org/cis#160.0.226.207 -->
<cis:Proxy rdf:about="%cis;160.0.226.207"/>
<!-- http://context.org/cis#160.0.226.208 -->
<cis:Proxy rdf:about="%cis;160.0.226.208"/>
<!-- http://context.org/cis#AntiVirus -->
<cis:ProcessRun rdf:about="%cis;AntiVirus"/>
<!-- http://context.org/cis#C01 -->
<cis:ClassRoom rdf:about="%cis;C01"/>
<!-- http://context.org/cis#C02 -->
<cis:ClassRoom rdf:about="%cis;C02"/>
<!-- http://context.org/cis#C03 -->
<cis:ClassRoom rdf:about="%cis;C03"/>
<!-- http://context.org/cis#C04 -->
<cis:ClassRoom rdf:about="%cis;C04"/>
<!-- http://context.org/cis#C05 -->
<cis:ClassRoom rdf:about="%cis;C05"/>
<!-- http://context.org/cis#C06 -->
<cis:ClassRoom rdf:about="%cis;C06"/>
<!-- http://context.org/cis#Celcom -->
<cis:GSM rdf:about="%cis;Celcom">
  <rdf:type rdf:resource="%cis;UMTS3G"/>
</cis:GSM>
<!-- http://context.org/cis#D01 -->
<cis:ClassRoom rdf:about="%cis;D01"/>
<!-- http://context.org/cis#D02 -->
<cis:ClassRoom rdf:about="%cis;D02"/>
<!-- http://context.org/cis#D03 -->
<cis:ClassRoom rdf:about="%cis;D03"/>
<!-- http://context.org/cis#D04 -->
<cis:ClassRoom rdf:about="%cis;D04"/>
<!-- http://context.org/cis#D05 -->
<cis:ClassRoom rdf:about="%cis;D05"/>
<!-- http://context.org/cis#D06 -->
<cis:ClassRoom rdf:about="%cis;D06"/>
<!-- http://context.org/cis#DataCom -->
<cis:Laboratory rdf:about="%cis;DataCom"/>
<!-- http://context.org/cis#Digi -->
<cis:UMTS3G rdf:about="%cis;Digi">
  <rdf:type rdf:resource="%cis;GSM"/>
</cis:UMTS3G>
<!-- http://context.org/cis#Firefox -->
<cis:Browser rdf:about="%cis;Firefox"/>
<!-- http://context.org/cis#GAIM -->
```

```
<cis:IMApplication rdf:about="%cis;GAIM"/>
<!-- http://context.org/cis#GTalk -->
<cis:IMApplication rdf:about="%cis;GTalk"/>
<!-- http://context.org/cis#IE -->
<cis:Browser rdf:about="%cis;IE">
  <rdf:type rdf:resource="%cis;InternetApplication"/>
  <cis:logInto rdf:resource="%cis;Netware"/>
</cis:Browser>
<!-- http://context.org/cis#IMTray -->
<cis:ProcessRun rdf:about="%cis;IMTray">
  <rdf:type rdf:resource="%owl;Thing"/>
</cis:ProcessRun>
<!-- http://context.org/cis#IPAddress -->
<owl:Thing rdf:about="%cis;IPAddress"/>
<!-- http://context.org/cis#LH1 -->
<cis:LectureHall rdf:about="%cis;LH1"/>
<!-- http://context.org/cis#LH2 -->
<cis:LectureHall rdf:about="%cis;LH2"/>
<!-- http://context.org/cis#LH3 -->
<cis:LectureHall rdf:about="%cis;LH3"/>
<!-- http://context.org/cis#LH4 -->
<cis:LectureHall rdf:about="%cis;LH4"/>
<!-- http://context.org/cis#LH5 -->
<cis:LectureHall rdf:about="%cis;LH5"/>
<!-- http://context.org/cis#LH6 -->
<cis:LectureHall rdf:about="%cis;LH6"/>
<!-- http://context.org/cis#LectureRoom -->
<cis:OfficeRoom rdf:about="%cis;LectureRoom"/>
<!-- http://context.org/cis#MSN -->
<cis:EmailApplication rdf:about="%cis;MSN"/>
<!-- http://context.org/cis#MSNChat -->
<cis:IMApplication rdf:about="%cis;MSNChat"/>
<!-- http://context.org/cis#Maxis -->
<cis:GSM rdf:about="%cis;Maxis">
  <rdf:type rdf:resource="%cis;UMTS3G"/>
</cis:GSM>
<!-- http://context.org/cis#Mozilla -->
<cis:Browser rdf:about="%cis;Mozilla"/>
<!-- http://context.org/cis#Multimedia -->
<cis:Laboratory rdf:about="%cis;Multimedia"/>
<!-- http://context.org/cis#Netware -->
<cis:Server rdf:about="%cis;Netware"/>
<!-- http://context.org/cis#Opera -->
<cis:Browser rdf:about="%cis;Opera"/>
<!-- http://context.org/cis#Outlook -->
<cis:EmailApplication rdf:about="%cis;Outlook"/>
<!-- http://context.org/cis#PDFReader -->
<cis:OfficeApplication rdf:about="%cis;PDFReader"/>
<!-- http://context.org/cis#PGLab -->
<cis:WifiNetwork rdf:about="%cis;PGLab">
