2009:020

MASTER'S THESIS

Development of a Knowledge Based Tool to Assist Spacecraft Attitude Control Subsystem Design

Ms Khadija Tahera

Luleå University of Technology

Master Thesis, Continuation Courses
Space Science and Technology
Department of Space Science, Kiruna

2009:020 - ISSN: 1653-0187 - ISRN: LTU-PB-EX--09/020--SE

Cranfield University

Khadija Tahera

Development of a knowledge based tool to assist Spacecraft Attitude Control Subsystem design

School of Engineering

MSc in Astronautics and Space Engineering

MSc Thesis

Cranfield University

School of Engineering

MSc Thesis

2008

Khadija Tahera

Development of a knowledge based tool to assist Spacecraft Attitude Control Subsystem design

Supervisor: **Dr. Jennifer Kingston**

Academic Year 2007 / 2008

This thesis is submitted in partial fulfilment of the requirements for the degree of MSc

ABSTRACT

To achieve the goals of effective Attitude Determination and Control subsystem design (ADCS), a methodology of knowledge based system has been proposed in this project. An open-licence knowledge representation software tool, Protégé is used to develop an ontology. This ontology presents the vocabulary for representing and communicating knowledge about the ADCS subsystem design by defining a set of relationships that hold among the terms in that vocabulary. This ontology system is modeled by the pre-existing expert's domain knowledge and concepts, to be used in the early stages of ADCS subsystem design with a goal of improved design. The utility and potential applications of the ontology have been evaluated, and its effectiveness in representing ADCS design knowledge assessed.

The ontology is expected to generate a greater understanding of the process of knowledge capture and representation, in the context of ADCS systems, and also to identify the current state-of-the-art and methodologies used in practical field. It is anticipated that the work will be a useful tool that can be used to represent ADCS subsystem knowledge during design processes or training.

ACKNOWLEDGEMENTS

I am grateful to my supervisor Dr. Jennifer Kingston for all her support during my work and want to specially thank her.

Thanks to Dr. S.E. Hobbs who provided a nice guideline about thesis writing and referencing and Dr. Peter Roberts for suggestions during my study.

Thanks to my Towhid, may be without his support it was not possible to continue my study and our family.

Love to my daughter Nazia.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

LIST OF NOTATION

ADCS	Attitude Determination and Control Subsystem
AI	Artificial Intelligent
API	Application Programming Interface
CDF	Concurrent Design Facility
CLEPE	Conceptual LEvel Programming Environment
CMG	Control Moment Gyros
CoM	Centre of Mass
CO-ODE	Collaborative Open Ontology Development Environment
CSS	Coarse Sun Sensor
CSV	Comma Separated Value
DBMS	Database Management Syatem
DL	Description Logic
DMC	Direction Cosine Matrix
ESA	European Space Agency
FOV	Field of View
NLP	Neuro Linguistic Programming
OWL	Web Ontology Language
RDF	Resource Description Framework
SHriMP	Simple Hierarchical Multi-Perspective
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
XML	Extensible Markup Language

Chapter 1 Introduction

Since the 1980s, research groups have been working on knowledge based systems and associated methodology to build intelligent computer systems. Knowledge based system is a methodology of representing behaviour of a system which is already recognized by the experts of a domain via estimation of operation and performance. It can be used in the early stages of design, with an objective of improved design.

This thesis aims to explore the potential of the knowledge based system in ADCS system engineering. Early chapters describe the literature and background of the knowledge based system and afterwards the potential of this system for use in spacecraft system design will be shown.

1.1 Background

All space missions consist of a set of concepts, and the arrangement of these concepts form a space mission architecture (Wertz and Larson, 1999). The practical scenario of ADCS subsystem design needs involvement of expert team, with access of several kinds of documentation like scientific paper, previous mission's reports, books and lot of studies to gather the information about the domain and the concepts. So during the design process large amount of pre-existing knowledge is accessed to get the adequate information about specific concepts of the domain.

The initial design is just for first order approximation and to show how each system is related to each other. That will help to build up the concept of the system and subsystems, estimate the subsystems' size, weight, cost, power requirement and actually integrating the concepts of the subsystem (Wertz and Larson, 1999). During the concept development the developer must develop alternative methods considering requirements and needs to show the trade-offs of the methods. It is crucial because by evaluating the initial design, if the method satisfies the needs at an acceptable cost it passes through to the detailed design phases.

For example, if a mission requires a very high accuracy sensor, based on the requirements of the mission it is essential to choose the appropriate sensors from the thousands of available sensors in market. Maybe few of them satisfy the requirements but which one is the most cost efficient and meets the choice of the user needs to be considered. And a designer must need to find the best option for effective design.

Currently most of the system is redesigned for every new mission from scratch, though it is a fact that the overall preliminary design procedure and selecting procedure varies a little mission to mission. During the design procedure the knowledge is captured from the domain experts could outline the model on a piece of paper but it only works when the design is considerably small and experts are willing to give lot of time and hard work. So it would be useful if the knowledge is already in the computer and directly translated to the program for further processing.

As stated earlier, the most common approach in satellite engineering is to design a completely new model for each mission, according to the requirements. There are some commercial satellite platforms designed as a series using the same concepts and procedure. But in traditional approaches, it is usually difficult to design software with future thinking and this can cause the misconstruction problem like Ariane5. As it is very difficult to change and validate the code, the developer prefers to design the system and start writing the code from the beginning, though every new design needs time, effort, expertise and this all relates to the overall cost. So for reusing software for future missions it is desirable to change the existing software according to the requirements of the future mission; this is relatively easy and less effort than designing a from the beginning.

Most of the space companies are depending on their experts who have worked in this field for many years, and in most cases the whole preliminary design methods and hardware selection procedure is saved in the expert's brain. Maybe there are documents that state which hardware are chosen but why? and how? may not be recorded. Knowledge can be lost e.g by a particular expert moving/retiring and no one is there to recover the information except the designer. So, the knowledge based system is here to work like a designer with all of his thinking and to represent the whole model of the design with all the details.

1.2 Aim and Objectives

The aim is to design a methodology where Protégé-OWL, an open-licence knowledge representation software tool, is used as a knowledge representation system. The aim of the project is to build an intelligent ontology, a tool to support the ADCS designer task by providing information about the ADCS domain. The ontology will be a central repository with shared knowledge of the experts. Through reusing the ontology, the knowledge and expertise can also be reused for the future missions and will save a significant amount of time by ignoring same kind of analysis each time. This tool will help to understand the design steps and requirements of systems, the relationships between different

subsystems and their attributes. This might be useful as learning and decision making tool.
To be more specific, the objectives of the thesis can be summarized:

To represent the ADCS design information and knowledge using a knowledge based system by specifying the concepts of the domain.

To illustrate how each and every concept exists in the system with properties and constraints and how they are related to each other.

To develop a model for spacecraft system design, mainly emphasising the preliminary design procedure of the attitude determination and control subsystem, and to evaluate and validate the system using one or two previous missions.

To show the possible ways of sharing the knowledge from the ontology to other software tools like Excel and Matlab.

So the idea is to design an ontology which will help the design procedure of ADCS and that can be reused for future missions and will show the overall and alternative preliminary design paths.

1.3 Methodology

Though knowledge based representation systems now are widely used in different fields, mostly medical and bioinformatics, it is quite new in engineering especially in the space field. Detailed knowledge of the space system is required to represent the knowledge in knowledge based system design.

The research mainly looked at the Attitude Determination and Control Subsystem (ADCS), the methodologies are illustrated bellow:

1. First step involves extensive analysis of the Attitude Determination and Control subsystem from supporting books, thesis and journals to gather the concepts that are needed to describe the domain.

2. This step includes top-down break down of the system, subsystems, elements, functions and the relationships. Ontology will be implemented based on the concepts about the system, subsystems, equipments and components by imposing the conditions that are required to describe a class as a concept of the domain.

3. Introduce rules for intelligent classifications of knowledge and simple calculations which are required to support the design, for example disturbances that affect the attitude of the spacecraft, size of the wheels that are required to reject the disturbances.

4. To develop a model, mainly emphasising the preliminary design procedure of the Attitude Determination and Control Subsystem, and to evaluate and validate the system using one or two previous missions.

5. Perform required calculations and numerical analysis using Excel by using the data from the ontology.

6. Use the knowledge in the ontology in the simulation of the ADCS model in MATLAB/SIMULINK.

7. In parallel, examine the literature for additional ideas and concepts relevant to the study. This will also allow the current investigation to be placed in the context of a longer-term, more detailed project.

Figure 1.1 shows the overall data flow diagram of this project.

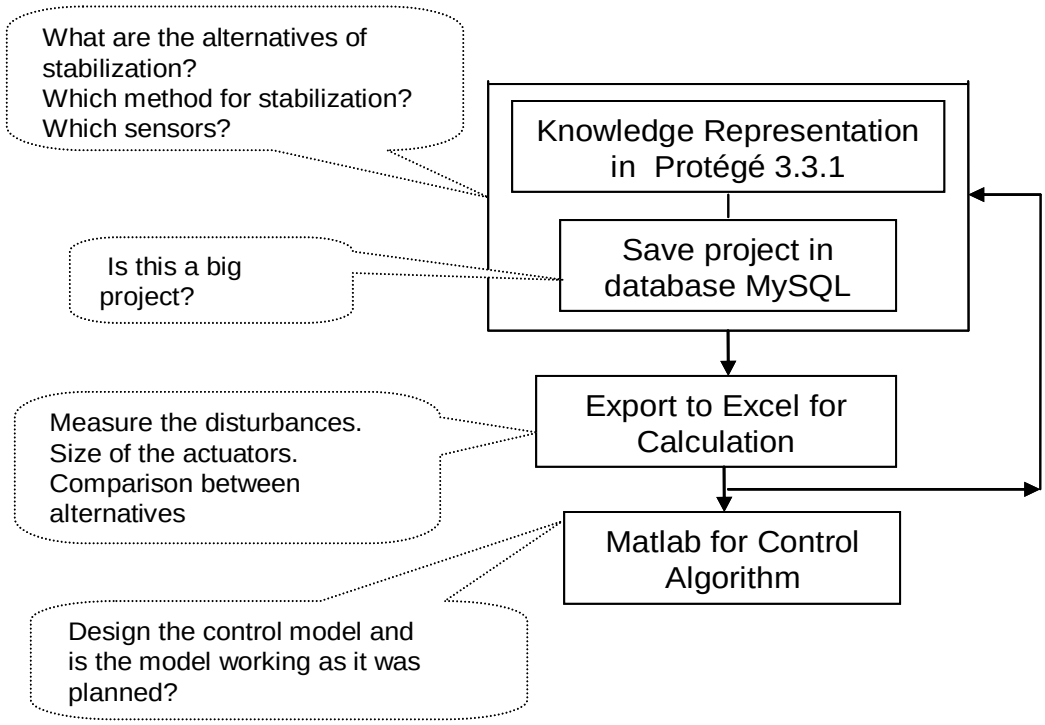


Figure 1.1 The overall flow diagram of the project.

1.4 Thesis outline

This report follows the objectives of this project sequentially:

Chapter 2 reviews the literatures of related field, the basics of knowledge based systems and brief discussion about ADCS subsystem.

The proposed design procedure and the concerns during the design phase are presented in Chapter 3.

An example of ADCS subsystem design using the designed model is illustrated in Chapter 4.

Discussion about the results and validation procedure is presented in Chapter 5.

Chapter 6 contains the conclusion remarks and some suggested further work.

Chapter 2 Literature Review

The reviews of related topics of this project are presented in this chapter. Basic discussion about the knowledge representation system, ontology, Protégé and ADCS subsystem are given in the following sections.

2.1 Knowledge Representation System

The term "*Knowledge representation*" tells how the knowledge of an existing thing can be presented. The term Knowledge representation is well known and comes from the Artificial Intelligence (AI) field. "The knowledge representation and reasoning is the area of Artificial Intelligence (AI) concerned with how knowledge can be presented symbolically and manipulated in an automated way by reasoning programs. More informally, it is the part of AI that is concerned with thinking and how thinking contributes to intelligent behaviours"(Ronald J. Brachman and Hector J. Levesque, 2004; Ronald J. Brachman and Hector J. Levesque, 2004).

The basic question is "What is it?" Davis, Shrobe, and Szolovits claimed that "We believe that the answer is best understood in terms of the five fundamental roles that it plays:

- A knowledge representation (KR) is most fundamentally a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: In what terms should I think about the world?
- It is a fragmentary theory of intelligent reasoning, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency is supplied by the guidance a representation provides for organizing information so as to facilitate making the recommended inferences.
- It is a medium of human expression, i.e., a language in which we say things about the world." (Davis et al.1993)

Knowledge is one form of intelligence about existing things and intelligence is associated with logics which drive a set of inferences. Knowledge is presented in

the machine language through an ontology, and then this ontology is used in the practical problem solving method which needs the knowledge about the domain.

"Knowledge representation is a multidisciplinary subject that applies theories and techniques from three other fields:

1. Logic provides the formal structure and rules of inference.
2. Ontology defines the kinds of things that exist in the application domain.
3. Computation supports the applications that distinguish knowledge representation from pure philosophy " (John, 2000).

2.2 Ontology

Ontology defines the terms which exist in reality to represent the knowledge of the area. An ontology formally defines the terms and concepts of the domain that are used to describe and present the domain. Tom Gruber defines the term as "*An ontology is a specification of a conceptualization*" (Gruber, 1995). The conceptualization is the interpretation of the concepts or terms which provides the knowledge of the domain and the specification is the explicit conceptualization of the terms. It encodes the knowledge of the domains and also that expands the domains (Heflin, 2003).

Informally, ontology is a language which describes the existing objects, the properties and attributes of the objects and the relationships constraints of the objects of an area. The ontology has a significant kind of structure with (Heflin, 2003):

- Classes (definition of the existing objects)
- The relationships that exist among the objects.
- The properties of the object.

2.2.1 Why develop an ontology?

An ontology defines the terms and vocabulary of a domain in a machine interpretable form, which is a useful field for various reasons (Noy and Mcguinness, 2001) :

- To share the common understanding among people of same domain of interest.
- To reuse a model of the knowledge of a domain.
- To make explicit the assumptions of the domain.
- To separate the domain knowledge from the operational knowledge.
- To analyse and expand the domain knowledge.

2.2.2 Steps of developing an ontology

There are various recommendation and choices for building ontology (Pinto and Martins, July, 2004)(Noy and Mcguinness, 2001) and all of them use almost the same ideas. The formal approach of building a system engineering ontology is slightly different and maintains the steps as Figure 2.1 sequentially (Sarder and Ferreira 2007(a)). Detailed description of these steps are given in (Sarder et al., 5-9 Aug. 2007)

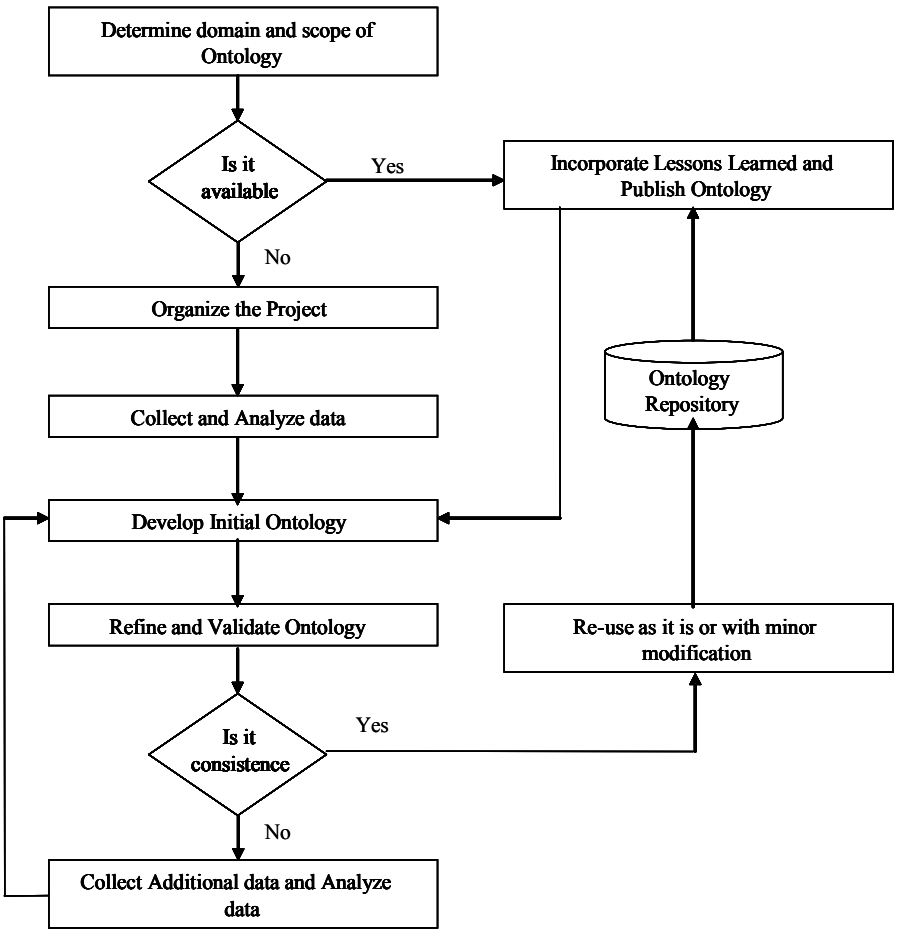


Figure 2.1 Steps of ontology development (Sarder and Ferreira 2007(a))**.**

2.3 Protégé

Since the effectiveness of knowledge based system was realized, a large number of tools have been developed for building and maintaining ontologies. Though ontologies are constructed to represent the domain knowledge and as problem solving methods, the optimization of the operational component and the user interface of the ontology tools become more significant to assist the user as well as to keep up with competitor tools in this field.

2.3.1 Why Protégé

Each of the ontology development tools has its strengths and weaknesses. Among the several tools, Protégé was chosen (by the supervisor and (Verrier, 2001)) for this project by considering some evaluation results which have been done by other parties. Evaluations are influenced by a number of parameters. Considering the amount of time and knowledge that is required to learn the tools and relative difficulties, five tools are evaluated in (Duineveld et al., 2000). The results are shown in the Figure 2.2.

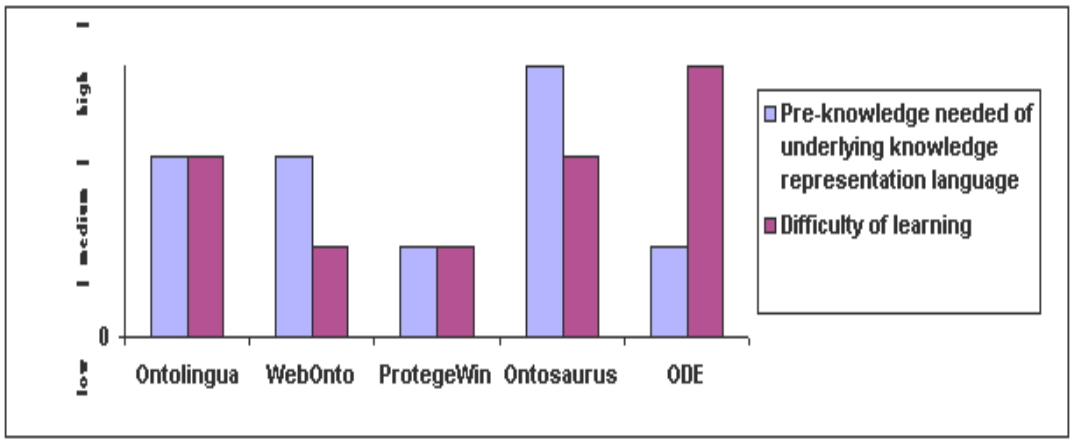


Figure 2.2 Results of the Evaluation of different tools. (Duineveld et al., 2000)

The other criteria that are evaluated in for Protégé, Chimaere, OilEd and DAG-Edit are Availability, Functionality, Multiple inheritance, Data model, Reasoning, Example ontologies, Reuse, Formats, Visualization, Help, Shortcuts, Stability, Customization, Extendibility, Multiple users (Lambrix et al., 2003). According to this document the main strengths of Protégé are excellent user interface, expandability using plug-ins, enhanced functionality and user-friendliness.

Table 2-1 shows some questions that were asked to the user of these four tools. Same ontology in Bioinformatics field was used for evaluation of the tools.

There are several documents on evaluation and comparisons of different tools are presented in following website (Protege Team, 2008b(a)).

Table 2-1 Evaluation of Graphical User Interface (Lambrix et al., 2003)

Parameters	**Protégé**	**Chimaere**	**OilEd**	**DAG-Edit**
Does the tool solve the tasks?				
(never =1, sometimes=5, always=10)	9	9	9	8.9
The functions in the tool to create and manage ontologies are				
(too few =1, right=5, too many=10)	5.5	5	6.5	6.4
Parameters	**Protégé**	**Chimaere**	**OilEd**	**DAG-Edit**
Efficiency				
Do you get feedback on your operations?				
(never=1, sometimes=5, always=10)	5	5.6	6.5	5.8
To navigate between the different windows in the tool feels				
(difficult=1, easy=10)	9.4	8.4	7.8	6.4
The overview over the concepts, attributes and instances feels				
(bad=1, good=10)	8.5	7	5.6	6.3
Attitude				
The user interface designed in a uniform way.				
(do not agree=1, agree=10)	8.5	7.5	7.3	7.4
The available help feels				
(not enough=1, adequate=10)	7.4	-	-	7
Learnability				
Is it easy to understand the meaning of the icons and menus?				
(difficult=1, easy=10)	6	6	5	4
Is the terminology easy to understand and adapted to the user?				
(never=1, sometimes=5, always=10)	7.8	6.8	6.6	6.6
To use the help feels				
(difficult=1, easy=10)	8	-	-	6.6
To learn the interface for ontology management feels				
(difficult=1, easy=10)	8	6.4	6	5.3
To remember how to use the interface feels				
(difficult=1, easy=10)	9	8.4	7.6	5

2.3.2 About Protégé

Protégé is a free and open source platform to build knowledge based applications with ontologies, was developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. Though the development of Protégé was mainly for bioinformatics, a large number of users from different fields including engineering are successfully using Protégé for knowledge based systems.

Since the 1980s, a series of Protégé versions have been developed from one generation to the next with increased capabilities, additional knowledge acquisition constraints, more declarative, more explicit and more user-friendly. Good reports on the Protégé lineage are (Fensel and Straatman, 1998; Lambrix et al., 2003; Musen et al., 2000; Musen, 1989). The latest version of Protégé is written in Java and platform independent allowing developers to create reusable intelligent ontologies and problem solving methods (Musen, 2000(b)). A rich website is available including all documentations and sources.

There are two ways of modelling ontologies in Protégé, using *Protégé Frame* editor and *Protégé OWL* editor. Both of the editors describe the classes, properties and the individuals.

Protégé Frame editor supports to build frame based domain ontologies, customizing data entry form and entering the individual's data, whereas *Protégé OWL* also check the consistency of the ontologies by introducing logical reasoning.

The idea of this ontology development was to present the knowledge of ADCS design as well as include the logical reasoning that support the processing of knowledge during design phase therefore the more suitable Protégé OWL editor was chosen for this project.

Web Ontology Language (OWL) is a species of the knowledge representation language that was developed for the Semantic Web applications authorized by World Wide Web consortium (Bechhofer et al., 2004). "The Semantic Web is a web of data" or information which is easily accessible, processable by machine, representing on the World Wide Web. More about OWL and Semantic web can be found in (Antoniou and Harmelen, 2004) and (Smith et al., 2004) . OWL is the extension of XML (Extensible Markup Language) (Bray et al., 2000) and RDF (Resource Description Framework) (Lassila and Swick, 1999)

OWL language is further divided into three sublanguages depending on the expressability: OWL Lite, OWL DL, and OWL full (Smith et al., 2004). OWL Lite supports simple reasoning and constraints features. OWL DL supports reasoning with maximum expressiveness with computational completeness, has

the decidability with description logic. OWL full is the most expressive language but without computational completeness therefore might not be logical. Details description on OWL language can be found (Smith et al., 2004). Among the three types, the OWL DL was chosen for this project and the preference of OWL DL was focused by the requirement of completeness of the logical computation and also reasonable expressivity.

2.3.3 The Architecture of the Protégé OWL

As mentioned in Section 2.3.2, OWL is the extension of core Protégé editor and also includes a collection of custom tailored plugins of its own. Figure 2.3 presents the overall architecture of OWL with Protégé core system. By using the OWL on top of Core Protégé the facilities of Core Protégé is merged with OWL (as shown in Figure 2.3). As Protégé is an open development platform, the user and developer community also enhanced the capability of OWL to a great extent by building plugins (Protege Team, 2008a).

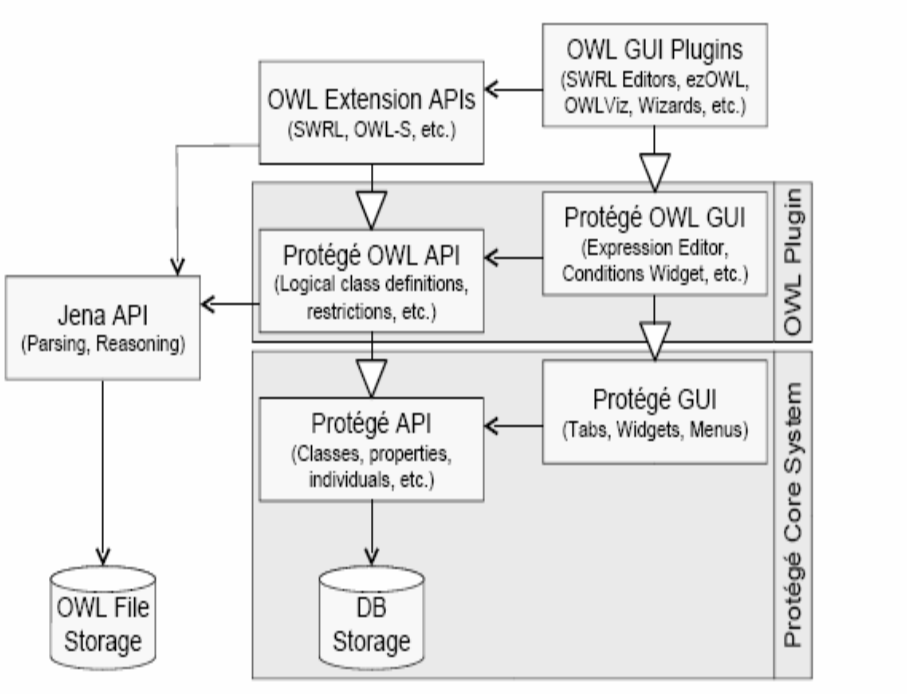


Figure 2.3 The OWL Plugin, the extension of Core Protégé (Protege Team, 2008a).

When an ontology is created or saved in Protégé it actually generates two files, a project file and a source file (Protege Team, 2008a). The project file (with the extension .pprj) stores the information about the interface customizations and editor options. When the ontology is required to be sent to others, usually this file is not required to be sent with the ontology unless the forms of the ontology are also edited. The source file (with extension .owl, .rdf, .rdfs) stores all the

information about the classes, properties, individuals and the relations that are defined for the ontology.

Protégé supports load, save, and edit and create in various format like RDF, XML, UML, OWL (Knublauch et al., 2004b). These are all file based and limited by size of file. Protégé allows a highly scalable relational database backend to create large ontologies with hundreds to thousands of classes. The ontology which is created or saved in database format, usually create only one table and can be used further for database applications.

Protégé OWL supports importing ontologies which enables users to share and reuse of ontologies thus all the classes, properties and individuals are imported for direct use in the current ontology (Knublauch et al., 2004a(a)).

2.3.4 The Plugins

As Protégé is open source, a large number Java based Application Programming Interface (API) is available to extend Protégé. Hundreds of Plugins are available, only some of them are used in this project depending upon the requirements of the project. The brief descriptions of the Plugins are presented in the following sections, but the details about how these Plugins are used in this project will be presented in the Methodology chapter.

OWLclasses, OWLproperties, OWLindividuals, OWLform are basic tabs and populated when OWL project starts. But other Plugins require populating in the working environment. Some of them are already integrated with Protégé but most of them must be separately installed. All the Plugins need to install in the Plugin folder of Protégé install directory. When they are installed in the proper directory, then they can be populated in the Protégé working editor by selecting from Project->Configure->Tab widgets menu.

2.3.4.1 Queries Tab

Though the name is Queries Tab, it allows user to query the ontology and export the result to the spreadsheets in Figure 2.4. It finds the existing individuals that match with the queries. By Fewer and More button this tab allows to reduce or increase the constraints of the query respectively. The **E** button on the top of the right corner exports the results to the spreadsheet in CSV (Comma Separated Values). It is required to select the properties (or slot for Protégé Frame) that need to export. The Tab does not select the associated properties automatically. It includes the properties name on the first row of the columns. This tab comes with the standard distribution of Protégé 3.3.1(or higher) software, therefore is easy to populate and use.

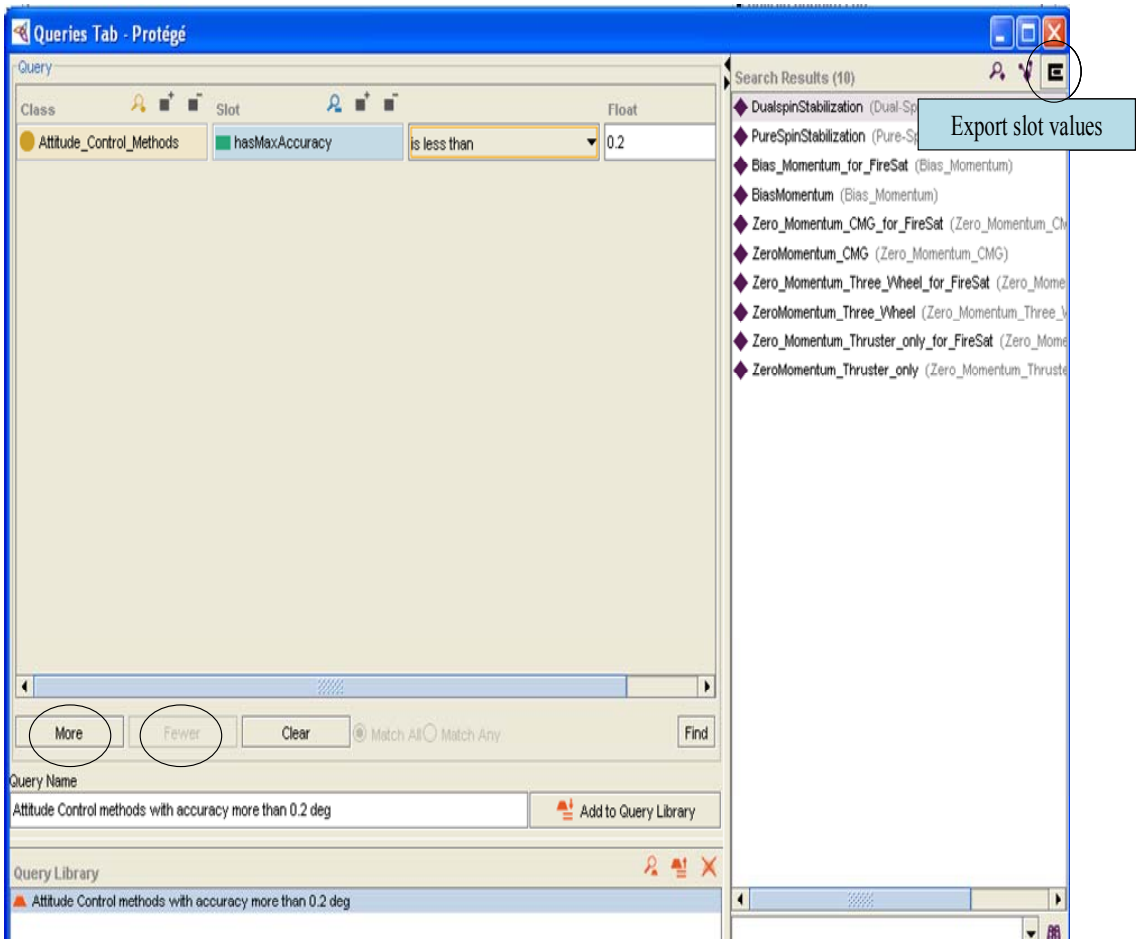


Figure 2.4 Queries Tab

2.3.4.2 InstanceXL Tab

As the name says it brings a look of Excel. This Tab in Figure 2.5, allows viewing and exporting the individuals of a class with the associated and assigned properties. It also allows exporting individuals only in CSV format with all directly related properties values but not the properties name.

The filter button permits to remove the properties for viewing and exporting without removing from the ontology. This tab does not come with the Protégé software, and needs to be installed from the distributors' website (Martin and Adrian, 2008).

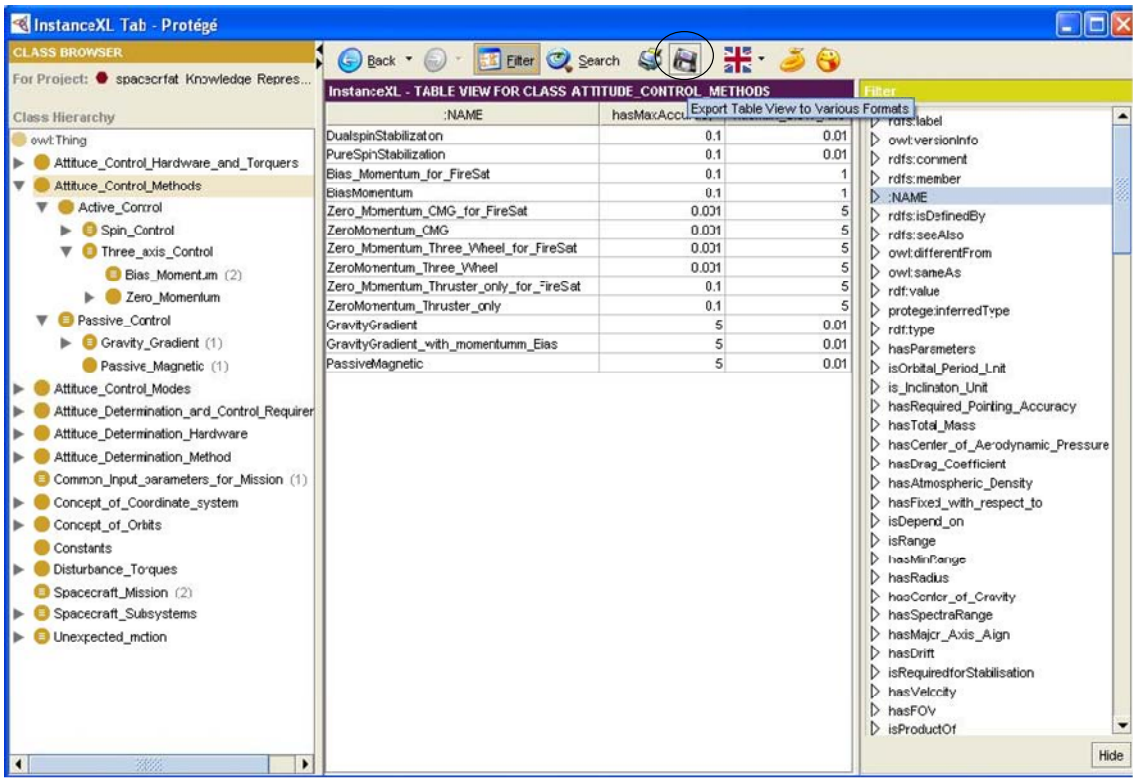


Figure 2.5 InstanceXL Tab

2.3.4.3 DataMaster v1.2 Tab

Importing data from a table is frequently required when a large amount of processed data is saved in a table. DataMaster plugin supports import of data into the ontology from most of the relational database or table (Nyulas et al., 2007). In Figure 2.6, screenshot of DataMaster plugin is shown. First of all a connection with the database is required to be established through *Connection Panel*. The *Superclass Selector Panel* allows to select the superclasses for the table class (Nyulas et al., 2007), and *Import Location Selector Panel* specify some other importing options. Details with example are given in Section 4.1.4.2.

2.3.4.4 SWRL Tab

SWRL stands for Semantic Web Rule Language. SWRL is based on OWL and as OWL is the ontology language for the Semantic web it is planned to be the rule language of the Semantic Web. It allows writing the rules for the existing individuals of the OWL ontology to infer new knowledge about the individuals (O'Connor, 2008). SWRL rules are based on classes, properties and individuals. This tab is part of Protégé 3.3.1, initially is disabled and must be activated for use (O'Connor, 2008). Figure 2.7 shows the SWRLTab interface in protégé.

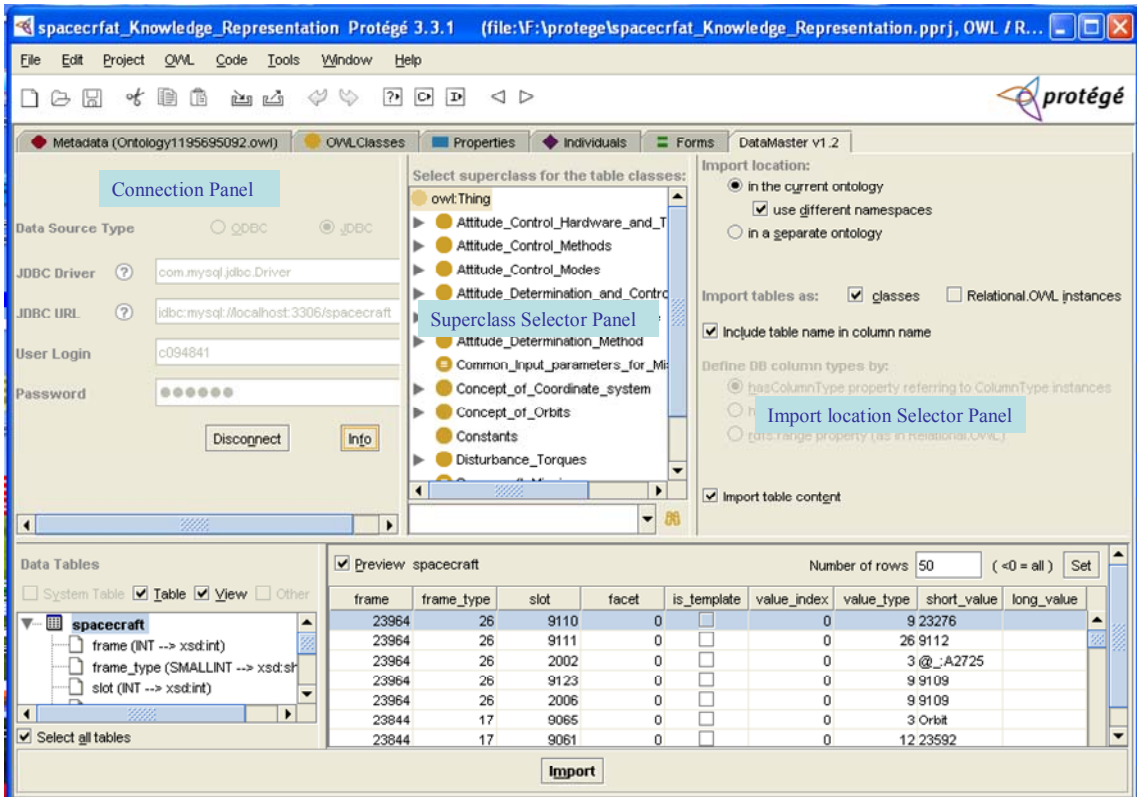


Figure 2.6 DataMaster Plugin

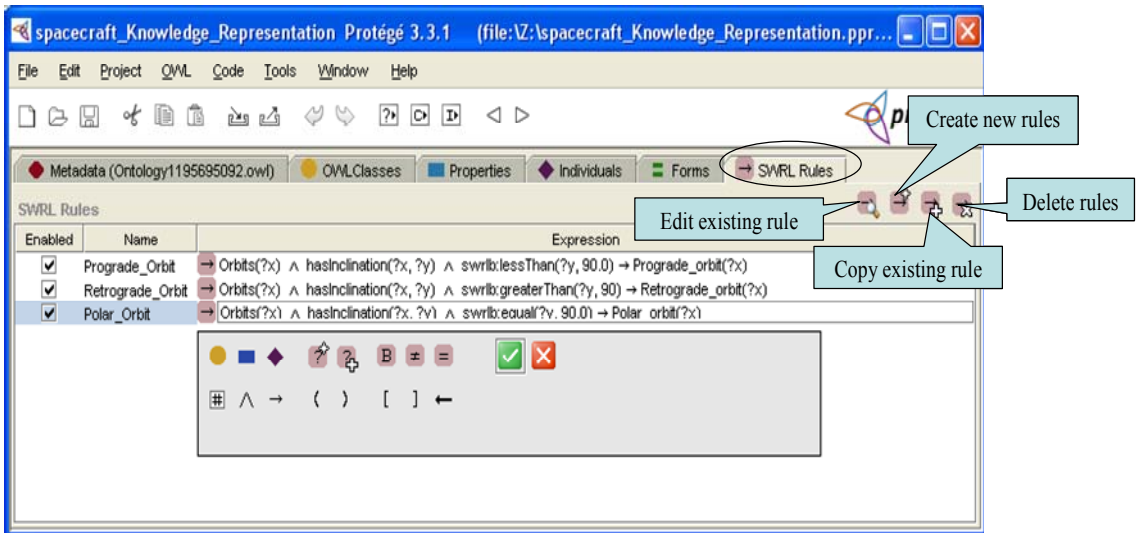


Figure 2.7 SWRLTab

2.3.4.5 OWLviz

Figure 2.8 shows the OWLViz plugin, which is designed for Protégé OWL to view the class hierarchies graphically. It uses different colour schemes for defined and primitive classes. It creates *asserted* class and *inferred* class

(discussed in Section 3.3) hierarchies to compare them. The inconsistent concepts are highlighted in red to distinguish clearly (Horridge, 2008).

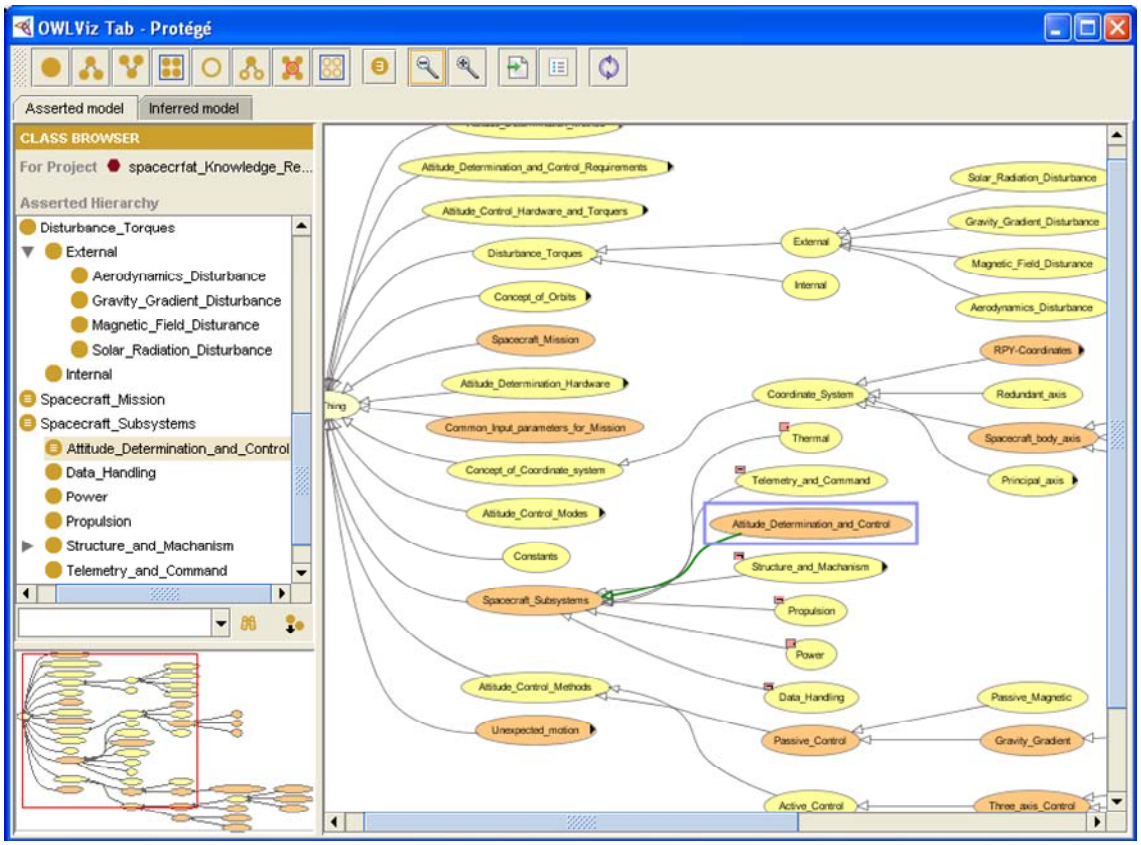


Figure 2.8 OWLviz Tab

2.3.4.6 Jambalaya

It is a challenging task to understand a complex ontology as the user has to clearly understand from the high level as well as the detailed knowledge of the ontology. Also to enable the reuse of an ontology especially which is developed by others is a complicated task (Storey et al., 2004).

This plugin is created for Protégé to visualize the ontology using the Shrimp tool. SHriMP (Simple Hierarchical Multi-Perspective) is a domain-independent visualization technique designed to visualise the large software architecture. It enhances the capability of browsing and exploring the complex information spaces (CHISEL Authors, 2008).

It helps to investigate the interface between the classes, properties and individuals of ontology in different views (Storey et al., 2001). Figure 2.9 shows the relationships between the superclasses.

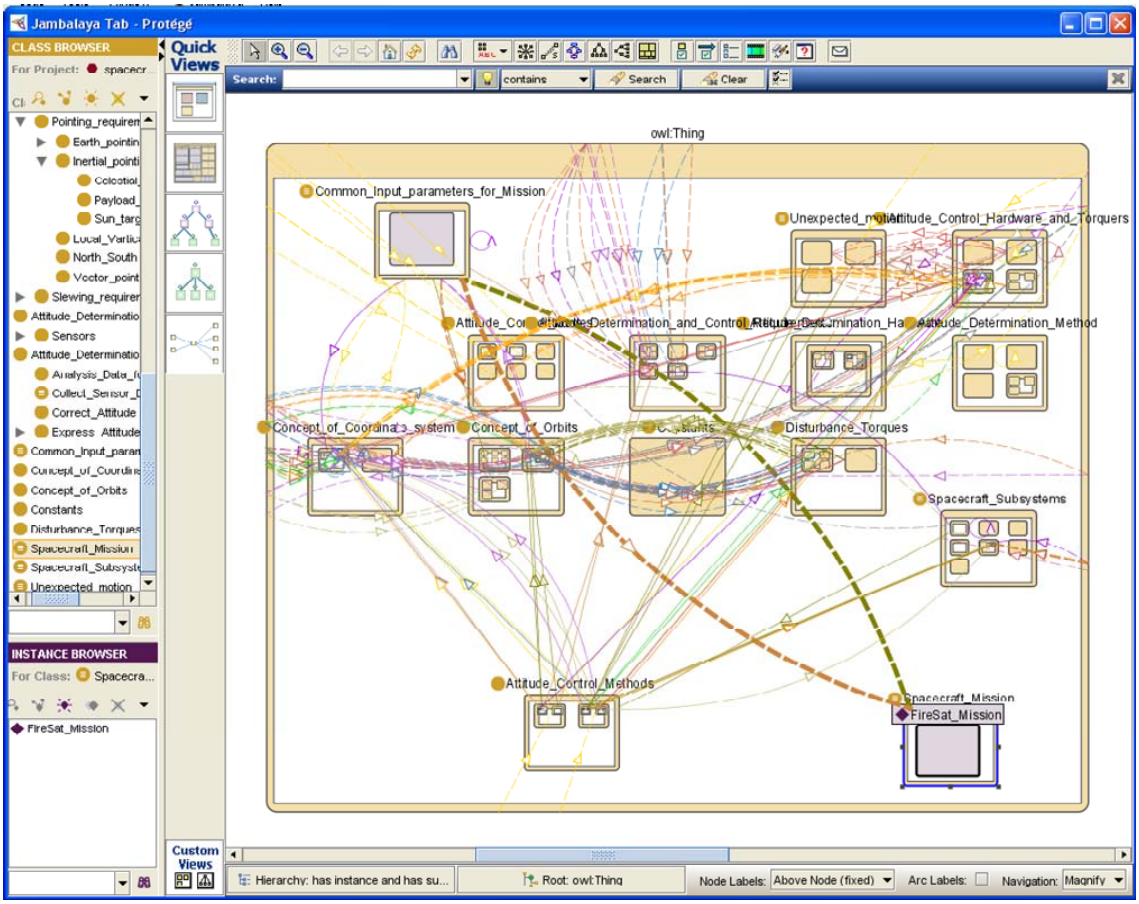


Figure 2.9 Jambalaya Tab

2.3.5 Why database?

When the ontology becomes very large then it is required to save in a database format. There are several advantages and disadvantages of both file base and database system.