  <owl:sameAs rdf:resource="%cis;020212"/>
</cis:WifiNetwork>
<!-- http://context.org/cis#Programminglab -->
<cis:Laboratory rdf:about="%cis;Programminglab"/>
<!-- http://context.org/cis#Safari -->
<cis:Browser rdf:about="%cis;Safari"/>
```

```
<!-- http://context.org/cis#Service -->
<cis:ProcessRun rdf:about="%cis;Service"/>
<!-- http://context.org/cis#SpreadSheet -->
<cis:OfficeApplication rdf:about="%cis;SpreadSheet"/>
<!-- http://context.org/cis#TMNet -->
<cis:WifiNetwork rdf:about="%cis;TMNet"/>
<!-- http://context.org/cis#TR1 -->
<cis:TutorialRoom rdf:about="%cis;TR1"/>
<!-- http://context.org/cis#TR2 -->
<cis:TutorialRoom rdf:about="%cis;TR2"/>
<!-- http://context.org/cis#TR3 -->
<cis:TutorialRoom rdf:about="%cis;TR3"/>
<!-- http://context.org/cis#TR4 -->
<cis:TutorialRoom rdf:about="%cis;TR4"/>
<!-- http://context.org/cis#TR5 -->
<cis:TutorialRoom rdf:about="%cis;TR5"/>
<!-- http://context.org/cis#TR6 -->
<cis:TutorialRoom rdf:about="%cis;TR6"/>
<!-- http://context.org/cis#Thunderbird -->
<cis:EmailApplication rdf:about="%cis;Thunderbird"/>
<!-- http://context.org/cis#VRLab -->
<cis:Laboratory rdf:about="%cis;VRLab"/>
<!-- http://context.org/cis#VirtualMachine -->
<cis:ProcessRun rdf:about="%cis;VirtualMachine"/>
<!-- http://context.org/cis#WebMessenger -->
<cis:IMApplication rdf:about="%cis;WebMessenger"/>
<!-- http://context.org/cis#WordProcessor -->
<cis:OfficeApplication rdf:about="%cis;WordProcessor"/>
<!-- http://context.org/cis#YM -->
<cis:InternetApplication rdf:about="%cis;YM">
  <rdf:type rdf:resource="%cis;IMApplication"/>
</cis:InternetApplication>
<!--
```


Appendix C

OWL-Z Semantic Definition

This section describes complete transformation from OWL W3C syntax into Z Model. All descriptions related to OWL constructs and axioms in this section are taken from <http://www.w3.org/TR/owl-ref/>.

[*DOMAIN*]

Class : $\mathbb{P} \text{ DELTA}$
Property : $\mathbb{P} \text{ DELTA}$
Individual : $\mathbb{P} \text{ DELTA}$

Property \cap *Class* = \emptyset
Property \cap *Individual* = \emptyset
Individual \cap *Class* = \emptyset

instances : *Class* \rightarrow $\mathbb{P} \text{ Individual}$

ObjectProperty : $\mathbb{P} \text{ Property}$
DatatypeProperty : $\mathbb{P} \text{ Property}$

ObjectProperty \cap *DatatypeProperty* = \emptyset
ObjectProperty \cup *DatatypeProperty* = *Property*

propval : *ObjectProperty* \rightarrow (*DELTA* \leftrightarrow *DELTA*)

[XSD]

$propvalD : DatatypeProperty \rightarrow (Individual \leftrightarrow XSD)$

$subClassOf : Class \leftrightarrow Class$

$\forall class1, class2 : Class \bullet$

$(class1, class2) \in subClassOf \Leftrightarrow instances(class1) \subseteq instances(class2)$

$equivalentClass : Class \leftrightarrow Class$

$\forall class1, class2 : Class \bullet (class1, class2) \in equivalentClass \Leftrightarrow$

$instances(class1) = instances(class2)$

$domain : Property \rightarrow Class$

$\forall prop : Property; class : Class \bullet domain(prop) = class \Leftrightarrow$

$prop \in ObjectProperty \Rightarrow dom(propval(prop)) \subseteq instances(class)$

$range : ObjectProperty \rightarrow Class$

$\forall prop : ObjectProperty; class : Class \bullet range(prop) = class \Leftrightarrow$

$ran(propval(prop)) \subseteq instances(class)$

[XSD]

$rangeD : DatatypeProperty \rightarrow \mathbb{P} XSD$

$\forall dprop : DatatypeProperty; data : \mathbb{P} XSD \bullet rangeD(dprop) = data \Leftrightarrow$

$ran(propvalD(dprop)) \subseteq data$

$disjointWith : Class \leftrightarrow Class$

$\forall class1, class2 : Class \bullet$