File Base system

Advantages

1. Fast processing time and simple
2. No conversion and creation effort
3. No need of database software, so cost saving

Disadvantages

1. Limited size, not possible to create very large ontology
2. Only stand alone mode
3. Does not support the parallel access of multiple users on the same file.

Database system

Advantages

1. Possible to create very large ontology.
2. Multiple and collaboration mode works nicely.
3. Can be used the data from the database for further analysis easily.

Disadvantages

1. Slower then file base
2. Extra cost of database software
3. Little background of database software is required.

Protégé OWL supports work simultaneous on the same database, facilitating real time collaboration (Robert et al., 2005) which is not possible to do in file based system. In database format all classes, properties, and individuals are saved in a single database table, most of the time the database table is not accessed by the front user and detailed knowledge and proper understanding of the database system and its functionality is not required. The database ontology not only solves the limitation of file based project but also helps to enhance the functionality of Protégé by using collaborative mode or multiple user modes.

2.3.6 Existing knowledge based projects using ontologies

Thousands of projects have been using the outstanding facilities of knowledge based systems since the evolution of ontology came across in the engineering field. The ultimate purpose of building an ontology for engineering field is "*To provide a basis of building models of all things in which computer science is interested*" (Mizoguchi and Ikeda, 1997). This technology brings several benefits which have been realized by different level of engineering. Only a few existing judgments are given in this section.

An overview of interview system, MULTIS has been developed for practical problem solving methods in (Mizoguchi et al., 1995). Mizoguchi presented that ontology engineering is to "*bridge the gap between domain experts and computers to enable computers to extract domain experts' ways of problem solving using task ontology. This can be interpreted in another way: MULTIS can help end user describe how they perform a task at the conceptual level without considering how computer works*" (Mizoguchi et al., 1995).

Conceptual LEvel Programming Environment (named CLEPE), is an ontology to investigate the property of problem solving methods using knowledge based systems (Ikeda et al., 1998).CLEPE provides three major advantages of: "*(A) It provides human-friendly primitives in terms of which users can easily describe their own problem solving process (descriptiveness, readability). (B) The systems with task ontology can simulate the problem solving process at an abstract level*

in terms of conceptual level primitives (conceptual level operationality). (C) It provides ontology author with an environment for building task ontology so that he/she can build a consistent and useful ontology" (Seta et al., 1996)

A model of a prototypical AI/NLP system has been proposed by Pazienza, M.T. to support expert teams during scientific design tasks. It was envisioned for the practical scenario of the SHUMI project of the Concurrent Design Facility (CDF) of the European Space Agency (ESA)/ESTEC. A relevant step of this project was to build a domain ontology which will able to represent the knowledge emerging from the design process through pre-existing knowledge repositories (that is the ontology itself) (Pazienza et al., 2005). Significant benefits have been expected from this project, such as extraction of implicit information, and analysis of explicit information which is expressed through natural language.

A standard Intelligent System ontology for automatic combat vehicles has been developed using Protégé to provide (Schlenoff et al., 2005) :

- A set of standard domain concepts with their attributes and relationships
- Capturing the knowledge of the domain and reuse.
- Facilities of system specification, design and integration to accelerate the research in this field.

The intelligent system contains all the knowledge of intelligent behaviour that is usually done by well trained crew. To allocate these intelligent behaviours to an automated machine it was necessary to retrieve and describe knowledge from domain experts (Schlenoff et al., 2005).

A knowledge based modelling of space transportation systems has been developed to provide support during the early stages of design (Ruiz-Torres et al., 2006). This model uses the knowledge based logics and equations to determine the ground processes, and their duration, allowing the estimation of operational measurements performances. To develop such a model it is necessary to represent the characteristics that have been recognized by operations experts as these have effects on ground process and during the conceptual and early phases of the design process. A significant benefit in this project is the involvement of the vehicle designer not only for development and manufacturing assessment but also the operational assessments (Ruiz-Torres et al., 2006). Therefore this procedure helps to take the decisions of operation level earlier during the design phases.

Although significant works have not been done in the space field using knowledge based systems (as per authors knowledge), by analysing literature in different engineering field it is realized that the knowledge based systems can offer a lot for future space technology.

2.4 Basics of Attitude Determination and Control Subsystem

A brief description of ADCS is given in this section which is essential to design the ontology. The Attitude Determination and Control System (ADCS) is a subsystem of spacecraft system design, and probably the most important part of the mission is attitude stabilization and control of the Spacecraft. It is requires to achieve and maintain the desired attitude of the spacecraft. The payload of the spacecraft will be required to point in some direction and need to maintain a specified accuracy. Different types of disturbance perturb the orientation of the payload and the spacecraft from the original direction of pointing. The ADCS determine and maintain the attitude of the spacecraft. Figure 2.10 shows the loop of ADCS.

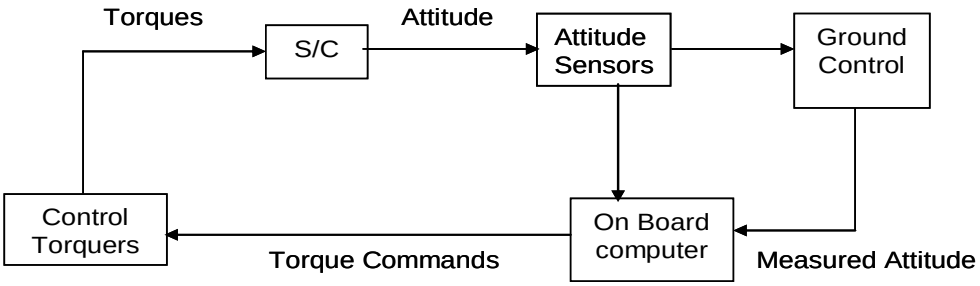


Figure 2.10 General Attitude Determination and Control System of Spacecraft

The ADCS can be considered as two parts: (1) Attitude Determination, (2) Attitude Control. The determination hardware (Sensors) determine the current attitude, then calculate the pointing errors due to the internal and external disturbances, then the controller calculates the required torque to counter them and actuators produce the necessary torque to remove error to the maintain the spacecraft in correct orientation.

2.4.1 Attitude control Methods

There are several methods for attitude control available, mainly chosen according to mission requirement, orbit and pointing of payload. Several control method might be suitable for a specific mission. Two types of control system are practiced, passive and active.

2.4.1.1.1 Passive Methods

These types of stabilization methods are mainly dependent upon the inertia properties of the spacecraft and environmental torques which act to compensate the disturbance and provide the required accuracy. The advantages of these

methods are that they are simple, cheap, long life but lower pointing accuracy and limited maneuverability restrict the use of these methods.

2.4.1.1.2 Gravity Gradient

An object in the gravitational field tends to align its long axis along with the Earth's centre (Wertz and Larson, 1999) and this principle is used in gravity gradient stabilization method, thus use the inertial properties of spacecraft. The yaw remains uncontrolled so a variation of gravity gradient method is to use a small momentum wheel spinning about the pitch axis.

2.4.1.1.3 Passive Magnetic

A permanent magnet is used on board to align the spacecraft with the Earth's magnetic field. It is most effective in near equatorial orbit and provides a constant orientation for the vehicle (Wertz and Larson, 1999).

2.4.1.2 Active Methods

Continuous monitoring is required for these types of control. Large number of control and determination hardwares are involved so cost is major factor here, but they provide enhanced usability including accurate pointing, fast maneuvering. Life time is dependent on the hardware and thrusters that are used for the method.

2.4.1.2.1 Spin Stabilization

The whole spacecraft rotates about the axis with the largest moment of inertia so that the angular momentum vector remains fixed in inertial space, which provides gyroscopic stiffness in two axes to resist disturbance (Wertz and Larson, 1999). Energy dissipation makes this system unstable so the vehicle should be short and fat but launch vehicles are tall and thin. Dual spin stabilization methods overcome this problem and allow spin about minor axis, where one part spins relatively fast and other part spins slowly. These methods are also comparatively cheap, simple and can survive for long time but with the lack of pointing accuracy and constraints of maneuvering.

2.4.1.2.2 Three axis stabilization

All three axes are precisely controlled and can keep pointed in specific direction thus these systems are more common today, however they are also more expensive, less reliable and obviously complex. A combination of reaction wheel, momentum wheel, control moment gyros, magnetic rods, thrusters are usually used in these system. Several variations of these systems are used depending on the combination of torquers. One uses momentum bias by using a

momentum wheel to control yaw axis and another one is called zero momentum, and uses three reaction wheels to control attitude (Wertz and Larson, 1999). Both systems also use thrusters or magnetic torquers to desaturate the wheels and other control hardware.

2.4.2 Disturbance Torques

It is required to determine and characterise the disturbance torques that spacecraft must have to tolerate and which is mainly influenced by the spacecraft orientation, mass properties and the design symmetry (Wertz and Larson, 1999) shown in Table 2-2. The external torques due to environmental effects are generally gravity gradient torque, solar radiation pressure, aerodynamics and magnetic torque.

Table 2-2 External disturbance torques.

Disturbance	Primarily Influenced by (Wertz and Larson, 1999)	Dominant Altitude (Fortescue and Swinerd, 2003)
Solar Radiation	Spacecraft surface reflectivity, Spacecraft geometry and the location of the center of gravity	> synchronous all height
Aerodynamics	Orbit altitude, Spacecraft geometry and the location of the center of gravity	< about 500 km
Gravity Gradient	Orbit altitude, The magnitude of the moment of inertia of each axis of spacecraft	500-35000 km
Magnetic field	Orbit Altitude, Orbit Inclination, Residual spacecraft magnetic dipole	500-35000 km

Internal torques are mainly due to mechanism, fuel movement, astronaut movement, flexible appendage, general mass movement and thruster misalignments. Usual technique is to try to specify the greatest disturbance, then use that value for sizing (Wertz and Larson, 1999).

2.4.3 Attitude Sensors

Sensors generally determine the attitude and the pointing direction. The sensors are typically selected depending upon the required accuracy and their

performance. For the full range of attitude knowledge more than two sensors and a combination of different sensors are required.

Sun sensors measure the angle between the incident sunlight and the mounting base. Accuracy is typically $<0.001^{o}$, but need clear fields of view so during eclipse period another source of determination method is required.

Earth sensors provide Earth related information and the position of the nadir vector. The location of the horizon is determined by the difference between the IR emission of the Earth and the cold deep space (Brown, 2002). Typical accuracy is $< 0.1^{o}$.

Star sensors determine the attitude of the vehicle using known stars in field of view. They can be scanner, tracker and mapper and use different methods to determine the attitude. They are the most accurate sensors (Attitude accuracy =1 arc sec) (Wertz and Larson, 1999).

All above mentioned types are reference sensors. Another type of sensor is the inertial sensor, which measures the rotational motion using gyros and position and velocity using accelerometers. These provides very accurate measurement, but need updating periodically as they drift errors build up over time.

.

2.4.4 Actuators

The actuators are the torque producing elements, generally thrusters, momentum exchange and environmental torquers. Solar radiation, magnetic torques, and gravity gradient pressure are the main environmental torques used as passive actuators. Depending on the torque authority and capability there are several types of momentum exchange devices are available to produce torque against disturbance. Thrusters are the mass expulsion actuators (using hot or cold gases).

Reaction wheels are torque motors with a high inertia rotor, and can rotate in either direction; each wheel can provide control of one axis (Wertz and Larson, 1999). These devices exchange the momentum by changing the speed of the wheel.

Momentum wheel rotates at a nonzero speed, which provides gyroscopic stiffness to two axes. This device only differs from reaction wheel by the biased speed.

Control moment gyros are single or double gimbaled wheels spinning at a constant rate, produce high torque, rarely used in small spacecraft because of weight.

Magnetic torquers are used to compensate the magnetic torque produced by the Earth's magnetic field. Usefulness of this hardware is limited by the altitude of the orbit and the magnetic field strength. They are also used to desaturate the momentum exchange device, but require long time.

Thrusters produce the most effective, large and instant torque by expelling mass, and are used in most of the spacecraft. During large angle and fast maneuver, thrusters are the only choice, and they can also provide torque to control the attitude, dasaturate the wheels and control the spin rate but with a cost penalty.

2.4.5 Attitude Determination Methods

Attitude determination is a process of estimating the attitude of the spacecraft with respect to the known reference frame (Brown, 2002). Attitude is determined by the following steps:

1. Collect the data from the sensors
2. Determine the location of spin axis or body frame axis from the data.
3. Express the attitude as a set of vectors using transformation method
4. Correct the attitude

The basic information of the ADCS subsystem and the knowledge based system has been provided in this chapter. Next chapter follows the details of design procedure and some other issues.

Chapter 3 Methodology

This chapter discusses the methodology of this project. A brief discussion of working procedure is presented in Section 1.3 and has been followed in the following sections.

3.1 Basic elements of Protégé OWL

Before going to the details of working method of Protégé OWL, a brief description of the elements of OWL are given in the following sections. An ontology is the combination of classes, individuals and properties.

3.1.1 Individuals

Individuals are the existing objects of interest. Depending on the individuals, the ontology is build up and all the classification, relations and constraints are all about individuals.

Figure 3.1 shows the Individual Tab in Protégé OWL, where FireSat_Mission (I_1) is an individual of class Spacecraft_Mission (C) which is linked with individual Orbit_FireSat (I_2) by the property hasOrbit (P).

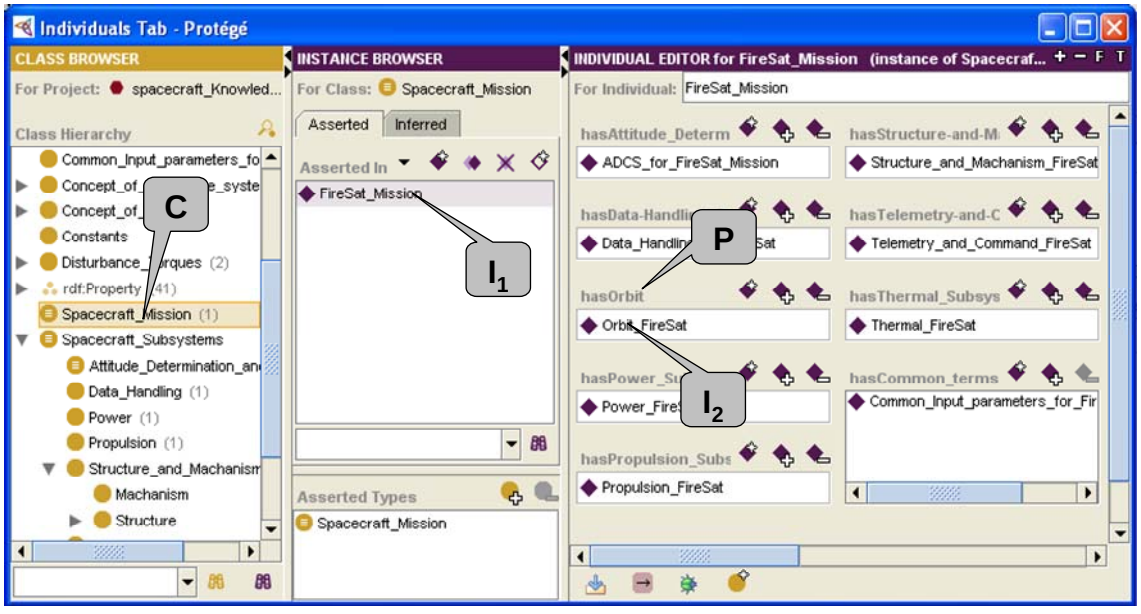


Figure 3.1 Individuals Tab in Protégé OWL.

3.1.2 Classes

Classes can be interpreted as the sets where individuals exist. A class is explicitly explained and defined for the individuals to be the member of that class. The classes are further classified as *subclasses* and *superclasses*. For example there are reaction wheels, momentum wheels and control moment gyros which are all wheels but with different types, therefore they can be divided into subclasses of Wheel class, and all the wheels are kind of Actuator so the *Actuator* class is the superclass of *Wheel* class. Figure 3.2 represents the idea of superclass, class and subclass.

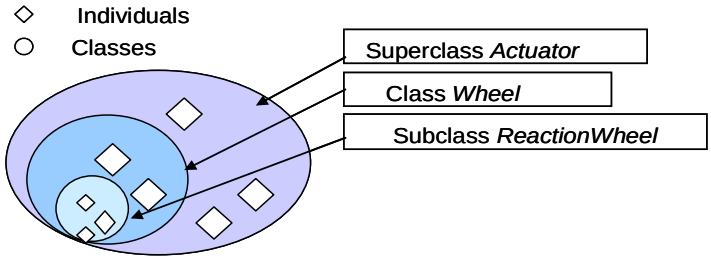


Figure 3.2 Representation of Superclass, Class and Subclass

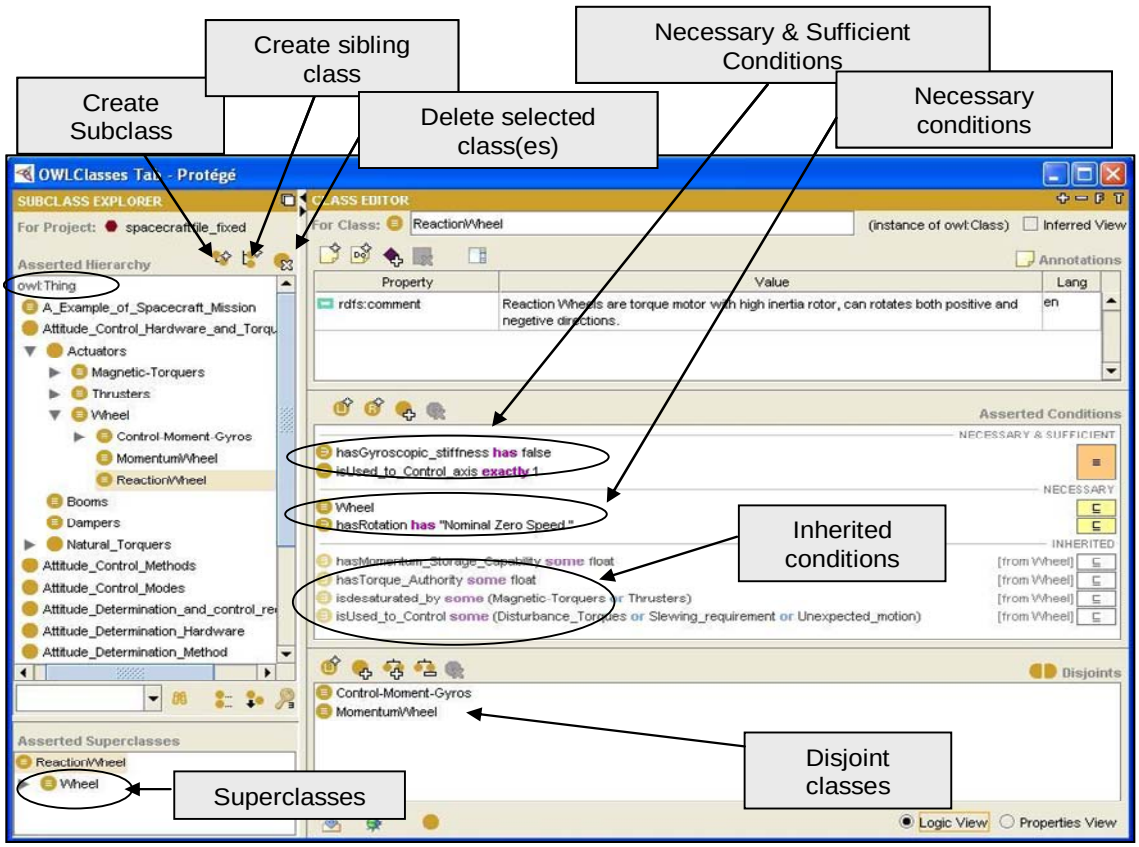


Figure 3.3 OWLClasses tab

In Protégé OWL, the OWLClasses tab looks like Figure 3.3. The empty ontology always contains only one class called owl:Thing, which is the superclass of all classes.

3.1.2.1 Defining and describing classes

It is required to describe and define a class properly so that by definition an individual belongs to a class. To define the classes different types of restrictions are used. OWL properties are used to create restrictions. Restrictions in OWL are mainly three categories: (1) Quantifier Restrictions (2) Cardinality restrictions and (3) hasValue Restrictions.

1. Quantifier Restrictions

As the name suggest this restriction indicates the range of the individuals, composed of a quantifier (a property) and filler. Two quantifiers are used in OWL: The *existential quantifier* (∃), can be read like 'someValuesFrom' and The *Universal quantifier*(∀), which can be read as 'allValuesFrom' in OWL language. The existential restriction is most common in defining the classes, tells about the existence of an individual through a property within a class. For example, ∃ hasWheel some ReactionWheel describe the class of individuals that have some, or at least one wheel that is from the class *ReactionWheel*, here hasWheel is the quantifier and ReactionWheel class is the filler. By the Universal quantifier, for example ∃ hasWheel only ReactionWheel, if an individual of this class have a wheel then that has to be only an individual from *ReactionWheel* class.

2. Cardinality Restrictions

Three types of cardinality restriction exist in OWL, minimum, maximum and exactly. These restrictions specify the number of relationships that an individual must hold for a given property. The terms are described in Figure 3.5.

3. hasValue Restrictions

A class is related to a specific individual of another class through this restriction.

Figure 3.4 Condition Widget tool

The 'Condition Widget' tool in Figure 3.4 shows all the possible restrictions and is a very useful tool for editing and displaying the description of the classes. The descriptions of the operators used in 'Condition Widget' tool are given below in Figure 3.5.

Protégé-OWL Syntax

OWL Element	Symbol	Key	Example	Meaning of example
allValuesFrom	∀	*	children ∀ Male	All children must be of type Male
someValuesFrom	∃	?	children ∃ Lawyer	At least one child must be of type Lawyer
hasValue	∋	$	rich ∋ true	The rich property must have the value true
cardinality	=	=	children = 3	There must be exactly 3 children
minCardinality	≥	>	children ≥ 3	There must be at least 3 children
maxCardinality	≤	<	children ≤ 3	There must be at most 3 children
complementOf	¬	!	¬ Parent	Anything that is not of type Parent
intersectionOf	⊓	&	Human ⊓ Male	All Humans that are Male
unionOf	⊔	|	Doctor ⊔ Lawyer	Anything that is either Doctor or Lawyer
enumeration	{...}	{ }	{male female}	The individuals male or female

Figure 3.5 Description of the Protégé OWL syntax.

A class with a necessary condition means, to be a member of this class it is necessary to fulfil the condition. This type of class is called Primitive class. With the necessary conditions, it is not possible to say that if an individual fulfil the conditions of a class, then it must be a member of this class. But if an individual fulfil the necessary and sufficient conditions of a class then it must be a member of that class. This type of classes are called defined classes.

3.1.2.2 Disjoint Classes

Sometime it is required to disjoint concepts or objects from each other, so that an individual of one class cannot be also an individual of its disjoint class. For example a reaction wheel can not be also a momentum wheel; therefore the classes of these individuals are disjoint, shown in Figure 3.3.

3.1.3 Properties

Properties link two individuals together. Mainly two types of properties are there in Protégé OWL, *Object properties* and *Datatype properties*. *Object properties* link an individual with another individual. For example James isStudentOf of Cranfield University, where the object property isStudentOf links James with Cranfield University and James as an individual of Student class and Cranfield University is an individual of University class. *Datatype properties* links an

individual with datatype value. James *hasStudentID* 091234, here *hasStudentID* is a *Datatype property* which tells that James's student ID is 091234.

There is also another type of property *Annotation property*, which is used to add information about an individual. Types of properties are shown in Figure 3.6. The properties can be also divided into Subproperties and Superproperties like classes.