$(class1, class2) \in disjointWith \Leftrightarrow instances(class1) \cap instances(class2) = \emptyset$

inverseOf : *ObjectProperty* \leftrightarrow *ObjectProperty*

$\forall \text{prop1, prop2} : \text{ObjectProperty} \bullet (\text{prop1, prop2}) \in \text{inverseOf} \Leftrightarrow$
 $\text{propval}(\text{prop1}) = (\text{propval}(\text{prop2}))^\sim$

[XSD]

subPropertyOfD : *Property* \leftrightarrow *Property*

$\forall \text{prop1, prop2} : \text{Property} \bullet (\text{prop1, prop2}) \in \text{subPropertyOfD} \Leftrightarrow$
 $\text{prop1} \in \text{DatatypeProperty} \wedge \text{prop2} \in \text{DatatypeProperty} \Rightarrow$
 $\text{propvalD[XSD]}(\text{prop1}) \subseteq \text{propvalD[XSD]}(\text{prop2})$

subPropertyOf : *Property* \leftrightarrow *Property*

$\forall \text{prop1, prop2} : \text{Property} \bullet (\text{prop1, prop2}) \in \text{subPropertyOf} \Leftrightarrow$
 $\text{prop1} \in \text{ObjectProperty} \wedge \text{prop2} \in \text{ObjectProperty} \Rightarrow$
 $\text{propval}(\text{prop1}) \subseteq$
 $\text{propval}(\text{prop2})$

[XSD]

equivalentProperty : *Property* \leftrightarrow *Property*

$\forall \text{prop1, prop2} : \text{Property} \bullet (\text{prop1, prop2}) \in \text{equivalentProperty} \Leftrightarrow$
 $(\text{prop1} \in \text{ObjectProperty} \wedge \text{prop2} \in \text{ObjectProperty} \Rightarrow$
 $\text{propval}(\text{prop1}) = \text{propval}(\text{prop2})) \wedge$
 $(\text{prop1} \in \text{DatatypeProperty} \wedge \text{prop2} \in \text{DatatypeProperty} \Rightarrow$
 $\text{propvalD[XSD]}(\text{prop1}) = \text{propvalD[XSD]}(\text{prop2}))$

oneOf : \mathbb{P} *Individual* \rightarrow *Class*

$\forall x : \mathbb{P} \text{ Individual}; \text{class} : \text{Class} \bullet \text{oneOf}(x) = \text{class} \Rightarrow x = \text{instancesclass}$

someValuesFrom : *Class* \times *ObjectProperty* \rightarrow *Class*

$\forall \text{class1, class2} : \text{Class}; \text{prop} : \text{ObjectProperty} \bullet \text{someValuesFrom}(\text{class1, prop})$
 $= \text{class2} \Leftrightarrow \text{instances}(\text{class2}) = \{a : \text{Individual} \mid \exists b : \text{Individual} \bullet (a, b) \in$
 $\text{propval}(\text{prop}) \wedge b \in \text{instances}(\text{class1})\}$

$$\text{allValuesFrom} : \text{Class} \times \text{ObjectProperty} \rightarrow \text{Class}$$

$$\forall \text{class1}, \text{class2} : \text{Class}; \text{prop} : \text{ObjectProperty} \bullet \text{allValuesFrom}(\text{class1}, \text{prop}) = \text{class2} \Leftrightarrow \text{instances}(\text{class2}) = \{a : \text{Individual} \mid \forall b : \text{Individual} \bullet (a, b) \in \text{propval}(\text{prop}) \Rightarrow b \in \text{instances}(\text{class1})\}$$

prove by reduce;

$$\text{minCardinality} : (\mathbb{N} \times \text{ObjectProperty}) \rightarrow \text{Class}$$

$$\forall c : \text{Class}; n : \mathbb{N}; \text{prop} : \text{ObjectProperty} \bullet \text{minCardinality}(n, \text{prop}) = c \Leftrightarrow \text{instances}(c) = \{x : \text{Individual} \mid \#\{(\text{propval}(\text{prop}))(\{x\} \cap \emptyset)\} \geq n\}$$

prove by reduce;

$$\text{maxCardinality} : (\mathbb{N} \times \text{ObjectProperty}) \rightarrow \text{Class}$$

$$\forall c : \text{Class}; n : \mathbb{N}; \text{prop} : \text{ObjectProperty} \bullet \text{maxCardinality}(n, \text{prop}) = c \Leftrightarrow \text{instances}(c) = \{x : \text{Individual} \mid \#\{(\text{propval}(\text{prop}))(\{x\} \cap \emptyset)\} \leq n\}$$

prove by reduce;