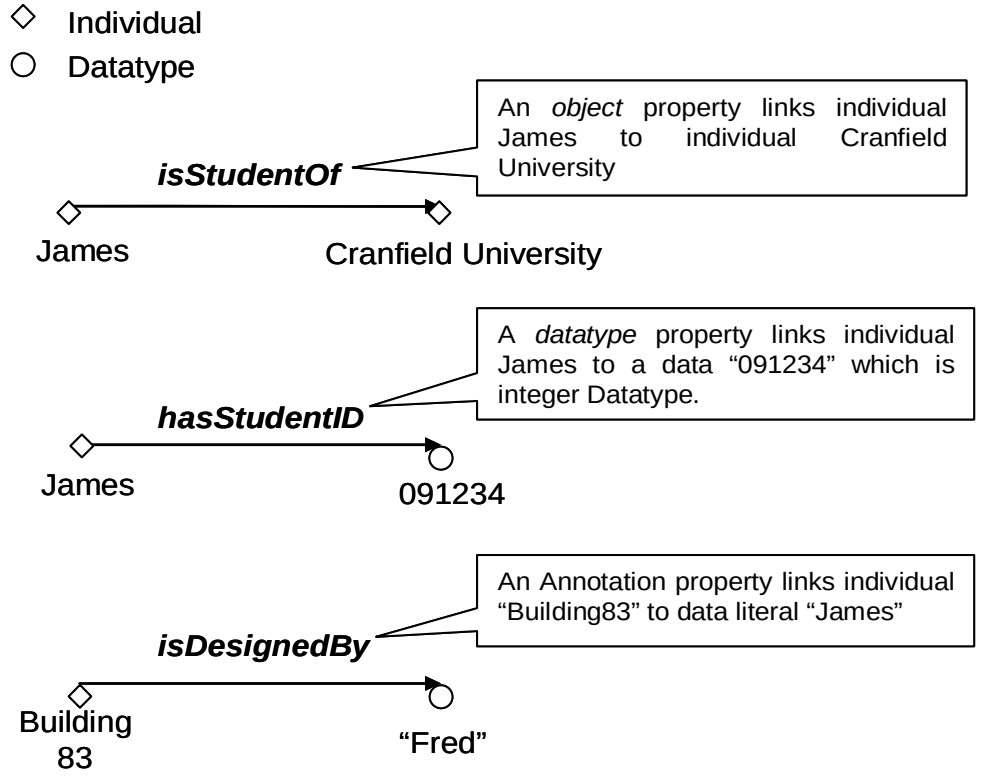


Figure 3.6 Types of Properties in Protégé OWL

1.4.1.1 Inverse Properties

Each object property can have an inverse property, but it is not necessary. If an individual a is linked with b then b can be linked with a via inverse property. For example in Figure 3.7 John is linked with Jean by *hasMother* property then Jean can be connected with John by *hasChild* property.

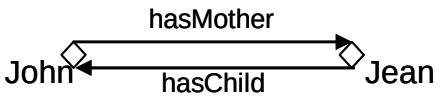


Figure 3.7 An Example of Inverse property: hasChild

1.4.1.2 Properties Characteristics

The capabilities of properties are enhanced through the use of properties characteristics. Four different characteristics are available in OWL for properties.

1. Functional.
2. Inverse Functional.
3. Transitive.
4. Symmetric.

1. Functional Characteristics

The property with this characteristic allows an individual to relate with at most one individual. Functional properties also called as Single Valued properties. For example in Figure 3.8, John is linked with Jean with functional property *hasBirthMother.*

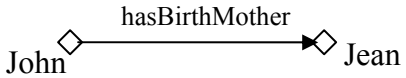


Figure 3.8 An example of Functional Property: hasBirthMother

2. Inverse Functional Characteristics

It is the inverse of Functional characteristic. If this property is given to an individual then at most one individual can relate to this individual via this property. For example in Figure 3.9 *isBirthMotherOf* is the inverse functional property of John, thus only one individual Jean can linked with John via this property.

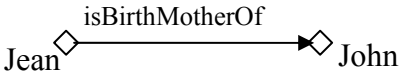


Figure 3.9 An example of Inverse Property: isBirthMotherOf

3. Transitive Characteristics

If the property is transitive and for instance, a is related to b and b is related to c then a is assumed to be related with c through this property. For example in Figure 3.10, If **a** is aligned with **b** and **b** is aligned with **c** then it means that a is also aligned with **c**.

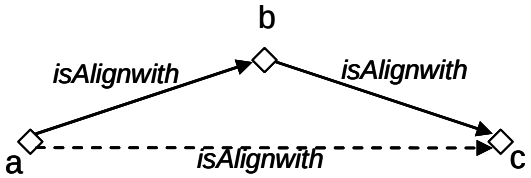


Figure 3.10 An example of Transitive property: isAlignwith

4. Symmetric Characteristics

If individual a is related to **b** via symmetric property then **b** is also related to **a** via this property. For example in Figure 3.11, if *isBrother* is a Symmetric property and John is related to Tom via this property then it is assumed that Tom is also connected to John via this *isBrother* property.

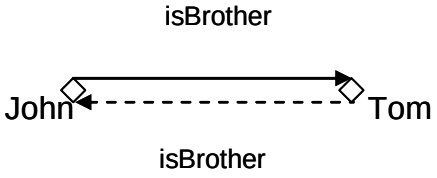


Figure 3.11 An example of Symmetric property: isBrother

1.4.1.3 Property Domains and Ranges

Properties connect individuals from domain to the individuals of range. In the example of Figure 3.12, *hasMother* property connects the individuals of domain class *Child* to the individuals of range class *Mother*. The inverse property *isMotherOf* is doing the inverse of *hasMother* property.

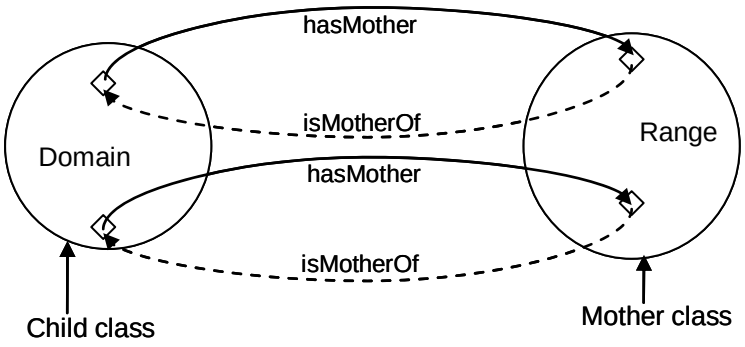


Figure 3.12 An example of Domain and Range of hasMother property

After getting all the ideas together, the model of the ontology can be developed. The starting point is to build the classes, subclasses and the super classes. It is not necessary that all the classes that belong to the whole system have to be built initially; the class can be populated very easily with the current model anytime it

is required. As it was mentioned several times, that the classes are the concept of the objects and sets of the existing objects, the definition of classes is vital to describe the existing individuals of that class. The classes are defined using the properties, so properties are created before defining the class.

The characteristics of properties play an important role in defining the class so it is required to be careful to select them to make the ontology consistent. A guide to build ontology, "*A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools*" is available which is very helpful (Horridge et al., 2004). The conditions, constraints, relations are all described in class, therefore after completion of the description, the structure for the individuals are built up, and can introduce the appropriate individuals to appropriate classes. The overall process might be well understood in Figure 3.13.

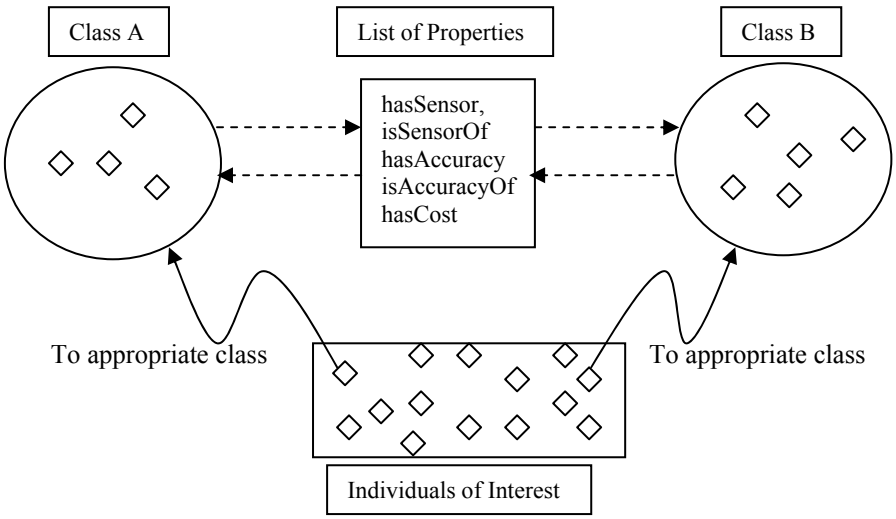


Figure 3.13 Representation of classes, properties and individuals

3.2 The methodology of simple ADCS ontology design

As the author is very new in this field and had no knowledge about design methodology a step by step guide from a good reference about creating first ontology was followed. The author of that guide "*Ontology Development 101:A Guide to creating your First Otology*" (Noy and Mcguinness, 2001) is acknowledged.

To be clear to the reader it is mentioned that a few headlines are also directly taken from this above guide.

3.2.1 Steps of Building an Ontology

Before starting ontology design some basic and important rules need to be considered. There is no single correct way or procedure to design an ontology, there are always possible alternatives and none of them is best or worst. The good solution depends on the types and requirements of the application. The ontology should depend on the practical domain of interest and be close to the object of the domain so that it reflects the objects of that field.

Ontology development is an iterative process. After defining an ontology, it is evaluated and debugged by using it in a practical application or by discussing with the expert of that field or both, so almost every time an ontology needs to be revised and changed and this process is likely to continue throughout of its lifecycle.

The suggested steps for development an ontology by (Noy and Mcguinness, 2001) are:

Step 1: Determine the domain and scope of the Ontology

Before starting an ontology development it is necessary to determine the scope and area or domain of the ontology. Some overall basic questions should be considered:

- What is the ontology about?
- Who is going to use and maintain the ontology?
- What concepts are going to be presented and which type of problem it will solve?
- How should details about the concepts (classes) be considered for the field?
- How reliable should it be?
- What are the risks and concerns that are associated with this?

The answers to these questions help to limit the scope of the ontology. As the objective of the project is to build an ontology which will assist the initial design procedure and selection procedure of ADCS, the representation of the ADCS is the domain of the ontology. In general the concepts and objects of the ADCS subsystem is the domain of this ontology.

As mentioned earlier an ontology is an iterative process and always changeable and upgradeable depending of its domain and scope. Therefore it is also a big question when an ontology is considered as complete? One of the best ways to scale the ontology is by sketching some questions that the ontology should be able to answer called *competency questions* (Gruninger and Fox, 1995).

These following sample competency questions were considered for this project:

- What are the control methods? Where they are appropriate? Which one is suitable for what situation?
- What are the possible attitude determination methods?
- What types of attitude determination hardware are available? What is the accuracy?
- What types of control hardware are required? Can the ontology decide the options?
- What are the disturbance torques for a mission? How to compensate them? What torquers are required to reject them?

By judging the competency question the ontology needs to expand and by answering them the ontology can be defined as being completed.

Step 2: Consider reusing an existing ontology

It is worth considering reusing an ontology that already somebody has done in the same field. Many ontologies already exist in electronic form and Protégé supports import in various formats easily, so it is appreciated to reuse the existing ones rather then building them from scratch if they meet the requirements and scope of the current ontology. It saves valuable time and effort which can be used for expanding the existing one.

For example the ontology of Attitude Control methods may already exist, by importing that to this project not only the classification of Attitude Control Method done but also a large range of properties and vocabulary also included to describe the ADCS system. However the author did not find any relevant ontology in this field so this ontology was designed from the scratch.

Step 3: Specify important vocabulary in the ontology

It is helpful to write down all the important terms that may required to describe the domain. It is difficult to list all terms at the beginning very precisely without overlapping, but no matter if they are complete or not, anytime they can be added during the design time.

For example this ontology contains hundreds of ADCS related terms like Sensors, Actuators, Wheels, Thrusters, Orbit, Principle axis, Disturbance, Torques, Torquers, Gravity gradient , Spin stabilization, Three axis stabilization and so on. Before starting ontology design, it is not only important to decide which concepts are going to be presented, it is also crucial to think why they are presented.

Step 4: Create the Classes hierarchy and define the Classes

There are several possible methods of creating class hierarchy. The *Top down* method is the process to start with the most general term of the class that means the superclass, then class and then the subclass. Obviously all the classes of *Subclass* are the class of *Class* and all the classes of *Class* are the class of *Superclass*. A *Bottom up* method is the reverse of top down method, starts from the subclasses of the hierarchy and grows to the root superclass. The *Combination* is the combination of the top down and bottom up approach. For example, If a class A is the superclass of class B and class B is the superclass of class C, then every individual of C is the individuals of B and every individual of B and C are also individuals of A. So what ever method is followed they present the same vocabulary. None of them is better then other methods, the approach is totally dependent on the personal interest. In this project the author followed the *Combination* method and found it easier. For instance the most general terms in this Figure 3.14, are *Attitude_Control_Hardware_and_Torquers*, *Actuators*, *Booms*, and *Dampers* are the top level terms, and *Dul_gimbal* and *Single_gimbal* are the most specific class in this hierarchy and are bottom level term.

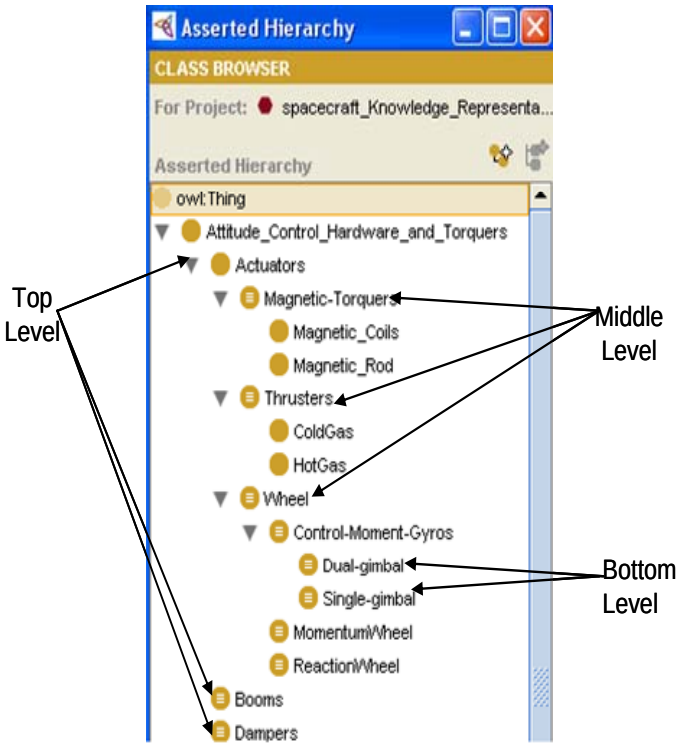


Figure 3.14 A breakdown of classes for different levels.

The Ontology presents the relations of the concepts linguistically expressed through fragments governed by the verbs (Pazienza et al., 2005). Table 3-1

shows how the class *Gravity_Gradient* class can be presented in OWL DL ontology language by some characteristics of this stabilization method.

Table 3-1 Some concepts that define Gravity Gradient stabilization method:

Characteristics	Defination in ontology
In orbit s/c Y axis is align with Pitch	**Y_Axis:** isAlign_with only Pitch
An elongated object in a gravity field tends to align its long axis with gravity gradient. That is maximum moment of inertia align with pitch axis.	**Gravity_Gradient:** hasMaximum_Moment_Of_Inertia_Align_with some (Y_Axis and (hasAxis_Control_by only Gravity_Gradient_Torque))
Booms are used in the long axis to improve the inertia.	Gravity_Gradient: hasInertia_improvement_by some (Booms and (isAlign_with only Pitch))
Sometimes dampers are added to gravity gradient s/c to reduce oscillations around the local vertical vector caused by disturbances.	**Gravity_Gradient:** hasUnwanted_motion some (Oscillation and hasUnwanted_Motion_Damping_by some (Dampers and (isAlign_with only Yaw)))
The orientation of gravity gradient s/c around nadir vector is unconstraint, can be control using momentum wheel	**Gravity_Gradient_and_Momentum_bias_Wheel:** hasMinimum_Moment_Of_Inertia_Align_with some (Z_Axis and (hasAxis_Control_by only MomentumWheel))
This stabilization method is more effective in low and medium altitude orbit	**Gravity_Gradient:** isEffective_Around only (Low_Altitude_orbit or Medium_Altitude_orbit)
Provide very limited manoeuvrability	hasManoeuvrability only Very_Limited
Can provide maximum 5^{o} accuracy	**Gravity_Gradient:** hasMaxAccuracy has 5.0
Provide pointing only towards Local vertical	**Gravity_Gradient:** hasPointing only Local_Vartical

Step 5: Define the properties of the classes

As discussed in Section 3.1.2.1 the classes need to be defined by the means of properties to give the answers of the competency questions. Once the class hierarchy is created the related properties need to be created to describe the classes. Again creating the class hierarchy and defining the properties are closely coupled. It is difficult to do one first and then decide to do another. It is better to create a few classes first, then define the classes by properties (as Table 3-1) and repeat the process.

As it is already mentioned that properties connect the individuals of a class to the individuals of another classes, so it is required to define which classes are going to connect: that is the range of the property, and as well as which classes it describes: that means the domain of the property. For example in Figure 3.15, the

property *isUsed_to_desaturate* connects the individuals of it domain classes for example *HotGas* to the individuals of its range classes *Wheel*.

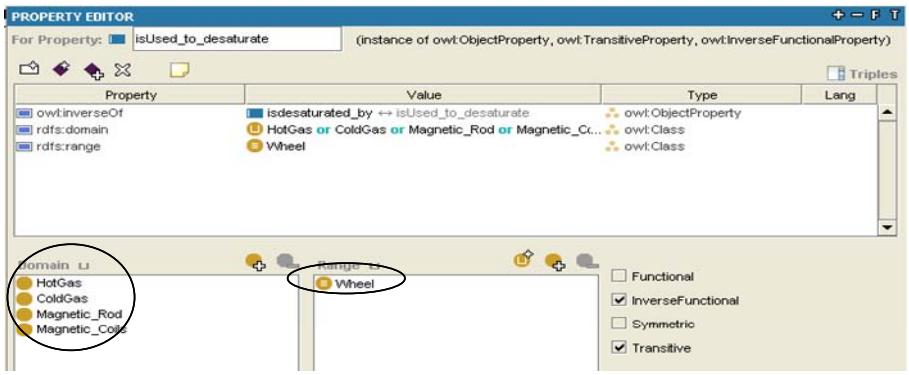


Figure 3.15 The Domain and Range of property: isUsed_to_desaturate

Properties can have different characteristics describing the type of values, allowed values, number of values, features of values that need to be considered. These characteristics are set by the nature of the individuals of domain classes. Again for the example of Figure 3.15 the property *isUsed_to_desaturate* have InverseFunctional and Transitive characteristics. In practice it was considered that a wheel needs one type of desaturation method and by only one device, this idea was implemented by setting InverseFunctional characteristics of *isUsed_to_desaturate.* A list of properties that was consider for ADCS ontology are given below in Figure 3.16

Name	Prefix	Range	Domain	Inverse	Other Characteristics
isAlign_with		Yaw or Roll or Pitch or Z_a...	owl:Thing		Functional, InverseFunctional, Symmetric, Tra
isAltitude_Orbit_of		Common_Input_parameters...	High_Altitude_orbit ⊔ Low_...	hasAltitude_clas:...	
isCaused_by		Disturbance_Torques	owl:Thing		
isCollected_Sensor_data_Store...			Sensors	hasCollection_of_...	
isCommon_terms_of		A_Example_of_Spacecraft...	Common_Input_parameters...	hasCommon_terr...	Functional
isControlled_by		Actuators	Disturbance_Torques	isUsed_to_Control	
isDepend_on		string	owl:Thing		
isDisturbanceOf		Altitude_classifications or I...	External ⊔ Internal	hasDisturbance	
isDominant_Disturbance_of		Low_Altitude_orbit or Medi...	Aerodynamics_Disturbanc...	hasDominant_Dis...	
isEccentric_Orbit_of			Elliptic_orbit ⊔ Hyperbolic_t...		
isEffective_in			owl:Thing		
isInclination_Orbit_of			Polar_orbit ⊔ Prograde_orb...	hasInclination_cl...	
isOrbital_Period_Unit		string	owl:Thing		
isOrbiting_Around		string	owl:Thing		Functional
isOrbitof			Earth ⊔ Moon ⊔ Sun	hasOrbit_Around	InverseFunctional
isPitch_Pointing_towards		Pointing_requirement	owl:Thing		
isPointing		string	owl:Thing		
isPrincipal_axisOf		Common_Input_parameters...	Principal_axis	hasPrincipal_axis	InverseFunctional
isProductOf		string	owl:Thing		
isRange		float	owl:Thing		
isRequiredforStabilisation		string	owl:Thing		
isRoll_Pointing_towards		Pointing_requirement	owl:Thing		
isSensor_of			Sensors	hasAttitude_Dete...	
isSettling_time		duration	owl:Thing		Functional
isSynchronized_Orbit_of			Geosynchronous ⊔ Sun-S'...	hasSynchronizat...	
isUsed_for			owl:Thing		
isUsed_to_Control		Disturbance_Torques	Actuators	isControlled_by	
isUsed_to_Control_axis		Redundant_axis or Yaw or...	Actuators ⊔ Natural_Torqu...	hasAxis_Control_...	
isUsed_to_Damp_Unexpected_...		Precession or Nutation or Y...	Thrusters ⊔ Dampers	hasUnexpected_...	
isUsed_to_Provide		High_rates_Slewing or Non...	owl:Thing		
isUsed_to_control_axis		Redundant_axis or X_axis ...	owl:Thing		Functional

All Properties

Figure 3.16 List of properties for ADCS ontology.

Some rules need to be considered during defining the domain and range of a property.

- If the domain or range of a property contains a list of subclasses and also the superclass of those subclasses then it is required to remove the subclasses.
- If the domain or range of a property contains a list of subclasses but not the superclass of those subclasses then the range should contain only superclass not all subclasses.
- If the domain or range of a property contains all the subclasses but only a few subclasses of a superclass then it should consider making the property generalize for all subclasses and contain only the superclass in the domain and range.

Step 6: Create the individuals

The last step of the ontology design is to introduce the individuals in the classes of the hierarchy by (1) choosing the class (2) creating the individuals of that class and finally (3) fill the properties values. For example, in Figure 3.17, DSS-256 is a individual of Digital_Sun_Sensor class. The following properties are defined for DSS-256:

hasMaxAccuracy	: 0.3 deg
hasWeight	: 0.12 kg
hasOutputSignal	: Digital
hasCost	: 30 pound
hasAxis_Pointing	: X_axis_of_sc_body(instance of *X_axis* class); Y_axis_of_sc_body(instance of *Y_axis* class)
hasPower	: 0.2
isUsed_to_determine	: Sun Angle
isProductOf	: Space Innovations Limited
isSensor_Of	: FireSat_Mission (instance of *Space_Mission* class)

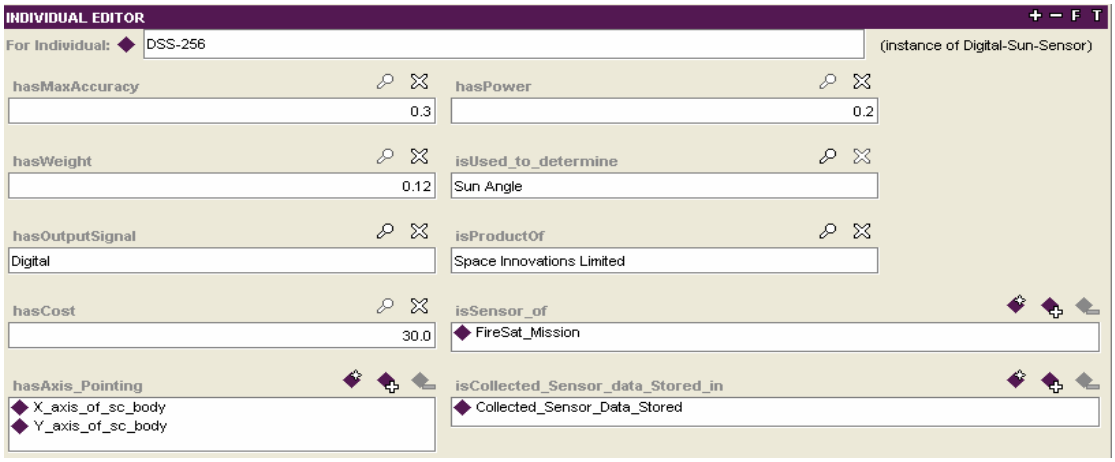


Figure 3.17 Representation of Individual DSS-256

3.2.2 Consideration during ontology design

The Next section is going to present the simple aspects that need to be considered during the design to avoid common mistakes. Thus there is no standard method of design and no one is the best but still it is better to follow some fixed conversion throughout the lifetime of the ontology to avoid ambiguity. Some useful tips from (Noy and Mcguinness, 2001) are given below:

3.2.2.1 Try to make sure that the class hierarchy is correct

The class hierarchy should maintain an "is-a" relation. Class A is a superclass of B if and only if all the individuals of class B also the individuals of class A. It is easy to determine by asking this "is-a" question. For example a sun sensor is-a sensor, so *Sensor* class is the superclass of the *Sun_Sensor* class.