$$\text{Cardinality} : (\mathbb{N} \times \text{ObjectProperty}) \rightarrow \text{Class}$$

$$\forall c : \text{Class}; n : \mathbb{N}; \text{prop} : \text{ObjectProperty} \bullet \text{Cardinality}(n, \text{prop}) = c \Leftrightarrow \text{instances}(c) = \{x : \text{Individual} \mid \#\{(\text{propval}(\text{prop}))(\{x\} \cap \emptyset)\} = n\}$$

prove by reduce;

$$\text{sameAs} : \mathbb{P} \text{Individual} \leftrightarrow \mathbb{P} \text{Individual}$$

$$\forall x, y : \mathbb{P} \text{Individual} \bullet (x, y) \in \text{sameAs} \Leftrightarrow x = y$$

$$\text{differentFrom} : \mathbb{P} \text{Individual} \leftrightarrow \mathbb{P} \text{Individual}$$

$$\forall x, y : \mathbb{P} \text{Individual} \bullet (x, y) \in \text{differentFrom} \Leftrightarrow x \neq y$$

$$\text{Transitive} : \mathbb{P} \text{ObjectProperty}$$

$$\forall \text{prop} : \text{ObjectProperty} \bullet \text{prop} \in \text{Transitive} \Leftrightarrow (\forall x, y, z : \text{Individual} \bullet (x, y) \in \text{propval}(\text{prop}) \wedge (y, z) \in \text{propval}(\text{prop}) \Rightarrow (x, z) \in \text{propval}(\text{prop}))$$

Symmetric : \mathbb{P} *ObjectProperty*

$\forall \text{prop} : \text{ObjectProperty} \bullet \text{prop} \in \text{Symmetric} \Leftrightarrow$
 $(\forall x, y : \text{Individual} \bullet (x, y) \in \text{propval}(\text{prop}) \Rightarrow$
 $(y, x) \in \text{propval}(\text{prop}))$

InverseFunctional : \mathbb{P} *ObjectProperty*

$\forall \text{prop} : \text{ObjectProperty} \bullet \text{prop} \in \text{InverseFunctional} \Leftrightarrow$
 $(\forall a, b, c : \text{Individual} \mid (a, c) \in \text{propval}(\text{prop}) \wedge$
 $(b, c) \in \text{propval}(\text{prop}) \bullet a = b)$

complementOf : *Class* \leftrightarrow *Class*

$\forall \text{class1}, \text{class2} : \text{Class} \bullet (\text{class1}, \text{class2}) \in \text{complementOf} \Leftrightarrow$
 $\text{Individual} \setminus \text{instances}(\text{class1}) = \text{instances}(\text{class2})$

intersectionOf : *seq Class* \rightarrow *Class*

$\forall \text{cseq} : \text{seq Class}; \text{class} : \text{Class} \bullet \text{intersectionOf}(\text{cseq}) = \text{class} \Leftrightarrow$
 $\text{instances}(\text{class}) = \bigcap \{x : \text{ran cseq} \bullet \text{instances}(x)\}$

Thing, Nothing : *Class*

$\text{instances}(\text{Thing}) = \text{Individual}$
 $\text{instancesNothing} = \emptyset$
 $\forall c : \text{Class} \bullet \text{instances}(c) \subseteq \text{Individual}$

hasValue : (*Class* \times *ObjectProperty*) \rightarrow *Individual*

$\forall \text{ind} : \text{Individual}; c : \text{Class}; p : \text{ObjectProperty} \bullet \text{hasValue}(c, p) = \text{ind} \Leftrightarrow$
 $\text{instances}(c) = \{a : \text{Individual} \mid \text{ind} \in \text{propval}(p) \cap \{a\}\}$

Appendix D

Z Specification of Context Ontology

Person, Network,
Activity, Location, Device : Class

(Person, Thing) ∈ subClassOf
(Network, Thing) ∈ subClassOf
(Device, Thing) ∈ subClassOf
(Activity, Thing) ∈ subClassOf
(Location, Thing) ∈ subClassOf

*Desktop, MobileDevice, NetworkDevice,
Hardware, Software, MobilePhone, Notebook, PDA, AccessPoint, Router,
Server : Class*

*(Desktop, Device) ∈ subClassOf
(MobileDevice, Device) ∈ subClassOf
(NetworkDevice, Device) ∈ subClassOf
(Notebook, MobileDevice) ∈ subClassOf
(PDA, MobileDevice) ∈ subClassOf
(MobilePhone, MobileDevice) ∈ subClassOf
(AccessPoint, NetworkDevice) ∈ subClassOf
(Server, NetworkDevice) ∈ subClassOf
(Router, NetworkDevice) ∈ subClassOf
(Software, Desktop) ∈ subClassOf