A class is a set of objects, for example *Sun_Sensor* class is a class of all sun sensors, so some designer prefers to name the class plural like *Sun_Sensors*. No method is good or bad but it is required to be attached with one convention for all the time, so that it prevents the mistakes of designer as well as being less confusing for the users.

"Classes represent concepts in the Domain and not the words that denote these concepts " (Noy and Mcguinness, 2001), is an important clause for designing the classes. The name of the class can be changed anytime by its synonyms but the term itself should reflect the object of the domain. Synonyms of a class name should not represent different classes. For example, Moment of inertia, Mass moment of inertia or Angular mass are all terms that present the same concept, so the ontology should not have both *Moment_Of_InertiaI* class and *Angular_Mass* class. It can be called either *Moment_Of_InertiaI* or *Angular_Mass* better to try to generalize the name.

It is also crucial to avoid class cycles in the hierarchy. If class A is the superclass of class B and also class B is the superclass of A, then class A and B become equivalent. By making this cycle it means that all the individuals of class A are the individuals of class B and all the individuals of class B are the individuals of class A, though it is not the concept of subclass and superclass.

3.2.2.2 Analysing the siblings in a class hierarchy

The siblings are the direct subclasses of a class and should have the same level of generality. Example in Figure 3.18 shows that *Earth-Sensor*, *Magnetometer*, *Star-Sensor*, and *Sun-Sensor* are all siblings and they belongs to class *Reference-Sensor*.

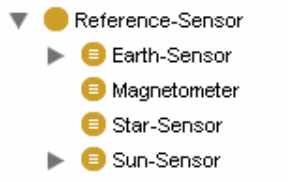


Figure 3.18 Represents the Siblings hierarchy

For example Digital sun sensors and Sun sensors should not be the subclass of same class because Digital sun sensors is the more specific term then Sun sensor whereas Earth sensor, Sun sensor, Star sensor, and Magnetometer are all the classification of reference sensors and present the same level of concept that means they fall in a same line. Before doing any classification one should bear in mind that the sibling classes hierarchy present same level of concepts but not the same concept.

Though there is no strict rule for the number of subclasses that a class may have, a well structured ontology should contain two to twelve subclasses. It is assumed that if a class contains only one subclass then may be there is some modeling problem or conceptual problem. In the same way if there are more then twelve subclasses in a class then further division of the subclasses may necessary. However if natural classification does not exist in the real world, then there is no need to create an artificial classification and a long list of siblings can be acceptable. Essentially an ontology should reflect the real world.

3.2.2.3 Multiple Inheritance

It is common in most of the ontologies that a class can be a subclass of multiple classes. For example Gravity gradient force can be considered as disturbance as well as the necessary torque for the Gravity gradient stabilization. So the class *Gravity_Gradient_Torque* is the subclass for both *External_Disturbace* class and *Passive_Stabilization* class.

3.2.2.4 When should a class be introduced (or not)?

An important decision for a good modelling is when a class should introduced. A few rules can be followed during the introducing of a new class in the class hierarchy.

> "*Subclasses should have (1) additional properties that superclasses do not have.(2) restrictions that are different from the superclasses(3) participate in different relationships than the superclasses.*(Noy and Mcguinness, 2001)"

In practice a subclass should have at least one added feature that the superclasses do not have but sometimes it is also necessary to build the tree of subclasses without adding new properties. For example, a medical system model may contains a list of classification of various diseases without properties or with same properties. This type of class hierarchy also exists because they are useful because of (1) easy exploration and navigation and (2) allow choosing the level of generality of the term which is appropriate for the situation.

3.2.2.5 A new class or property value?

During modelling a domain it is often required to decide whether a concept should be a property or a class. For example, an Earth's orbit can have different height like Low, Medium and High, so should *Low Earth Orbit* be a class or simply create an *Orbit* class and fill the *Height* property value as Low? This decision depends on what importance the *Low Earth Orbit* class may have and what are the particular implications that it may have in the domain. If a property has values with different concepts and these concepts become the restrictions for the different properties of other classes then that property should convert to a class.

3.2.2.6 An individual or a class?

An individual is the most specific knowledge of the domain or the exact representation of the object itself. For example Sensor is the collective knowledge of all sensors that exist in the domain, but DSS-256 is a specific sensor from that domain, so DSS-256 is an individual. If it is not possible to further divide the concepts then the concept is an individual or, conversely, if the concept may form a hierarchy then it is required to introduce a class for that concept.

3.2.2.7 Limiting the scope

The crucial question is when can the definition of the ontology be considered as complete? Some basic rules need to be considered to take the decision.

- The ontology can not contain all the information about the domain.
- It is not required to generalize or specialize the ontology more than the application need.

For example the knowledge and the classification of sensors are required but the design and electronics of the sensors are not the interest for this ontology. Therefore it is not required to introduce the information about the design and

electronics. May be the ontology about Sensor itself will consider these information. Similarly

- An ontology can not contain all the possible properties of a specific concepts and distinctions among the classes.

It is not possible to include all the possible properties that all the classes of ADCS ontology can have. Most relevant information that is usually practiced during initial ADCS design is represented here. As mentioned before, ontology development is a continuous process and requires continuous upgrading throughout its lifetime. Therefore new properties are likely to be added during the design process.

3.2.3 Naming Convention

Though there is no standard or restriction for naming an ontology, it is better to define and always stick to that definition so that the ontology is easy to understand. This helps to avoid mistakes during design. Several features of Protégé OWL were considered for choosing the naming convention:

1. Protégé does not support same name for classes, individuals and properties. For example it is not allowed to name a class *Sensor* and an individual *Sensor*.

2. Protégé is case sensitive. Protégé can not treat the name that differ in case, such as if a class *Sensor*(initial uppercase) and a class *sensor*(lowercase) exist, then Protégé will treat them as different class whereas may be they are presenting the same concept.

3. Protégé does not support most of the delimiters but only underscore and dash.

3.2.3.1 Capitalization and delimiters

The methods of capitalization and the use of delimiters are described in this section. To improve the readability, it is common to capitalize the class name and use the lower case for the properties name. For example when a class name contains more then one word like Attitude control system then the name should delimit the words. The several choices are:

- The words can be together and capitalize each new word like *AttitudeControlSystem*.

- Use a delimiter between the words, but then it is also needs to be decided whether each word is capitalized or not, for example *Attitude-Control-*

System, *Attitude-control-system*, *Attitude_Control_System* or *Attitude_control_system*.

3.2.3.2 Singular or Plural

It is common in most of the ontologies that they suggest to use prefix and suffix in the properties name to distinguish them from class name though it is not a rule or mandatory for correct ontology. The method that is followed in this project to add a *has* or *is* as prefix and *of* as suffix in the properties name, thus the properties names are like *hasAccuracy*, *isProducerof*. By these conventions it is easy to say whether a name is a class or property.

3.2.3.3 Other naming considerations

A few more things that also need to be considered during naming are as follows: It is usually a good idea to avoid the abbreviations to make the name clear to others. For example the name *Attitude Determination and Control* is more appreciable then *ADCS*.

Name of subclasses should either include or not include the name of the direct superclass. If the name of the superclass is included then all the subclasses should follow the same rule. For example, if two subclasses of *Sensor* class are to present analog or digital sensors, the two subclasses names should be either *Analog_Sensor* and *Digital_Sensor* or *Analog* and *Digital*, but not *Analog Sensor* or *Digital*.

3.3 Consistency checking using RACER

OWL DL language is based on first order language and provides computational completeness. RACER is used to check the consistency of ontology and the descriptions to ensure that there is no logical error, and also automatically compute the class hierarchy of OWL DL. For instance to be a parent it is a necessary and sufficient condition to have a child, so any individual with a property *haschild* will automatically be an individual of *Parent* class. Again If a necessary and sufficient imposed “Cows are vegetarian and do not eat meat” and if a person “Fred is vegetarian” also will be inferred as an individual of class *Cow* if *Cow* and *Person* class are not disjoint. Another example, when creating ontology about farm animals, “ one might say the following: (1) cows and sheep are animals; (2) cows are vegetarians; (3) vegetarians eat no animals, and eat no parts of any animal; (4) mad cows eat the brains of sheep; and (5) a mad cow is a kind of cow. However, there is an inconsistency lurking: a mad cow cannot be a kind of cow because that means it is a vegetarian that eats sheep brains.”

(Provine et al., 2004). Thus the consistencies of an ontology establish the logical completeness and increase the confidence of conflict avoidance among the logics.

For a large ontology without a reasoner it is extremely difficult to keep the ontology maintainable and logically correct. For example, natural torques like gravity gradient torque, magnetic torque are the disturbances for some stabilization methods and others may be use these torques to stabilize the spacecraft. So these torques would be under both *Disturbance_Torques* and *Natural_Torquers* superclasses. The best of way of doing it is to make a simple class tree without making multiple superclasses. The reasoner computes the multiple inheritance classes (Horridge et al., 2004) and minimizes the human error.

3.4 Why use Excel?

The steps of Attitude determination and control system design will be presented in the next chapter. During the design it is required to model the disturbances that are going to affect the orientation of the spacecraft. Then this knowledge is used to determine the size and authority of the actuators to reject that amount of disturbance to keep the spacecraft in the correct position. An ontology represents the knowledge of the domain but does not provide the support of doing mathematical calculation. Therefore it is required to gather the knowledge from the ontology and export this knowledge to another application which supports the mathematical modelling.

For this project the data from the ontology was exported for two reasons

1. To model and calculate the disturbances.
2. To estimate the sizes and capacities of the actuators.

A large amount of processed data is also sometimes required to be imported into the ontology. For example, there are hundreds of sensors available from different companies and have different parameters. Storing all the information of sensors into the ontology manually is not trivial task; therefore it is desirable to do the process automatically by some sort of importing facility.

3.4.1 Why was Excel chosen?

The spreadsheet program Microsoft Excel was chosen for the mathematical computations for a few reasons:

- There are existing plugins which support export of the Protégé knowledge bases to spreadsheets.

- Excel is very easy for calculation as well as for programming, and is available on almost every desktop PC.
- Finally, the processed results or tables can be also imported into the Protégé environment using some existing plugins.

3.4.2 How does it work?

Figure 3.19 shows the dataflow between the Excel and Protégé environments. Exporting data from Protege to Excel is straight forward. As discussed in section 2.3.3, Queries Tab plugin supports exporting the individuals from the ontology to Excel in CSV format, then the data ready to do required calculations.

But importing data from Excel to the ontology is not as simple. As the DataMaster plugin allows importing data from a relational database to the ontology, the processed Excel data is saved in a table in MySQL database, then by using the DataMaster plugin the content of the table from MySQL is directly shifted to the ontology (as discussed in Section 2.3.4.3 and Section 4.1.4.2).

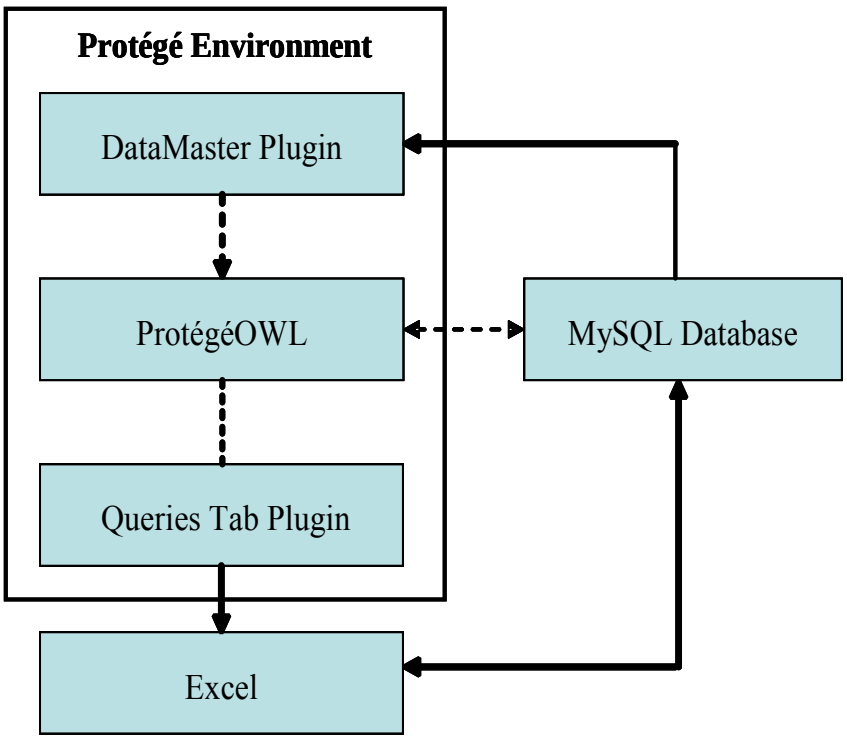


Figure 3.19 Dataflow between ProtégéOWL and Excel.

3.5 Why using Matlab

Dr. Hardacre, a renowned ADCS engineer mentioned "You learn by doing, if you want to get a feel for it, simulate it. Visualize it- Matlab is your friend." (Hardacre 2008). Most of the simulation and visualization tasks for ADCS design are now done in Matlab. One of the objectives of this project was to show the possible way of using the knowledge of the ontology in simulation of ADCS the system in Matlab.

The same procedure of Section 3.4.2 has been followed to export the data from the ontology using the Queries Tab, but now the exported file is saved in Matlab workspace for direct using. Figure 3.20 shows the dataflow diagram of the procedure.

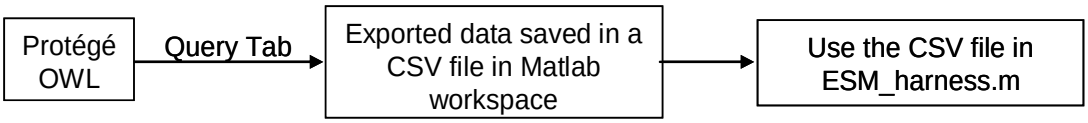


Figure 3.20 Dataflow from Protégé Environment to Matlab

To build a complete knowledge based model of ADCS design, the ontology is act as the central repository of data. The data is shared by other supporting software tools. In this project, the knowledge sharing between ontology and other engineering tools, Excel and Matlab are illustrated. Next chapter is going to show the practical implementation of ADCS design using ontology.

Chapter 4 Results

After building the ontology the question arises, how it is going to contribute in ADCS system design? Often the goal of developing an ontology is to define a set of data and their structures for other program to use. Software agents, problem solving methods, and domain independent applications use the ontologies and the knowledge of the ontologies as data for their applications (Noy and Mcguinness, 2001).

Building an ontology of ADCS domain is not the only goal of this project but the aim of the project is to explore, how the knowledge of the ontology can be used in practical problem for ADCS system design. Since this ontology contains all the information of the ADCS domain and ready to provide all the available knowledge of the ADCS domain, this chapter presents how the ontology is going to be used in practical ADCS system design.

4.1 The steps of ADCS design process

The typical step by step procedures that are followed in ADCS design is given below (Wertz and Larson, 1999):

1. Define the control modes and determine the control requirements by the control modes.
2. Select the attitude control method determined by control modes.
3. Model the disturbances.
4. Select and calculate the size of ADCS hardware.
5. Define determination and control algorithms.
6. Iterate the process and documentation.

How ontology helps a person to follow exact steps of the design procedure will be presented in next sections, though the person might not be an expert of this field. Thus ontology shares the understanding among the people who are not even expert on that field.

The simple example of FireSat spacecraft from “Space Mission Analysis and Design” (Wertz and Larson, 1999) was chosen to show the design steps of ADCS using the designed ontology. Table 4-1 shows the mission requirements and input parameters of hypothetical FireSat mission.

Table 4-1 Input Parameters and requirements for FireSat Spacecraft

Input Parameters	Parameters	Values
	Orbit Altitude	800km, Circular
	Mass	215 kg
	I_x	90 kg.m^2
	I_y	90 kg.m^2
	I_z	60 kg.m^2
Mission Requirements	Lifetimes	5 years
	Slew rate	$< 0.1^o$/s
	Pointing towards	Earth (nadir)
	Pointing accuracy	0.1^o
	Optional requirement	30^o in 10 min time manoeuvre per month to the target of opportunity

4.1.1 Control modes and requirements of the control modes

Typically during different modes of the mission for example during orbit insertion, acquisition, on station, slewing, safe mode and special situation, the attitude of the vehicle need to be controlled. The details about these modes can be found in (Wertz and Larson, 1999).

For simplicity it is assumed that the launch vehicle placed the spacecraft in its final orbit so during orbit insertion no control is required. For this specific mission the spacecraft is Earth pointing and required to maintain 0.1^o pointing accuracy during normal mode, that means when the spacecraft is on station. These requirements mostly drive the system design because spacecraft spend most of the mission lifetime in normal mode or on-station. An optional requirement for this mission is to slew the vehicle once every 30 days up to 30^o within 10 min and need to keep this orientation for 90 min. Nevertheless it is always safe to think about the Special and Safe mode to avoid any unexpected failure, but the Safe mode is only considered in this discussion and will be discussed in Section 4.1.5.

The function of the ontology starts by creating an individual named *FireSat* of class *Spacecraft_Mission* as showed in Figure 4.1. As the subsystem Attitude determination and control (ADCS) is the concern of this project, the property field of *hasAttitude_Determination_and_Control_Subsystem* in editor A is filled by creating an individual *ADCS_for_FireSat_Mission*. Then the editor B for individual *ADCS_for_FireSat_Mission* will appear as Shown in Figure 4.1. As already it is decided that the On-station (normal) and Slew mode are considered to dominate control for this project, now the modes can be added by choosing create instance button on the *hasControl_Modes* frame and the selection options will appear in another frame as Figure 4.2.

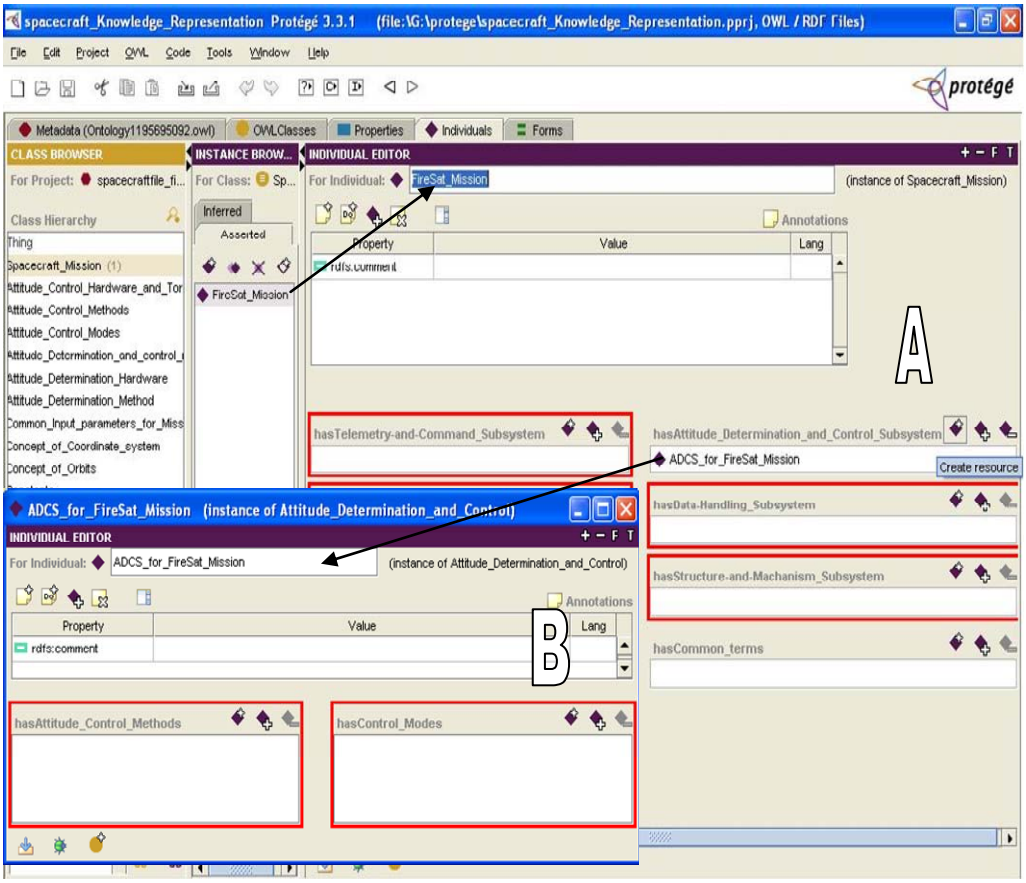


Figure 4.1 Individual FireSat and ADCS_for_FireSat_Mission

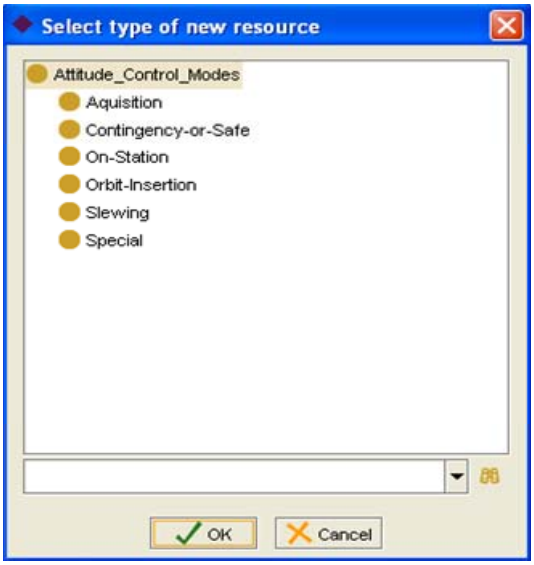


Figure 4.2 The options for Attitude_Control_Modes class.

After choosing the options from the Figure 4.2, the editor B in Figure 4.1 will show like Figure 4.3 with selected modes. The individual *On_Station_Mode_for_FireSat* and *Slewing_mode_for_FireSat* are the instance

of *On_Station* and *Slewing* classes respectively. The individual's names are renamed like this because *On_station* or *Slewing* class can contains a set of individuals which may be from different missions, it is better to include the mission name with the individuals.

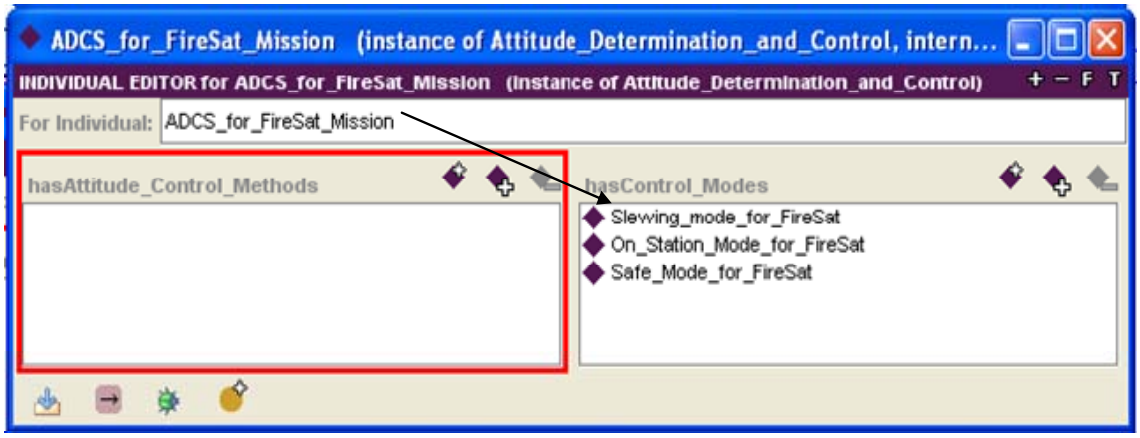
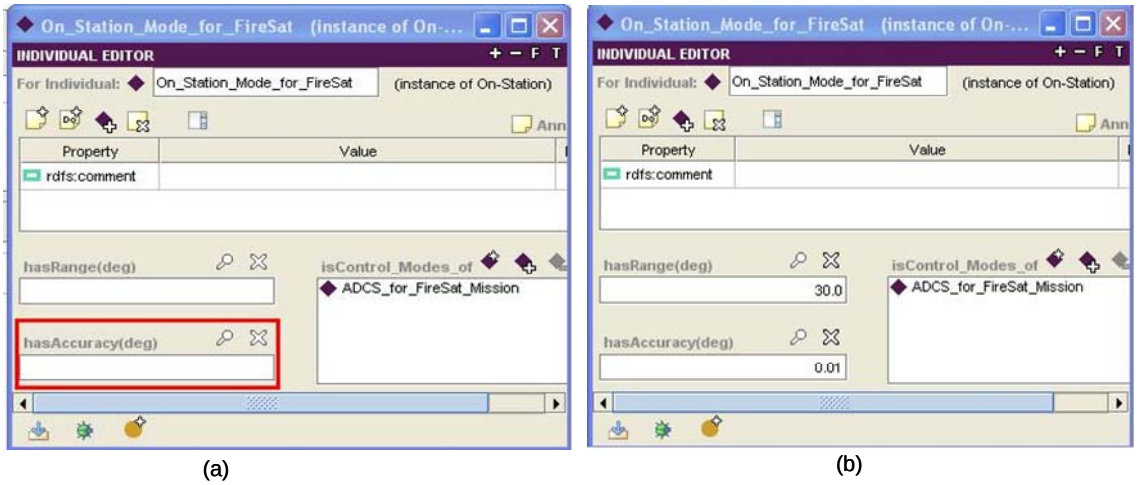


Figure 4.3 The control modes for individual ADCS_for_FirSat_Mission.

When the spacecraft is in Normal or On-station mode then it needs to keep 0.01 deg of accuracy with range of 30° of Nadir, therefore the individual *On_Station_Mode_for_FireSat* needs to be defined by these conditions. As Figure 4.4(a) shows, has Accuracy, is a mandatory field (red outline) and the *hasRange* are the properties of *On_Station_Mode_for_FireSat* before assigning the value and Figure 4.4(b) shows the form with assigned value.



(a) (b)

Figure 4.4 Individual On_Station_for_FireSat

In the same way the individual *Slewing_mode_for_FireSat* is defined with the mission requirements of slew rate 30° in 10 minutes of time or 0.05° per second.

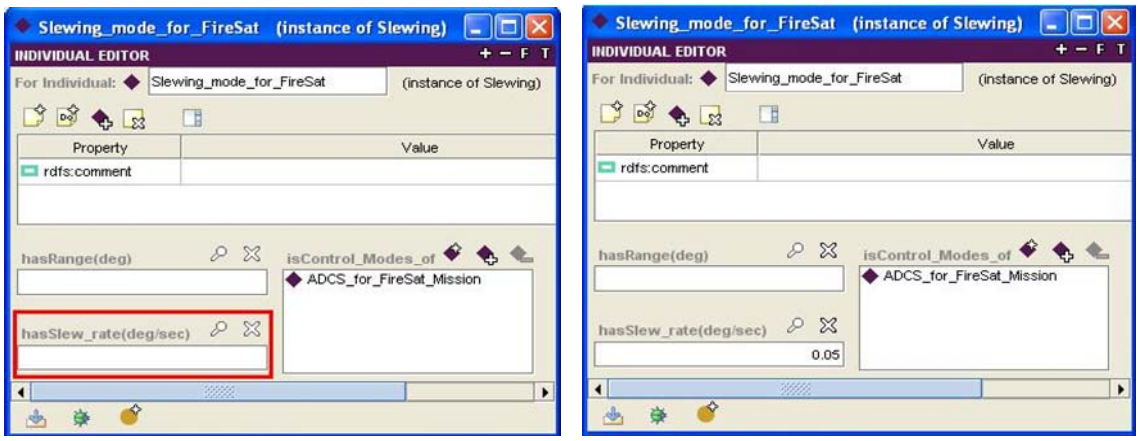


Figure 4.5 Individual Slewing_mode_for_FireSat.

In both Figure 4.4 and Figure 4.5, there is another property *isControl_Modes_of* is automatically filled by *ADCS_for_Firesat_Mission* as it is the inverse property (described in Section 1.4.1.1) of *hasControl_Modes* property by which it is easily recognisable that these control modes are connected with individual *ADCS_for_Firesat_Mission*

As it is already stated that the frame with red border means it is a mandatory field, before assigning the values it should be considered that if the frame is for an *objecttype* property then there is cardinality restriction, and if it is for the *datatype* property then there is data type restriction.

4.1.2 Select the Attitude Control Methods

The selection of the Attitude control methods depend upon the requirements and the different modes that the vehicle can have during the mission. All the possible options of the control methods are presented in the ontology. Therefore it is required to explore the ontology for the methods that can meet the requirements. It can be done by two ways:

1. By analysing the definition of every subclass of *Attitude_Control_Methods* class.
2. Using Query Tab to search.

Exploring each class and going through the definition may be useful to know about the class but not desirable during design selection. Most preferably and easily it is possible to find the desired options using Query Tab. Before going to details about searching and selection of attitude control methods, it is required to consider again that the classes are the sets of individuals, the properties are the characteristics of the individuals. The individuals are only the existing things of the domain.

Thus only individuals are possible to search and the Query Tab searches for the individuals. Therefore some *anonymous individuals* of subclasses of *Attitude_Control_Methods* class are already created to represent the classes. For example, Figure 4.6 shows some searched results which are the *anonymous* individuals of different classes. These classes satisfy the queries, actually do not belong to any mission.

The vehicle of FireSat mission needs to maintain 0.1^{o} of accuracy and frequent orientation with a slew rate greater than 0.1^{o}/s is also required. By querying with these two major requirements, the control methods will be chosen.

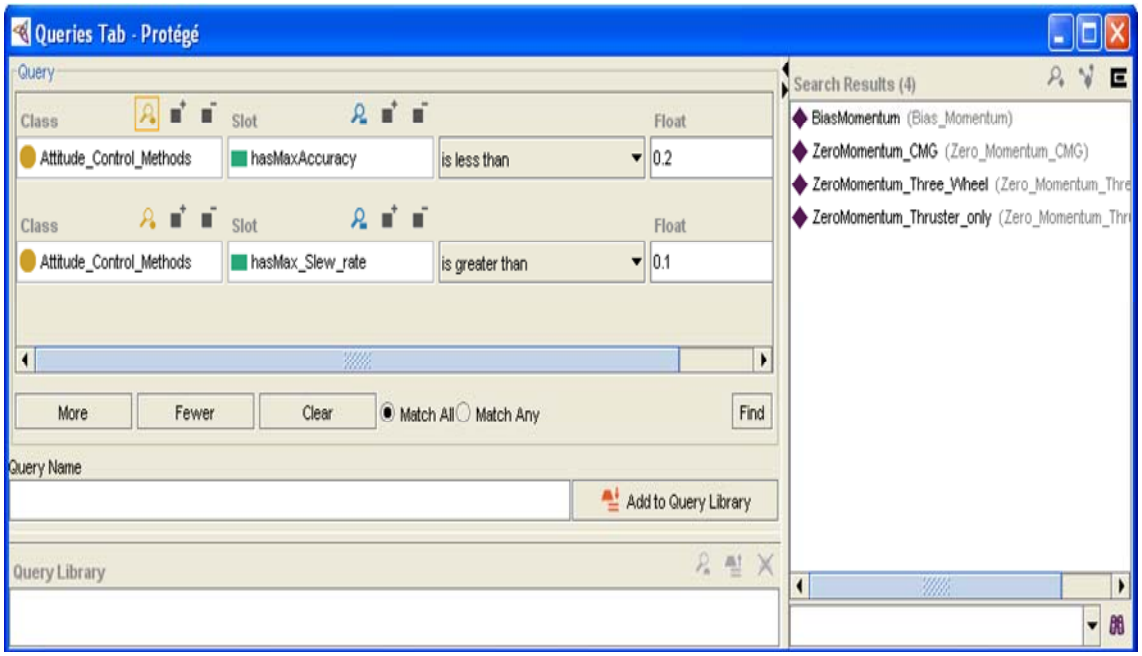


Figure 4.6 Suitable control methods for FireSat.

Figure 4.6 showing the result of the searching, here the value of *hasMaxAccuracy* is chosen less than 0.2, because Query Tab only support equal, greater and smaller checking.

These possible options are now required to be input in *hasAttitude_Control_Methods* properties field for individual *ADCS_for_FirSat_Mission* in the Figure 4.3. Figure 4.7 shows the possible control methods that can be considered for FireSat mission.

The idea of choosing several control methods for controlling the attitude is to analyse all the possible options and then choose the best one suited for the mission.

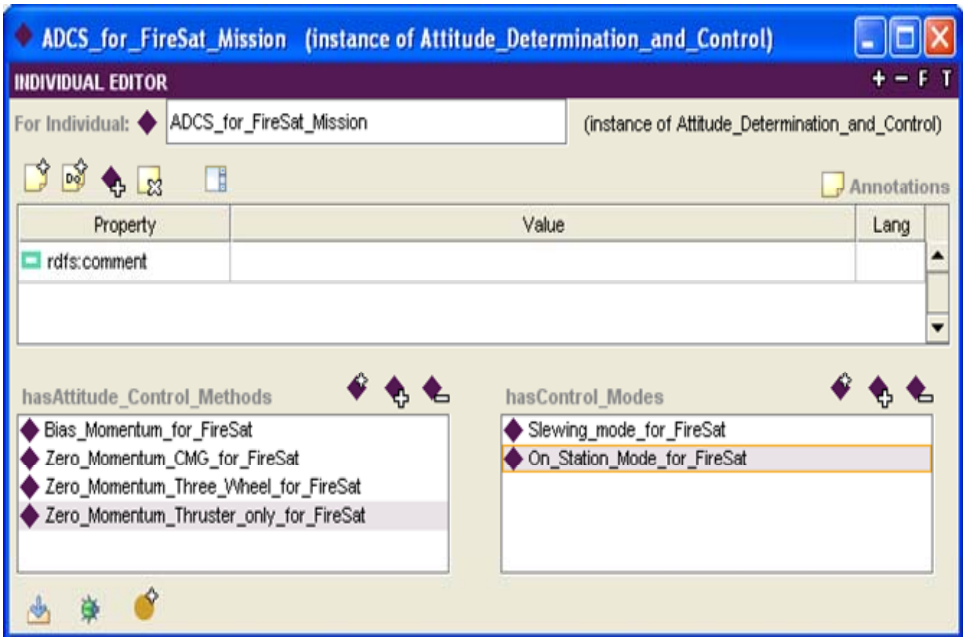


Figure 4.7 The control methods for individual ADCS_for_FireSat_Mission.

4.1.3 Model the disturbances

At this stage it is required to calculate the external disturbances that are going to affect the orientation of the spacecraft. Most of the disturbances are affected by the spacecraft orientation, design symmetry and the mass properties of the spacecraft (Wertz and Larson, 1999).

Since the ontology is not full and only the ADCS subsystem is designed, some parameters like mass properties, and spacecraft surface reflectivity is related to other subsystem and are not available to the ontology; therefore to make those available a class *Common_Input_parameters_for_Mission* has been created in the ontology to store input parameters that are needed to calculate external disturbances, hence the individual *Common_Input_parameters_for_FireSat_Mission* of that class is presenting all the parameters for FireSat mission (Figure 4.8)

As discussed in Section 3.4, the parameters need to be exported to Excel to calculate the disturbances. Query tab is used for querying individuals in the ontology and exporting data from the ontology by choosing all the slots/properties of individual *Common_Input_parameters_for_FireSat_Mission* as shown in Figure 4.10.

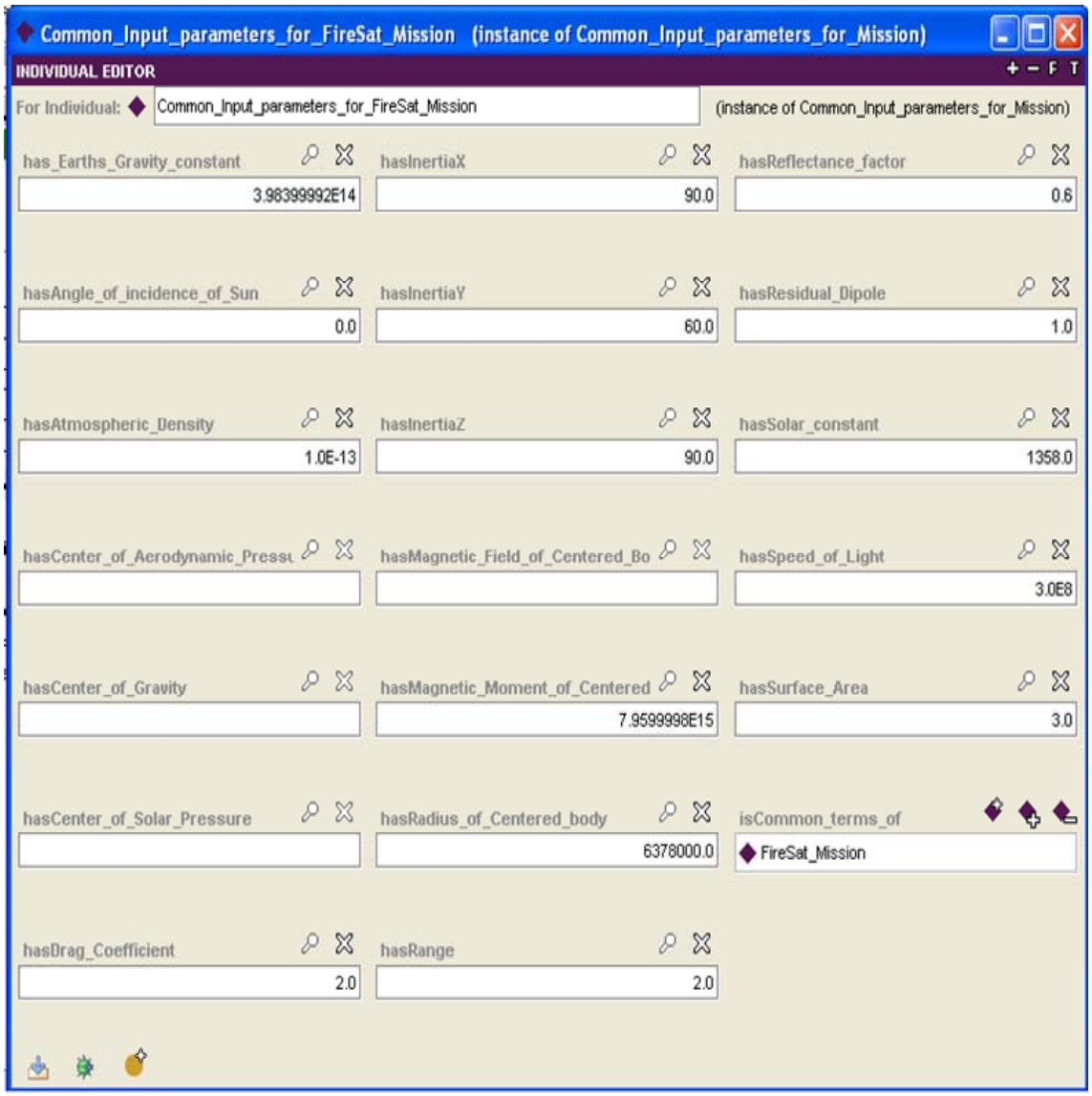


Figure 4.8 The individual Common_Input_parameters_for_FireSat_Mission

First two rows of Figure 4.11 shows the direct imported data to Excel from Protégé. Then with the macro programming in Excel the rows are organized in columns and the disturbance torques for FireSat Mission are calculated.

Four major types of environmental disturbances are calculated using simplified equations from (Wertz and Larson, 1999). The largest torque for this mission is $4.3046 * 10^{-5}$N.m due to the magnetic torque. Now the values of the disturbance torques can be imported into the ontology.

For example the value of magnetic torque from Figure 4.11 can be inputted in the mandatory red bordered field *hasMagnetic_Torque* in Figure 4.9.

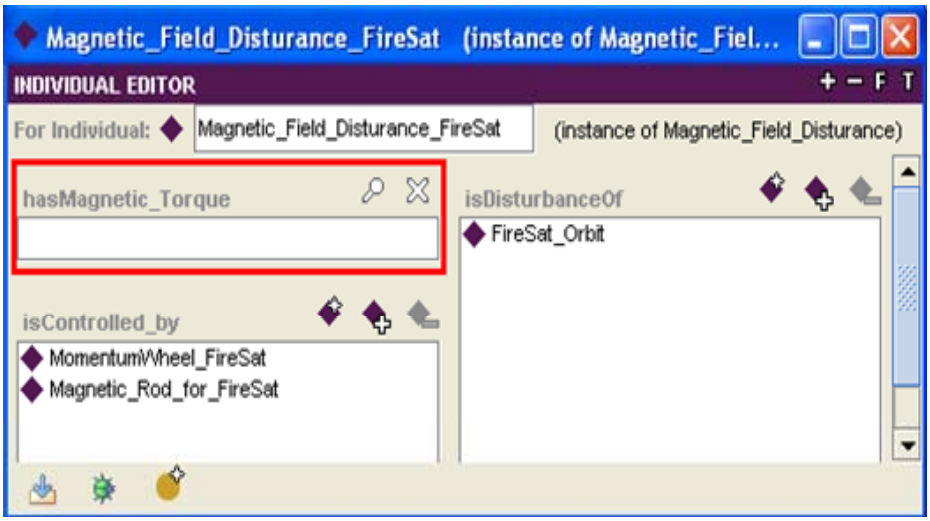


Figure 4.9 The individual Magnetic_Field_Disturbance_FireSat

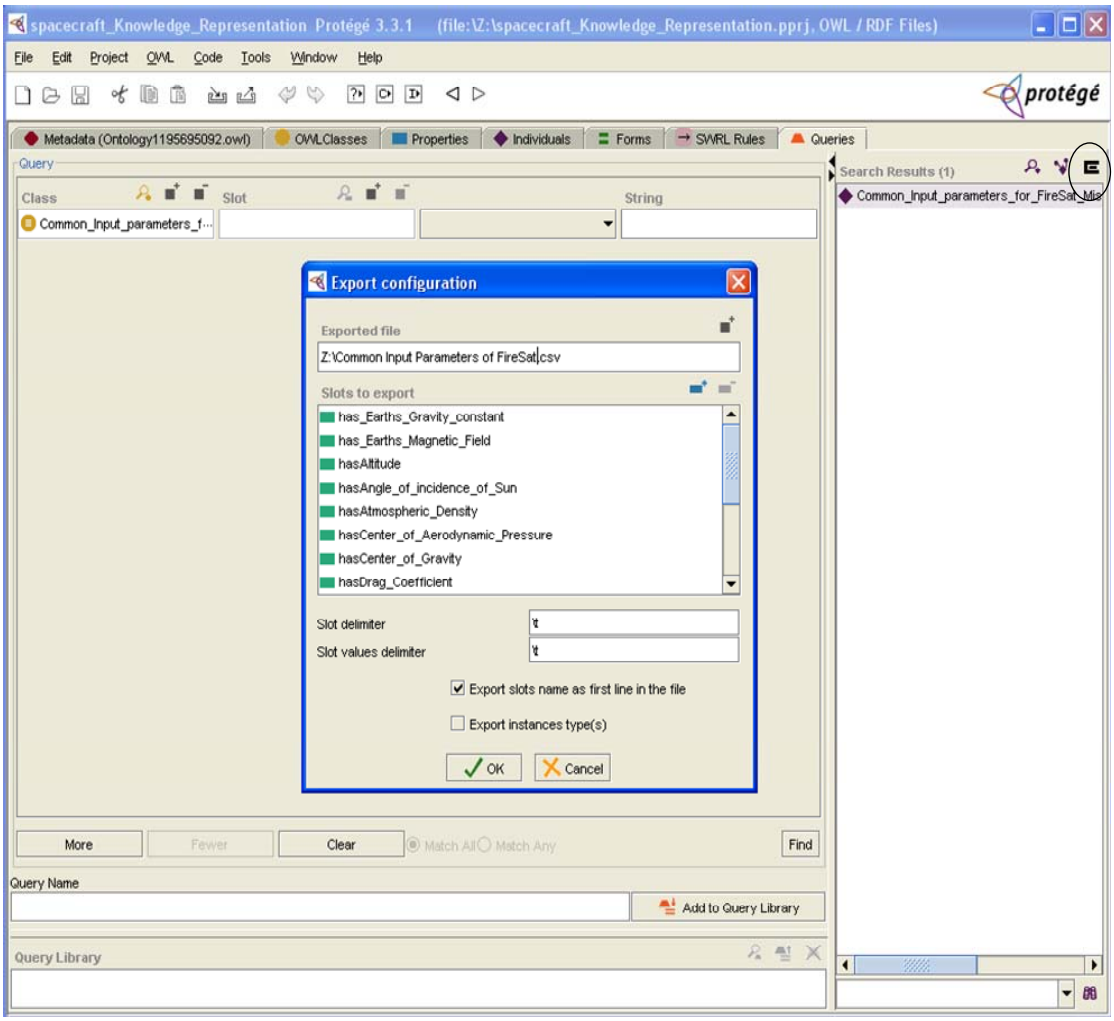
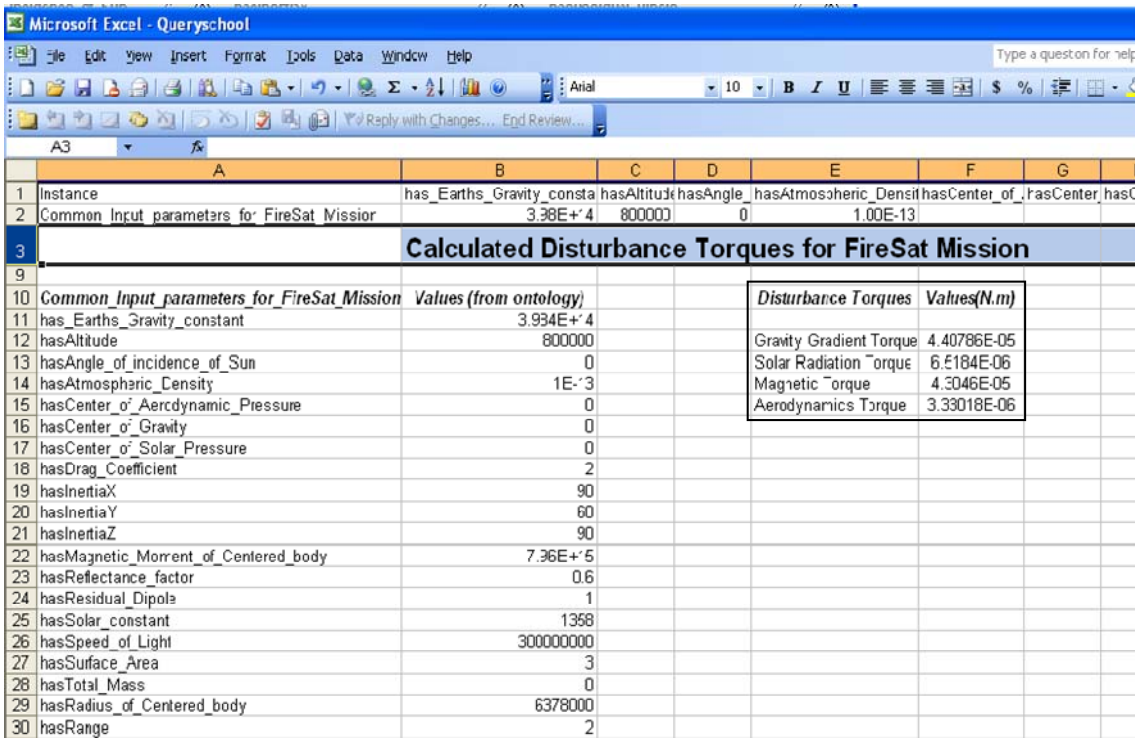


Figure 4.10 Export configuration of individual *Common_Input_parameters_for_FireSat_Mission*



Microsoft Excel - Queryschool

	A	B	C	D	E	F	G
1	Instance	has_Earths_Gravity_consta	hasAltitud	hasAngle_	hasAtmospheric_Densit	hasCenter_of_	hasCenter
2	Common_Input_parameters_for_FireSat_Missior	3.98E+14	800000	0	1.00E-13		
3		**Calculated Disturbance Torques for FireSat Mission**					
9							
10	*Common_Input_parameters_for_FireSat_Mission*	*Values (from ontology)*					
11	has_Earths_Gravity_constant	3.934E+14					
12	hasAltitude	800000					
13	hasAngle_of_incidence_of_Sun	0					
14	hasAtmospheric_Censity	1E-13					
15	hasCenter_of_Aerodynamic_Pressure	0					
16	hasCenter_of_Gravity	0					
17	hasCenter_of_Solar_Pressure	0					
18	hasDrag_Coefficient	2					
19	hasInertiaX	90					
20	hasInertiaY	60					
21	hasInertiaZ	90					
22	hasMagnetic_Morrent_of_Centered_body	7.96E+15					
23	hasReflectance_factor	0.6					
24	hasResidual_Dipole	1					
25	hasSolar_constant	1358					
26	hasSpeed_of_Light	300000000					
27	hasSurface_Area	3					
28	hasTotal_Mass	0					
29	hasRadius_of_Centered_body	6378000					
30	hasRange	2					

Disturbance Torques	*Values(N.m)*
Gravity Gradient Torque	4.40786E-05
Solar Radiation Torque	6.5184E-06
Magnetic Torque	4.3046E-05
Aerodynamics Torque	3.33018E-06

Figure 4.11 Exported and calculated results of disturbance torques for FireSat mission

4.1.4 Select and calculate the size of ADCS hardware

Selections of the control hardware depend upon the control methods of the mission. Among the four options of control methods that was chosen in Section 4.1.2 (Figure 4.7), *Bias_Momentum_for_FireSat* will be discussed in this chapter.