(Software, Notebook) ∈ subClassOf
(Software, PDA) ∈ subClassOf
(Hardware, Desktop) ∈ subClassOf
(Hardware, Notebook) ∈ subClassOf
(Hardware, PDA) ∈ subClassOf
« grule HardwareSoftwareDisjoint »
(Hardware, Software) ∈ disjointWith*

Lecturer, Student, Postgrad, Staff, Profile : Class

*(Student, Person) ∈ subClassOf
« grule LecturerInPerson »
(Lecturer, Person) ∈ subClassOf
(Postgrad, Person) ∈ subClassOf
(Staff, Person) ∈ subClassOf
(Profile, Staff) ∈ subClassOf
« grule ProfileInLecturer »
(Profile, Lecturer) ∈ subClassOf
(Profile, Student) ∈ subClassOf
(Profile, Postgrad) ∈ subClassOf*

ProcessRun, ApplicationRun, EmailApplication, OfficeApplication, IMApplication, InternetApplication, Browser, MailClient : Class

(ApplicationRun, Software) ∈ subClassOf
(ProcessRun, Software) ∈ subClassOf
(EmailApplication, ApplicationRun) ∈ subClassOf
(OfficeApplication, ApplicationRun) ∈ subClassOf
(InternetApplication, ApplicationRun) ∈ subClassOf
(Browser, InternetApplication) ∈ subClassOf
(IMApplication, InternetApplication) ∈ subClassOf
(MailClient, InternetApplication) ∈ subClassOf

Internet, Ethernet, GSM, Intranet, UMTS, WiFi : Class

(Internet, Network) ∈ subClassOf
(Intranet, Network) ∈ subClassOf
(GSM, Network) ∈ subClassOf
(UMTS, Network) ∈ subClassOf
(WiFi, Network) ∈ subClassOf
(Ethernet, Network) ∈ subClassOf

Planned, Deduced : Class

(Planned, Activity) ∈ subClassOf
(Deduced, Activity) ∈ subClassOf
« grule PlannedRule »
(Deduced, Planned) ∈ disjoint With

Available, Busy, Free, Browsing, Chatting, NotAtOffice, OpenEmail, OnThePhone, isBusy : Class

(*Available, Deduced*) \in *subClassOf*
 (*Free, Deduced*) \in *subClassOf*
 (*Browsing, Deduced*) \in *subClassOf*
 (*Busy, Deduced*) \in *subClassOf*
 (*Chatting, Deduced*) \in *subClassOf*
 (*NotAtOffice, Deduced*) \in *subClassOf*
 (*OnThePhone, Deduced*) \in *subClassOf*
 (*OpenEmail, Deduced*) \in *subClassOf*
 « grule *BusyFreedisjointWith* »
 (*Busy, Free*) \in *disjointWith*

Lecturing, Meeting, Research, Seminar, Tutoring, LabActivity : Class

(*Seminar, Planned*) \in *subClassOf*
 (*Meeting, Planned*) \in *subClassOf*
 (*Lecturing, Planned*) \in *subClassOf*
 (*Research, Planned*) \in *subClassOf*
 (*Tutoring, Planned*) \in *subClassOf*
 (*LabActivity, Planned*) \in *subClassOf*

Indoor, Outdoor : Class

(*Indoor, Location*) \in *subClassOf*
 (*Outdoor, Location*) \in *subClassOf*
 « grule *OutDoorIndoorDisjoint* »
 (*Indoor, Outdoor*) \in *disjointWith*

Building, Room, Lab, Classroom, SeminarRoom, LectureHall, MeetingRoom, OfficeRoom, notOfficeRoom : *Class*

(Indoor, Location) ∈ subClassOf
(Outdoor, Location) ∈ subClassOf
(Building, Indoor) ∈ subClassOf
(Room, Building) ∈ subClassOf
(Lab, Room) ∈ subClassOf
 « grule ClassroominRoom »
(ClassRoom, Room) ∈ subClassOf
(LectureHall, Room) ∈ subClassOf
(OfficeRoom, Room) ∈ subClassOf
(MeetingRoom, Room) ∈ subClassOf
(SeminarRoom, Room) ∈ subClassOf
(notOfficeRoom, OfficeRoom) ∈ complementOf

C01, C02, C03, C04, C05, C06, D01, D02, D03, D04, D05, D06 : *Individual*

C01 ∈ instances(ClassRoom);
C02 ∈ instances(ClassRoom);
C03 ∈ instances(ClassRoom);
C04 ∈ instances(ClassRoom);
C05 ∈ instances(ClassRoom);
C06 ∈ instances(ClassRoom);
 « grule D01inClassRoom »
D01 ∈ instances(ClassRoom);
D02 ∈ instances(ClassRoom);
D03 ∈ instances(ClassRoom);
D04 ∈ instances(ClassRoom);
D05 ∈ instances(ClassRoom);
D06 ∈ instances(ClassRoom);

FIREFOX, IE : *Individual*

« grule App1 »
IE ∈ instances(Browser); FIREFOX ∈ instances(Browser);

[XSD]

fullName, officeAddress, phoneNumber,