4.1.4.1 Select and size control Hardware

In momentum biased spacecraft, constant angular momentum is provided by a momentum wheel mounted along the pitch axis (Wertz and Larson, 1999). A Momentum biased spacecraft is three axis controlled as follows: the momentum wheel provide inertial stability to the mounted axis which is perpendicular to the orbital plane and torque capability of the wheel gives gyroscopic stiffness to other two axes (Sidi, 2000). As momentum biased satellites have a spinning body then there are some unwanted motions which need to be controlled periodically and the pitch wheel must be desaturated. Usually the momentum wheel is not sufficient to control the vehicle completely, therefore it is common to use additional controller like small reaction wheel, thrusters and magnetic torquers.

First of all the orientation of the spacecraft with respect to orbit is determined. The Figure 4.12 shows the fields that need to be filled. For this mission, the maximum moment of inertia of the satellite is aligned with Z axis, the minimum moment of inertia is aligned with Y axis and the intermediate moment of inertia obviously aligned with X axis (from Table 4-1).

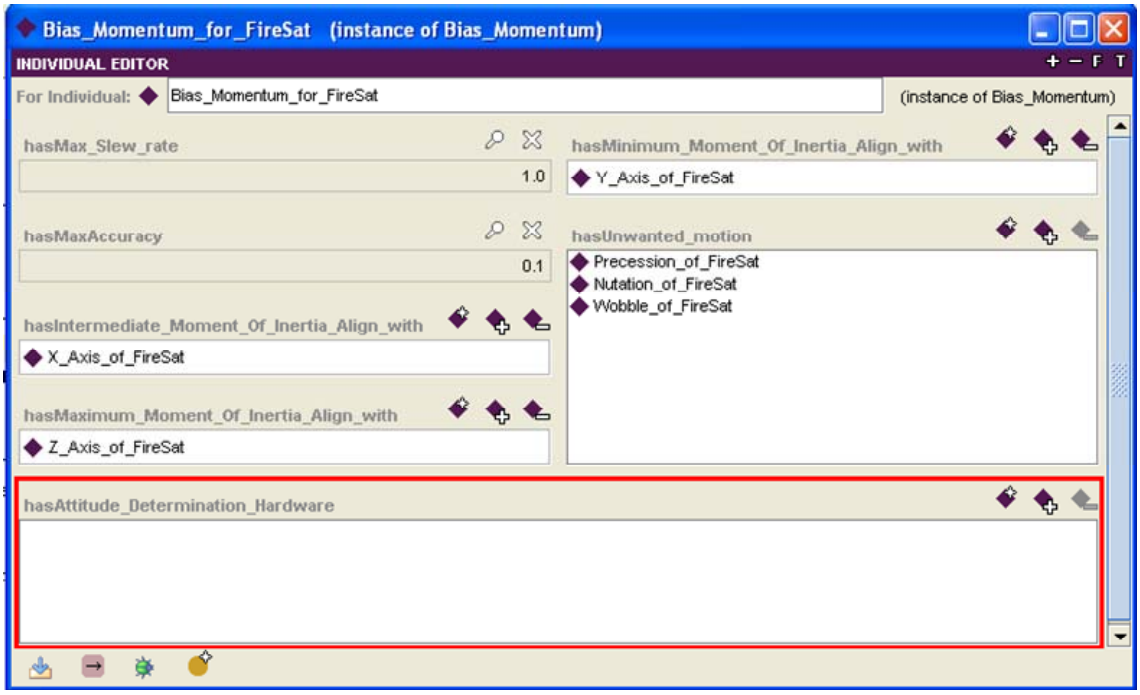


Figure 4.12 The editor of control method Bias_Momentum_for_FireSat.

Obviously X axis can align with only with Roll of an Earth orbiting satellite, so the option for the property *isAlign_with* for individual *X_axis_of_FireSat* is only X_axis will be automatically chosen by the definition of the ontology. The momentum wheel is chosen to control the X axis, and X is aligned with Roll as shown in Figure 4.13 Individual X_Axis_of_FireSat and in the same way Y and Z axes are also defined.

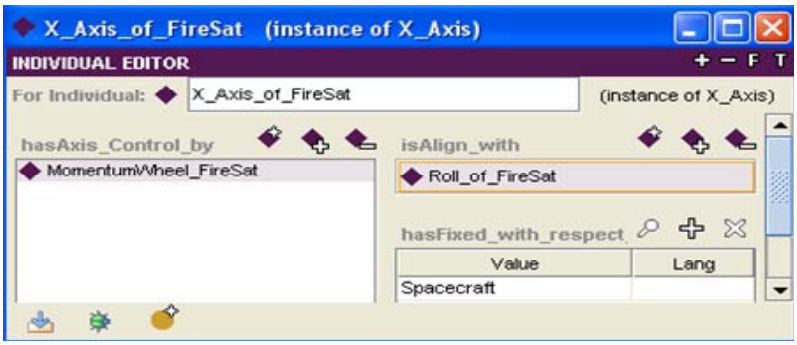


Figure 4.13 Individual X_Axis_of_FireSat

Now the individual *MomentumWheel_FireSat* is configured like Figure 4.14. Here the properties *hasGyroscopic_stiffness* and *hasRotation* are always defined because the momentum wheel rotates with a nominal speed above zero and provides gyroscopic stiffness as discussed before. The *MomentumWheel_FireSat* is mounted along the Y axis of the spacecraft and used to control X and Z axis by providing gyroscopic stiffness in these axes. Thus the momentum wheel will provide control over the worst case disturbances and also provide the baseline requirements when the spacecraft is on orbit with an accuracy of 0.1^{o} of the yaw axis (*Z_Axis_of_FireSat*).

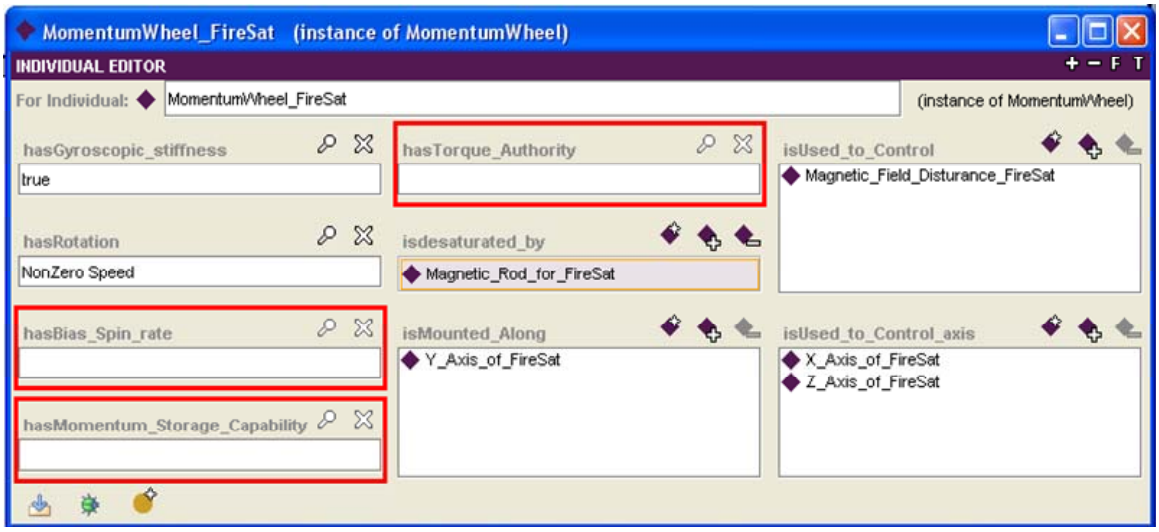


Figure 4.14 The individual MomentumWheel_FireSat

The optional requirements of 30^{o} slewing of yaw axis within 10 min time also need some kind of control and some momentum dumping mechanism is required to desaturate the momentum wheel. There are several options that can be chosen for the optional requirement:

1) Using thrusters to provide 30^{o} slewing, and momentum wheel dasaturation but include extra weight, cost. Unfortunately thrusters also induce nutation problem if not properly align with center of mass, contamination of surface and limited lifetime.

2) Using reaction wheel for optional 30^{o} slewing, and using magnetic torquers for desaturation. They are cheap can also be used to compensate the residual magnetic field of spacecraft or small disturbances, but are slow.

3) Using reaction wheel for optional 30^{o} slewing using thrusters for desaturation but again with same consideration as (1).

All the options need to be evaluated to find the cost effective and suitable one for the mission, but only option 2 is illustrated here because of the limitation of the

thesis size. As it is compulsory to consider some momentum damping or desaturation mechanism for momentum or reaction wheels, therefore same thrusters can be used for desaturation of reaction wheel. Figure 4.15 shows the configuration of individual *ReactionWheel_for_FireSat* which is mounted with *Z_axis_of_FireSat* (yaw axis), as reaction wheel can provide torque only about its spin axis.

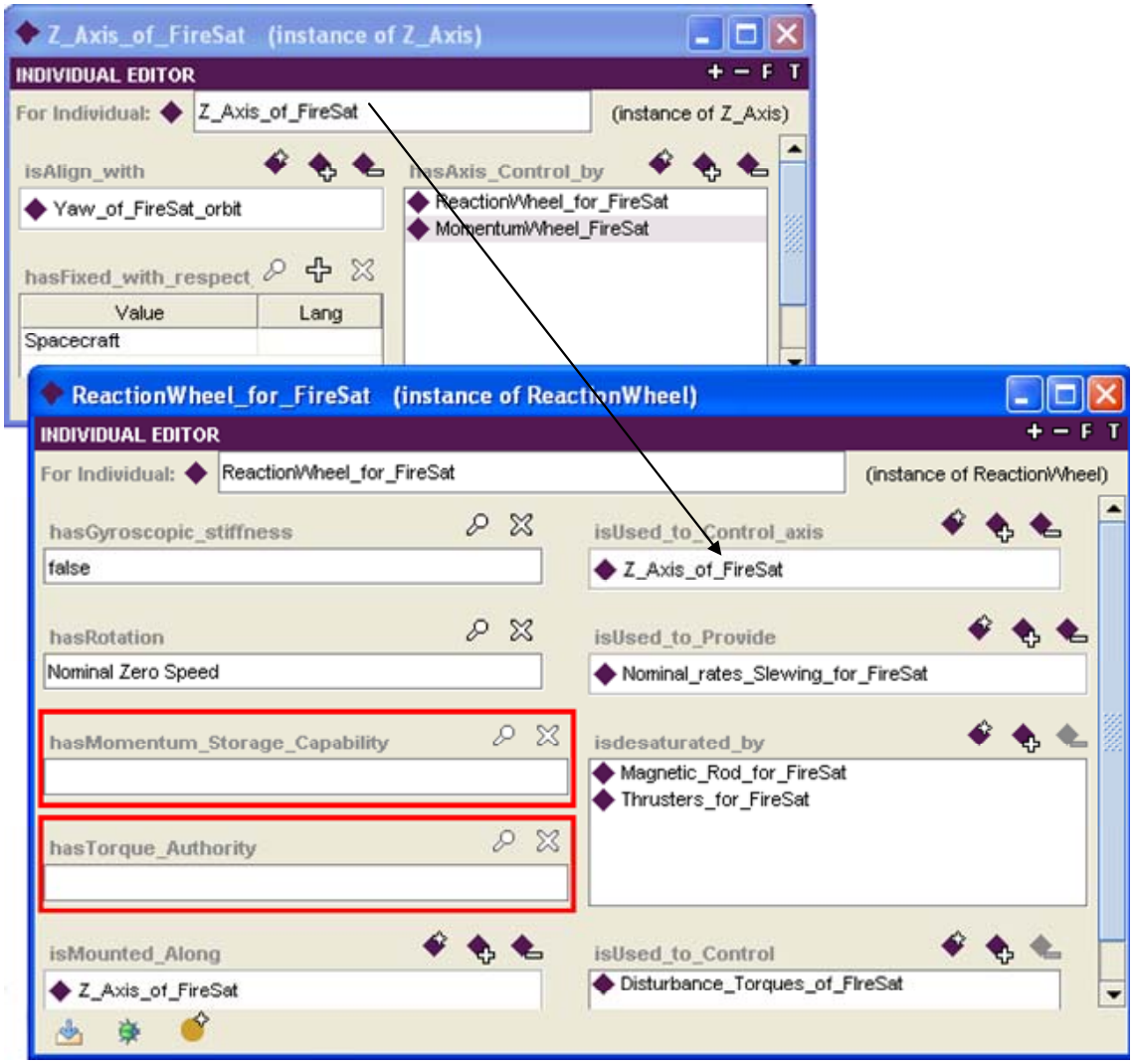


Figure 4.15 The configuration of individual ReactionWheel_for_FireSat

The individual *Thruster_for_FireSat* in Figure 4.16 is connected with *ReactionWheel_for_FireSat* by the property *isdesaturated_by* in Figure 4.15. thrusters can also be used for external disturbances rejection.

The unwanted motion as shown in Figure 4.12 also must be neutralised and can be done using again with magnetic torqures or thrusters, and the individual *Thrusters_for_FireSat* is considered for unwanted motion control, that is nutation, precession or wobble control in this example.

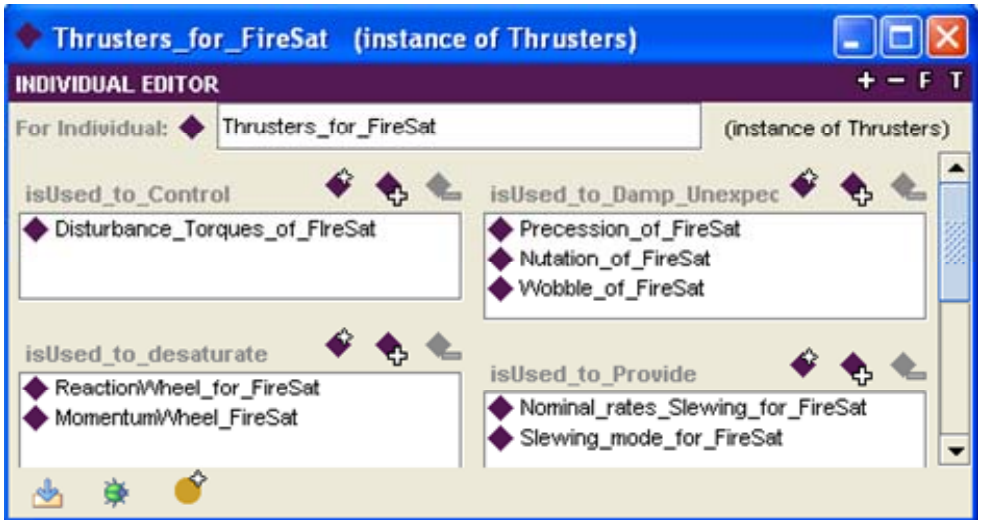


Figure 4.16 the individual Thrusters_for_FireSat

After deciding all the necessary control hardware, it is required to calculate the size of the actuators that need to counteract the disturbances and satisfy the pointing requirements. As it is already discussed the momentum wheel will be sized for baseline requirement, that is when spacecraft is on station. To reject the worst case disturbance, the reaction wheel be sized for optional slew requirement, and the thrusters sizing calculation for momentum dumping of the wheels.

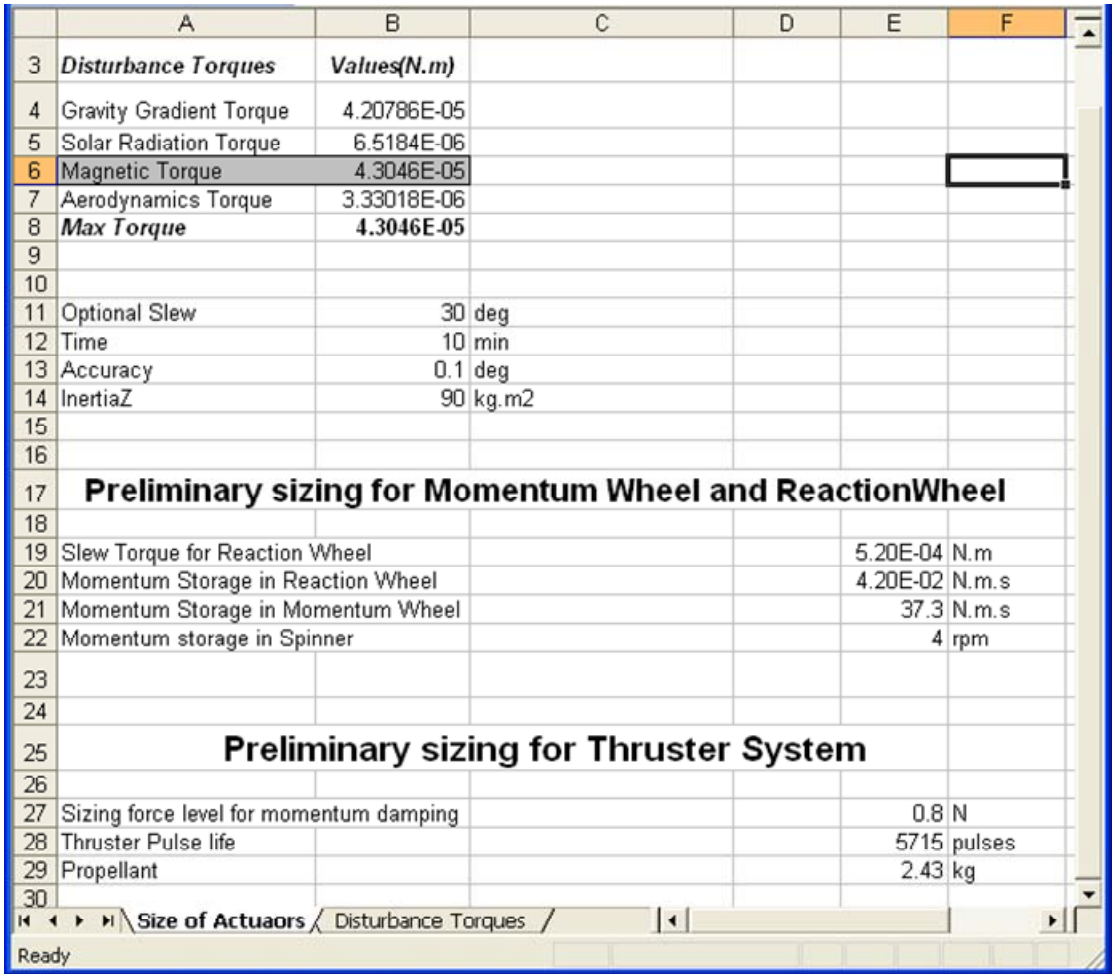
The sizes of the wheels depend upon the required momentum storage capacity and the torque authority and will be calculated using the simplified equations from (Wertz and Larson, 1999).

Now similar steps as Section 4.1.3, will be followed to calculate size of actuators by exporting all required data to Excel. The values of disturbance torques already exist in the Excel workbook and the common input parameters for the system are also already imported to calculate the disturbance torques. The calculated results are shown in Figure 4.17. To provide 30^{o} slew in 10 min time, the required torque for the reaction wheel is $5.20*10^{-4} N.m$ and the momentum storage is $4.20*10^{-2} N.m.s$. As the momentum wheel has a bias spin rate, so for 0.1 ^{o} yaw accuracy the rate is 4 rpm and the momentum storage capacity required is 37.3N.m.s.

As thruster is chosen for momentum damping then the force of 0.8 N is required to damp the $4.20*10^{-2} N.m.s$ wheels in 1 sec burn and 2.43 kg propellant is required to provide 5715 pulses for the 5 years life time (Table 4-1).

To make it clear for the reader it is again mentioned that, the ontology is not complete hence some missing data is directly taken from the “Space Mission

Analysis and Design" by (Wertz and Larson, 1999) and the calculated results are exactly same as in the book.



	A	B	C	D	E	F
3	***Disturbance Torques***	***Values(N.m)***				
4	Gravity Gradient Torque	4.20786E-05				
5	Solar Radiation Torque	6.5184E-06				
6	Magnetic Torque	4.3046E-05				
7	Aerodynamics Torque	3.33018E-06				
8	***Max Torque***	**4.3046E-05**				
9						
10						
11	Optional Slew	30	deg			
12	Time	10	min			
13	Accuracy	0.1	deg			
14	InertiaZ	90	kg.m2			
15						
16						
17	**Preliminary sizing for Momentum Wheel and ReactionWheel**					
18						
19	Slew Torque for Reaction Wheel				5.20E-04	N.m
20	Momentum Storage in Reaction Wheel				4.20E-02	N.m.s
21	Momentum Storage in Momentum Wheel				37.3	N.m.s
22	Momentum storage in Spinner				4	rpm
23						
24						
25	**Preliminary sizing for Thruster System**					
26						
27	Sizing force level for momentum damping				0.8	N
28	Thruster Pulse life				5715	pulses
29	Propellant				2.43	kg
30						

Size of Actuaors | Disturbance Torques

Ready

Figure 4.17 Preliminary sizing for the Actuators.

4.1.4.2 Select the attitude determination hardware

Sensor technology is developing rapidly and hundreds of sensors are available from different companies. It is not always an easy task to find the best combination of sensors to satisfy the mission requirements. The ontology might help by searching the available sensors saved in the ontology. Before doing that it is required to populate the sensors into the ontology with all physical characteristics.

Obviously one option is to input every sensor as individual of class *Sensors*, but it is time consuming and not desirable when a large number of sensors data is inputted. Second option is to create a database table with sensors data in MySQL and directly input into the ontology using DataMaster plugin (discussed in Section 2.3.4.3) but some database background knowledge is required to create

the database and the table. Third option is to create table in Excel with all sensor characteristics and then convert the table to MySQL database. A software called "ExcelMySQLConverter" can easily convert table from Excel to MySQL and vice versa. Once the data is in MySQL, DataMaster plugin can directly import to the ontology. For example as Figure 4.18A, the Excel sheet Sun_Sensor contains a list of sun sensor data which is then converted into MySQL.

After establishing connection with the MySQL with DataMaster plugin the available tables and the preview of the data are shown in the down panel of Figure 4.18B. The selected table is then possible to import in the desired location as in this example the class *Reference-Sensor* is chosen as the superclass. The table sun_sensor is imported to the otology as a subclass of the chosen class with a name of *db:sun_sensor* as shown in the red circle. The class is automatically renamed with a prefix of *db:* can be changed easily if the name is not preferable. If a class with name *db:sun_sensor* already exist under the selected superclass then only the content of the table will be copied to that class.

Now the ontology environment is ready to provide the support for selecting sensors. The selections of the sensors largely depend upon the orientation of the satellite and the accuracy that they need to provide for the mission (Wertz and Larson, 1999). Horizon/Earth sensors are the best choice for the Earth orbiting satellite since they directly measure the roll and pitch angle but not the yaw angle. To measure yaw angle sun sensors can be an option but they can not measure with sufficient accuracy and not at all during eclipse.

Another approach would be a magnetometer but these do not provide 0.1^{o} accuracy due to the fluctuation of Earth's magnetic field (Sidi, 2000). So the magnetometer is not accurate enough for this mission. Third option is star sensors which provide very accurate measurement but are quite complicated and sometimes not reliable thus not suitable for this mission. So for this momentum bias system the combination of magnetometer and sun sensor should work. So it is decided that an arrangement of horizon sensor, sun sensor and magnetometer will provide complete measurement of the attitude of this momentum biased spacecraft.