emailAddress, imAddress : DatatypeProperty
name, office, phone, email, im : P XSD

domain(fullName) = Profile
rangeD(fullName) = name
domain(officeAddress) = Profile
rangeD(officeAddress) = office
domain(phoneNumber) = Profile
rangeD(phoneNumber) = phone
domain(emailAddress) = Profile
rangeD(emailAddress) = email
domain(imAddress) = Profile
rangeD(imAddress) = im

NOVELNETWARE : Individual

« grule ServerInstance »
NOVELNETWARE ∈ instances(Server);

*use, run, connectedTo, currentActivity, locatedIn,
loginTo, currentSSID, associatedWith,
ownedBy, equippedWith : ObjectProperty*

*domain(use) = Person
range(use) = Device
domain(run) = Person
range(run) = Software
domain(connectedTo) = Person
range(connectedTo) = Internet
range(connectedTo) = Intranet
domain(currentActivity) = Person
range(currentActivity) = Activity
domain(associatedWith) = Device
range(associatedWith) = Network
domain(associatedWith) = Person
range(associatedWith) = Server
domain(equippedWith) = Room
range(equippedWith) = Desktop
domain(currentSSID) = AccessPoint
range(currentSSID) = WiFi
domain(loginTo) = Person
range(loginTo) = Server
domain(ownedBy) = Device
range(ownedBy) = Person
« grule runsubprop »
(run, use) ∈ subPropertyOf
« grule useIsTransitive »
(use) ∈ Transitive
« grule PersonRunningBrowser »
(allValuesFrom(Person, run) = Browser)
« grule PersonConnectedToInternet »
(allValuesFrom(Person, connectedTo) = Internet)
« grule PersonConnectedToIntranet »
(allValuesFrom(Person, connectedTo) = Intranet)
« grule PersonRunningOffice »
allValuesFrom(Person, run) = OfficeApplication*

```

« grule PersonCurrentActivityIsPlanned »
allValuesFrom(Person, currentActivity) = Planned
« grule PersonRunningIM »
allValuesFrom(Person, run) = IMApplication
« grule PersonRunningEmail »
allValuesFrom(Person, run) = EmailApplication
« grule PersonUseDevice »
allValuesFrom(Person, use) = Device
« grule PersonLocatedIn »
allValuesFrom(Person, locatedIn) = Location
« grule PersonLocatedInIndoor »
allValuesFrom(Person, locatedIn) = Indoor
allValuesFrom(Person, locatedIn) = Outdoor
« grule PersonLoginTo »
someValuesFrom(Person, loginTo) = Server
Browsing = someValuesFrom(Person, run) = Browser ∧
someValuesFrom(Person, connectedTo) = Internet
Busy = someValuesFrom(Person, run) = OfficeApplication ∧
someValuesFrom(Person, currentActivity) = Planned
Chatting = someValuesFrom(Person, run) = IMApplication ∧
someValuesFrom(Person, connectedTo) = Internet
« grule cardinal »
Cardinality(1, currentActivity) = Person
« grule maxRun »
minCardinality(3, run) = ApplicationRun
« grule HasValue »
hasValue(Person, loginTo) = NOVELNETWARE;

```

Appendix E

Screenshot of Proof Process

```
2. ZHVES (LaTeX mode)
Cardinality\declaration, Chatting\declaration, Busy\declaration,
BusyFroedisjointWith, Browsing\declaration, someValuesFrom\declaration,
Location\declaration, EmailApplication\declaration,
InApplication\declaration, Planned\declaration,
OfficeApplication\declaration, Browser\declaration,
allValuesFrom\declaration, subPropertyOf\declaration, select\_2\_1,
select\_2\_2, Transitive\declaration, WiFi\declaration,
AccessPoint\declaration, Desktop\declaration, Room\declaration,
Server\declaration, Network\declaration, Activity\declaration,
Intranet\declaration, Internet\declaration, Software\declaration,
Device\declaration, range\declaration, Person\declaration,
Class\declaration, Property\declaration, fun\_type, domain\declaration,
\done\declaration, ObjectProperty\declaration, '[internal items]' to ...
true
Proving gives ...
true
Beginning proof of ...
  \not allValuesFrom (Person, locatedIn) = Indoor ^^
\not allValuesFrom (Person, locatedIn) = Outdoor
Assuming OutdoorIndoorDisjoint generates ...
  (Indoor, Outdoor) \in disjointWith ^^
  \and \not allValuesFrom (Person, locatedIn) = Indoor ^^
\not allValuesFrom (Person, locatedIn) = Outdoor
Which simplifies
[forward chaining using KnownMember\declarationPart, knownMember,
'[internal items]
with the assumptions locatedIn\declaration, Person\declaration,
allValuesFrom\declaration, PersonRunningBrowser, PersonConnectedToInternet,
PersonConnectedToIntranet, PersonRunningOffice, PersonCurrentActivityIsPlanned,
PersonRunningIM, PersonRunningEmail, PersonUseDevice, PersonLocatedIn,
disjointWith\declaration, select\_2\_1, select\_2\_2, Outdoor\declaration,
Indoor\declaration, OutdoorIndoorDisjoint, '[internal items]' to ...
  Location = Indoor ^^
  \lor Location = Outdoor
Proving gives ...
  Location = Indoor ^^
  \lor Location = Outdoor
Done.
->
<
```

Figure E.1: Proofing Process in LaTeX Mode of Section 4.4.1

```

ZIVIS
File Edit Window Browser
ZIVIS (ZiAtex mode)
... axiom axiom\453
... axiom axiom\454
... axiom axiom\455
... axiom HardwareSoftwareDisjoint
declaration of Lecturer, Postgrad, Profile, Staff, Student
... axiom Lecturer\4declaration
... axiom Student\4declaration
... axiom Postgrad\4declaration
... axiom Staff\4declaration
... axiom Profile\4declaration
... axiom axiom\456
... axiom LecturerInPerson
... axiom axiom\457
... axiom axiom\458
... axiom axiom\459
... axiom ProfileInLecturer
... axiom axiom\460
... axiom axiom\461
Beginning proof of ...
(Profile, Person) \Nin subClassOf
Assuming LecturerInPerson generates ...
(Lecturer, Person) \Nin subClassOf \N
\Nimplies (Profile, Person) \Nin subClassOf
Assuming ProfileInLecturer generates ...
(Profile, Lecturer) \Nin subClassOf \N
\Nland (Lecturer, Person) \Nin subClassOf \N
\Nimplies (Profile, Person) \Nin subClassOf
Which simplifies
forward chaining using KnownMember\4declarationPart, knownMember,
[internal items]
with the assumptions Person\4declaration, subClassOf\4declaration,
select\N_1, select\N_2, Lecturer\4declaration, LecturerInPerson,
Profile\4declaration, ProfileInLecturer, [internal items] to ...
(Profile, Person) \Nin subClassOf
Done.
=>

```

Figure E.2: Proofing Process in L^AT_EX Mode of Section 4.4.2

```

ZIVIS
File Edit Window Browser
ZIVIS (ZiAtex mode)
\Nland (Local run, Local use) \Nin subPropertyOf \N
\Nland Local use \Nin Transitive \N
\Nland allValuesFrom (Person, Local run) = Browser \N
\Nland allValuesFrom (Person, Local connectedTo) = Internet \N
\Nland allValuesFrom (Person, Local connectedTo) = Intranet \N
\Nland allValuesFrom (Person, Local run) = OfficeApplication \N
\Nland allValuesFrom (Person, Local currentActivity) = Planned \N
\Nland allValuesFrom (Person, Local run) = MailApplication \N
\Nland allValuesFrom (Person, Local use) = Device \N
\Nland allValuesFrom (Person, Local locatedIn) = Location \N
\Nland allValuesFrom (Person, Local locatedIn) = Indoor \N
\Nland allValuesFrom (Person, Local locatedIn) = Outdoor \N
\Nland someValuesFrom (Person, Local loginTo) = Server \N
\Nland Browsing = someValuesFrom (Person, Local run) \N
\Nland someValuesFrom (Person, Local run) = Browser \N
\Nland someValuesFrom (Person, Local connectedTo) = Internet \N
\Nland Busy = someValuesFrom (Person, Local run) \N
\Nland someValuesFrom (Person, Local run) = OfficeApplication \N
\Nland someValuesFrom (Person, Local currentActivity) = Planned \N
\Nland Chatting = someValuesFrom (Person, Local run) \N
\Nland someValuesFrom (Person, Local run) = MailApplication \N
\Nland someValuesFrom (Person, Local connectedTo) = Internet \N
\Nland Cardinality (1, Local currentActivity) = Person \N
\Nland minCardinality (3, Local run) = ApplicationRun \N
\Nimplies (Person, Local loginTo) \Nin \4das hasValue
Beginning proof of ...
NOVELNETWORKARE \Nin instances Server
Which simplifies
forward chaining using KnownMember\4declarationPart, knownMember,
[internal items]
with the assumptions Server\4declaration, instances\4declaration,
NOVELNETWORKARE\4declaration, ServerInstance, [internal items] to ...
true
Proving gives ...
true
Done.
=>

```

Figure E.3: Proofing Process in L^AT_EX Mode of Section 4.4.3

```

ZIVIS (ZiTeX mode)
Beginning proof of ...
D01 \in instances Room
Assuming D01inClassRoom generates ...
  D01 \in instances ClassRoom \&
\implies D01 \in instances Room
Assuming ClassRoominRoom generates ...
  (ClassRoom, Room) \in subClassOf \&
  \land D01 \in instances ClassRoom \&
\implies D01 \in instances Room
Assuming instancesubclass with the instantiations:
c = ClassRoom, d = Room, ind = D01 generates ...
  (
    ClassRoom \in Class \&
    \land Room \in Class \&
    \land D01 \in Individual \&
    \land (ClassRoom, Room) \in subClassOf \&
    \land D01 \in instances ClassRoom \&
    \implies D01 \in instances Room
  )
  \land (ClassRoom, Room) \in subClassOf \&
  \land D01 \in instances ClassRoom \&
\implies D01 \in instances Room
Rearranging gives ...
  (ClassRoom, Room) \in subClassOf \&
  \land D01 \in instances ClassRoom \&
  \land (
    ClassRoom \in Class \&
    \land Room \in Class \&
    \land D01 \in Individual \&
    \land (ClassRoom, Room) \in subClassOf \&
    \land D01 \in instances ClassRoom \&
    \implies D01 \in instances Room
  )
\implies D01 \in instances Room
Substituting produces ...
  (ClassRoom, Room) \in subClassOf \&
  \land D01 \in instances ClassRoom \&
  \land (
    ClassRoom \in Class \&
    \land Room \in Class \&
    \land D01 \in Individual \&
    \implies D01 \in instances Room
  )
\implies D01 \in instances Room
Which simplifies
[forward chaining using KnownMember\&declarationPart, knownMember,
[internal items]]
with the assumptions Individual\&declaration, Class\&declaration,
instances\&declaration, D01\&declaration, D01inClassRoom,
subClassOf\&declaration, select\&_2\&_1, select\&_2\&_2, Room\&declaration,
ClassRoom\&declaration, ClassRoominRoom, [internal items] to ...
true
Proving gives ...
true
Done.

```

Figure E.4: Proofing Process in L^AT_EX Mode of Section 4.4.3

```

ZIVIS
File Edit Window Browser
ZIVIS (ZiTeX mode)
  \land hasValue (Person, loginTo) = NOVELNETWARE \&
  \implies hasValue (Lecturer, loginTo) = NOVELNETWARE \&
  \land (Lecturer, Person) \in subClassOf \&
\implies hasValue (Lecturer, loginTo) = NOVELNETWARE
Rearranging gives ...
  (Lecturer, Person) \in subClassOf \&
  \land (
    Lecturer \in Class \&
    \land Person \in Class \&
    \land loginTo \in ObjectProperty \&
    \land NOVELNETWARE \in Individual \&
    \land (Lecturer, Person) \in subClassOf \&
    \implies hasValue (Person, loginTo) = NOVELNETWARE \&
    \implies hasValue (Lecturer, loginTo) = NOVELNETWARE
  )
\implies hasValue (Lecturer, loginTo) = NOVELNETWARE
Substituting produces ...
  (Lecturer, Person) \in subClassOf \&
  \land (
    Lecturer \in Class \&
    \land Person \in Class \&
    \land loginTo \in ObjectProperty \&
    \land NOVELNETWARE \in Individual \&
    \land hasValue (Person, loginTo) = NOVELNETWARE \&
    \implies hasValue (Lecturer, loginTo) = NOVELNETWARE
  )
\implies hasValue (Lecturer, loginTo) = NOVELNETWARE
Which simplifies
[forward chaining using KnownMember\&declarationPart, knownMember,
[internal items]]
with the assumptions hasValue\&declaration, HasValue, Individual\&declarati
NOVELNETWARE\&declaration, ServerInstance, ObjectProperty\&declaration,
loginTo\&declaration, Class\&declaration, subClassOf\&declaration,
select\&_2\&_1, select\&_2\&_2, Person\&declaration, Lecturer\&declaration,
LecturerinPerson, [internal items] to ...
true
Proving gives ...
true
Done.
->

```

Figure E.5: Proofing Process in L^AT_EX Mode of Section 4.4.3