Now it is time to choose and find the individuals from the chosen categories of sensors. Figure 4.19 shows sensors with accuracy greater than 0.1^{o}. Some other choices like cost, weight, manufacturer etc might be considerable during choosing sensors but these parameters are ignored here for simplicity.

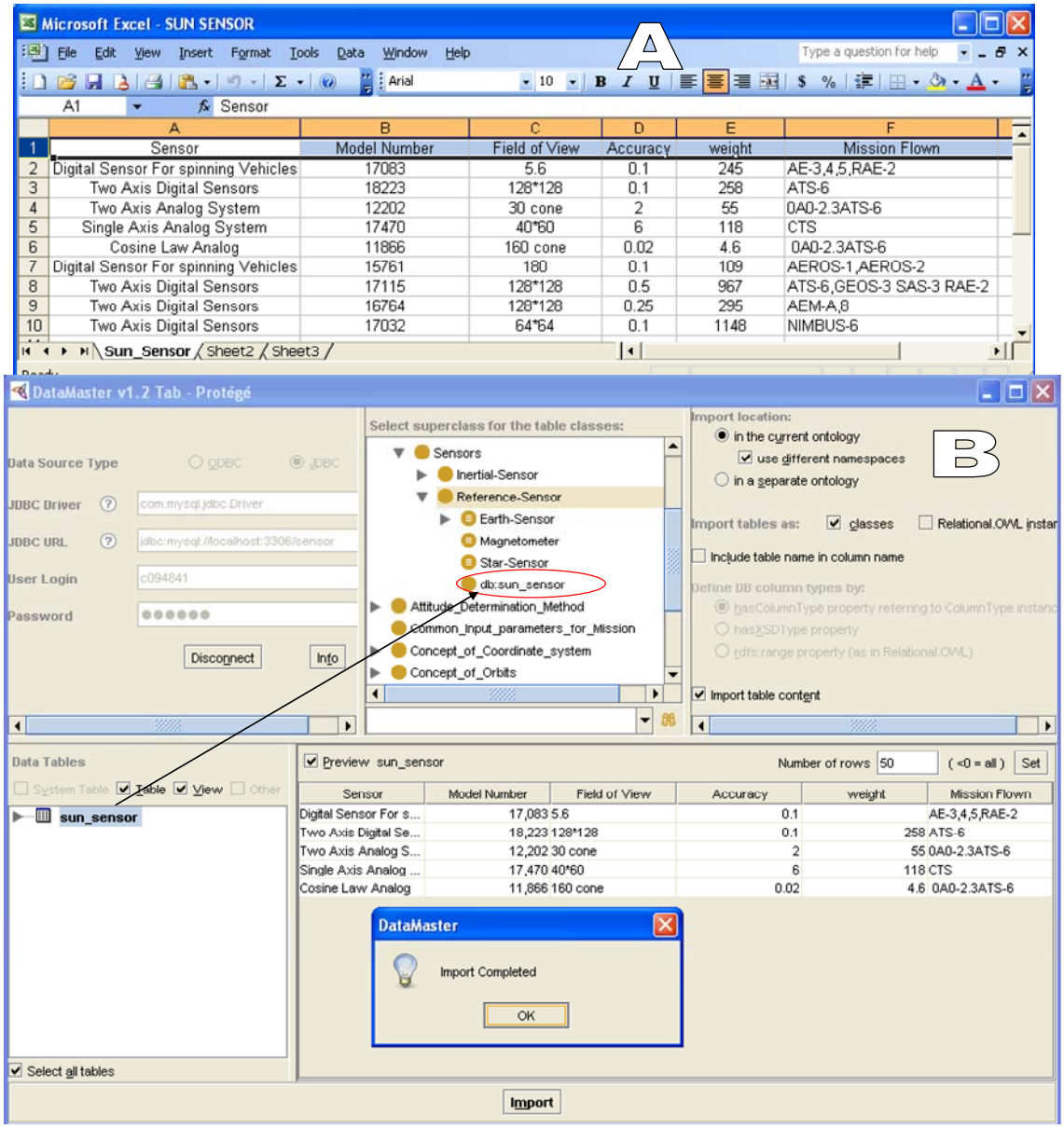


Figure 4.18 Sun sensor data is imported to the ontology using DataMaster plugin

The mandatory field *has_Attitude_Determination_Hardware* in Figure 4.12 for control method *Bias_Momentum_for_FireSat* will be completed by selecting the particular sensors. Therefore the hardware selection for *Bias_Momentum_for_FireSat* control method is completed and needs to be documented for use by the other subsystem and system designers.

The great thing is the ontology itself is an updated document and all information is available to other users of the same ontology. Any changes in selections are seen by the other engineers without any time delay. The lists of hardware are presented in this Figure 4.20 and the rationale of the hardware also can be analysed by browsing each individual.

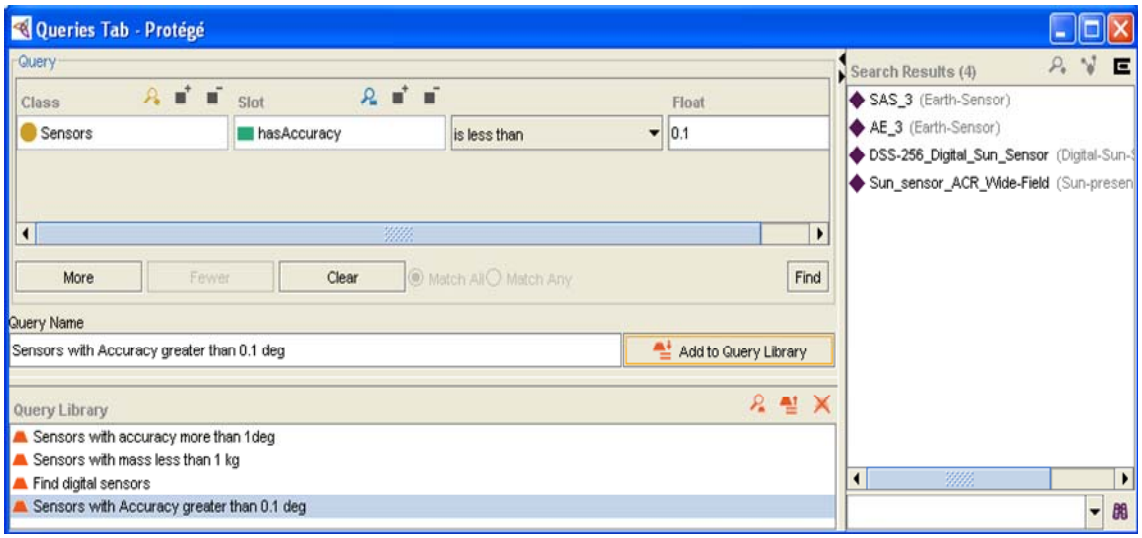


Figure 4.19 Query for the sensors with accuracy greater than 0.1°.

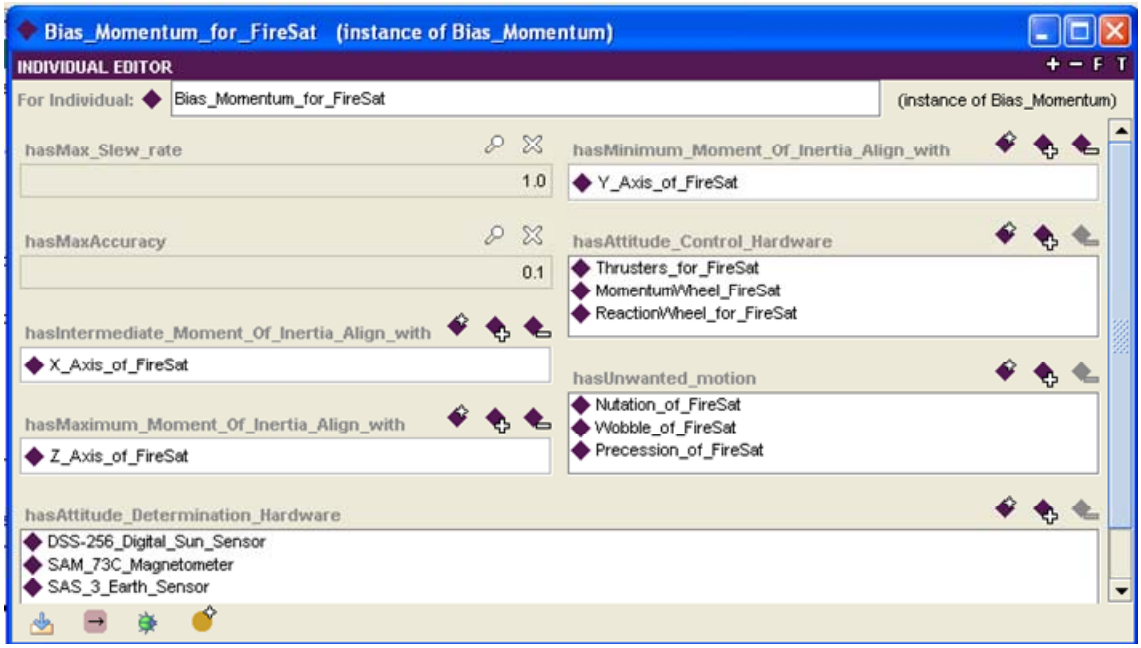


Figure 4.20 The individual Bias_Momentum_for_FireSat with determination hardware.

4.1.5 Define Determination and Control Algorithm

The control algorithm and model are implemented using Matlab and the aim of this section is to show the connectivity of the ontology with Matlab. In these step the data from the ontology will be exported to Matlab to use in the designed (Hardacre 2008) SIMULINK model. In this model the basics of a Sun-safe attitude of the Earth orbiting satellite have been covered. This attitude is used for the emergency mode or "Safe mode" of operation which needs to activate in the event of failure during normal mode as shown in Figure 4.3. "The top level

requirements of this emergency mode are to point the solar array normal to the Sun to ensure:

1. Power of satellite
2. A constant thermal environment
3. Protection from the Sun of potentially sensitive equipment, such as telescope." (Hardacre, 2008).

The key parameters assumed in the ADCS model, such as spacecraft inertia and mass properties, sensors characteristics, actuators performances, thrusters layout, wheels characteristics are taken from the ontology as it is the central repository for whole system.

A simple example of importing the mass properties of spacecraft is illustrated here for directly using in the model. Same steps as Section 4.1.3 (paragraph 2) have been followed to export the *hasInertiaX*, *hasInertiaY* and *hasInertiaZ* properties (inertia properties) from *Common_Input_parameters_for_FireSat _ Mission* individual using Query Tab. The exported CSV file is used in Matlab as described in Section 3.5.

As one objective of this project was to show the possible way of communicating the data from ontology to Matlab thus the inertia values are used in the Matlab simulation.

4.2 Analysis of the developed ADCS model for FireSat

The developed ontology has been analysed by several means. The following sections discuss the procedures.

4.2.1 Implementing the SWRL rules

An ontology can produce some inferred intelligence using SWRL rules from the existing knowledge (O'Connor et al., 2005). For example, the *FireSat_Orbit* is an individual of class *Orbits* as shown in Figure 4.21, so asserted type of this individual is *Orbits* also shown in red circle. This orbit has some parameters like altitude 800km, inclination 90^{o} and the value of eccentricity is 0. Using these input parameters it is possible to introduce some other characteristics of this the orbit like the inclination of this orbit is 90^{o} so it is a polar orbit. The ontology is able to construct this kind of intelligence by using SWRL rules. SWRL rules are written in terms of OWL classes, properties and individuals.

For this above example, any individual of class *Orbits* has a property *hasInclination* is equal 90.0 will be Polar orbit which can be written like this.

Orbits(?x) ^ hasInclination(?x, ?y) ^ swrlb:equal(?y, 90.0) → Polar_orbit(?x)

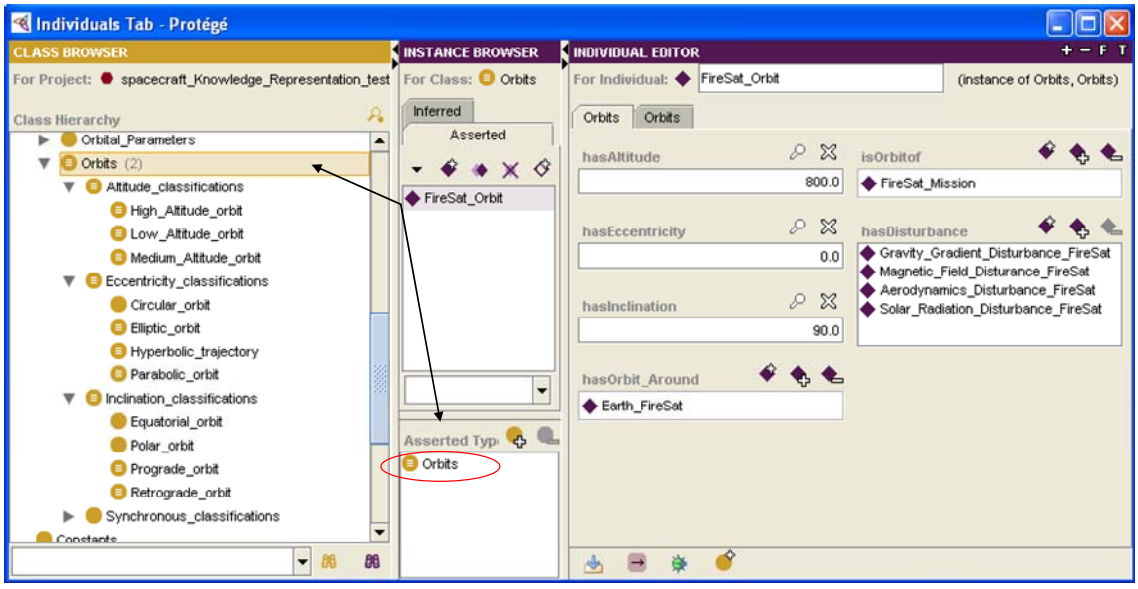


Figure 4.21 The individual FireSat_Orbit of class Orbits before executing SWRL rules.

Executing this rule would have the effect of classifying individuals of class *Orbits* to also be a member of class *Polar_orbit*. Figure 4.22 shows some SWRL rules for class *Orbits*.

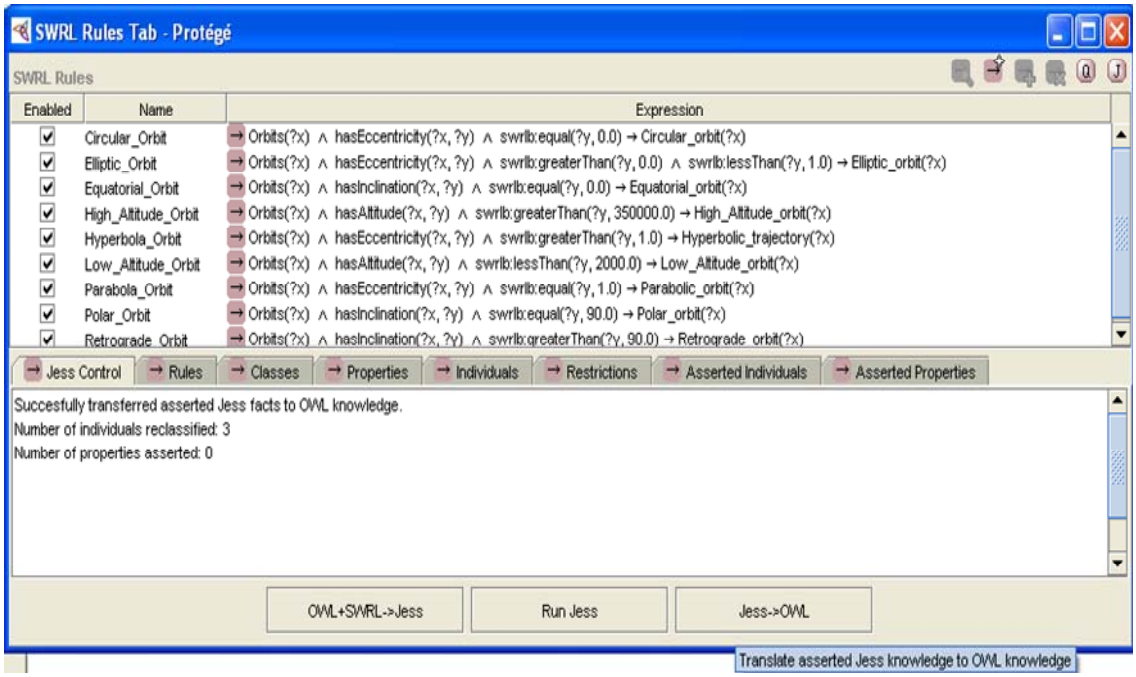


Figure 4.22 SWRL rules for class Orbits.

After executing these rules, OWL knowledge and SWRL produce further knowledge about the individual *FireSat_Orbit*. Now the individual *FireSat_Orbit* is also the instance of *Circular_orbit*, *Low_Altitude_orbit*, and *Polar_orbit*.

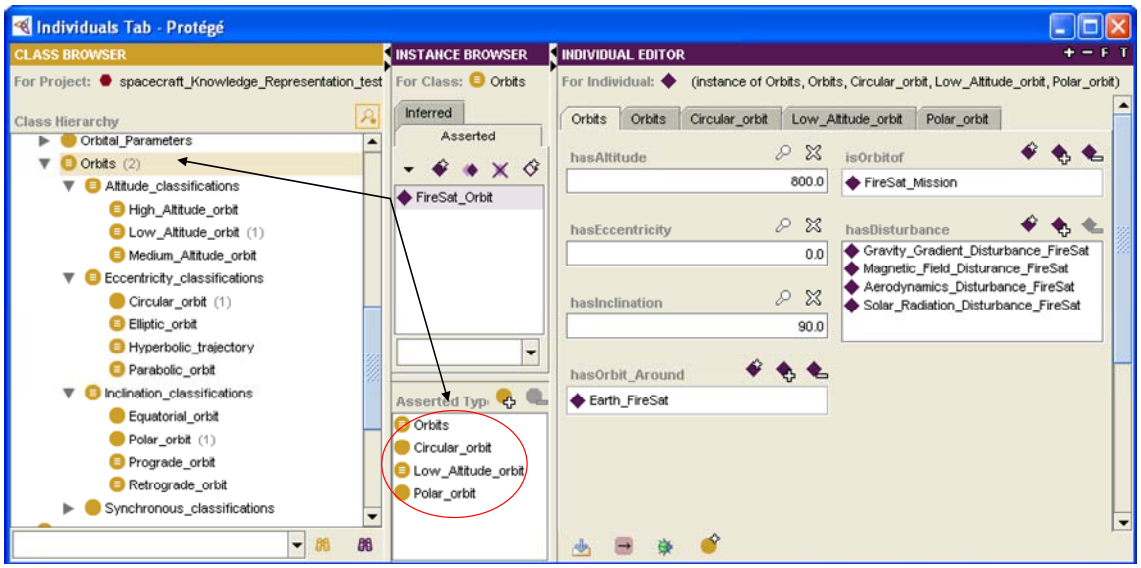


Figure 4.23 The individual FireSat_Orbit of class Orbits after executing SWRL rules.

Thus the ontology can help the user to produce additional knowledge about the domain.

4.2.2 Analysis of the ontology and ADCS model using graphical tools

Basic understanding of the structure of the ontology is required to design an effective model. Though it is not a big ontology, a complete one for Spacecraft system design can consist of hundred to thousands of concepts (classes) and the interface is increasingly complicated as the model grows big. After establishing the model for any mission it is often required to evaluate the model to the high level managers, communities whether the concepts, initial design, and potential requirements meets the mission requirements. Even the dataflow into an ontology is a complicated analysis.

Graphical representation assists the complex design study and supports the interactive data exploration. The graphical visualization tools OWLViz and Jambalaya are used to visualize the model.

4.2.2.1 Analysis the ontology using OWLViz

The OWLViz plugin was used in this project to visualize the hierarchy classification of the ontology. As the ontology grows bigger the class hierarchy becomes too complex to explore manually, thus OWLViz helps to understand easily graphically. Both asserted and inferred classifications can be presented

using this plugin. The asserted models of ADCS ontology are given in Appendix B.1

4.2.2.2 Analysis the ADCS model using Jambalaya

The visualization tool Jambalaya provides excellent layouts, navigation and querying facility to meet user preferences. The techniques such as nested graph view, combined pan, zoom and fisheye-view with different layouts such as Radial, Spring, horizontal Tree and vertical Tree Layout are powerful enough for analysis and navigation of complex ontologies (Lintern and Storey). Jambalaya allows searching of classes and individuals and able to navigate the selected classes or individuals from the searching results, and also help users to visualize the nodes connected through defined relationships.

Figure 4.24 shows the graphical representation of FireSat ADCS grid view (alphabetic order) model with its all components and interface with other individuals. The individual *Bias_Momentum_for_FireSat* is separated here from the panel to view the incoming and outgoing links clearly. Any selected link (bold orange) provides the information about the property by which nodes are connected.

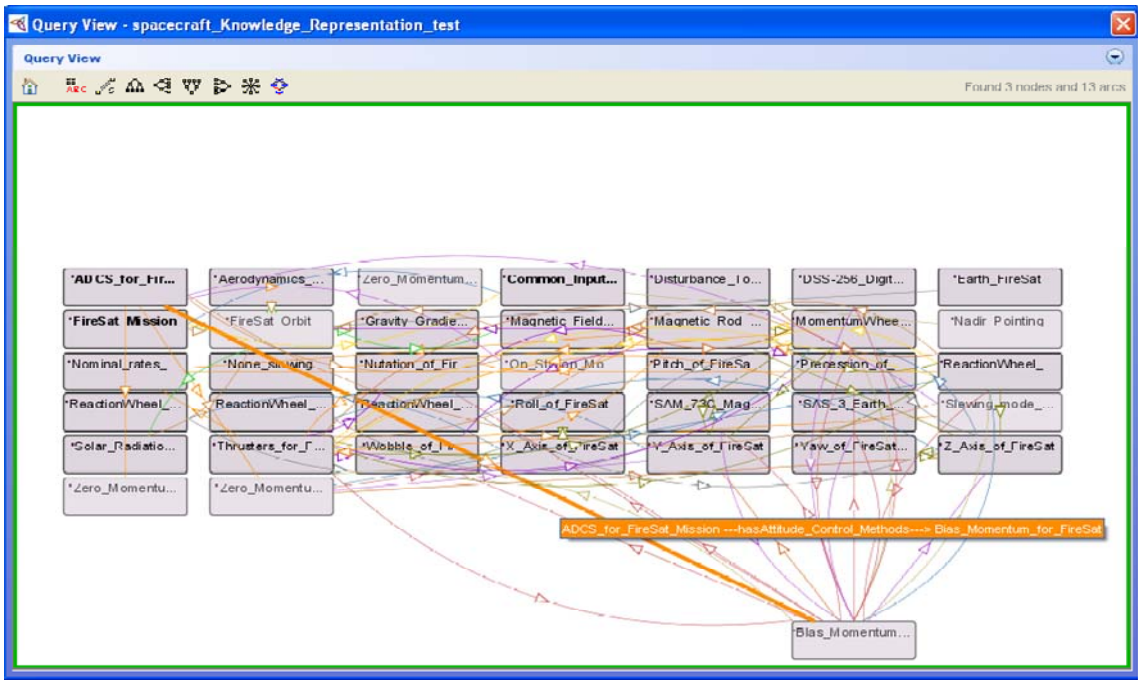


Figure 4.24 Analysis all concepts of FireSat ADCS model using Jambalaya.

Jambalaya is a powerful tool for analysis and navigation of complex ontologies and provides huge support to the user with different visualization facilities. It is not only useful for understanding the otology but also helpful to explain the ontology or the interface to others who are not even used to with the system.

Chapter 5 Discussion

In this chapter the author tried to connect the results of this project with the objectives.

All space missions consist of a set of concepts, and the arrangement of these concepts form a space mission architecture (Wertz and Larson, 1999). The practical scenario of ADCS subsystem design needs involvement of expert team, with access of several kinds of documentation like scientific paper, previous mission's reports, books and lot of studies to gather the information about the domain and the concepts. So during the design process large amount of pre-existing knowledge is accessed to get adequate information about specific concepts of the domain. One important thing is that the knowledge representation system consists of a significant amount of knowledge based on opinions and backgrounds of multiple sources. The relevant knowledge of the ADCS system was only gathered from mainly "Space System Design and Analysis" by (Wertz and Larson, 1999) and several other books on ADCS system design. But in a real case knowledge would be taken from a large number of sources, including subsystem experts.

The aim of the project was to build an intelligent tool to support the ADCS designer's task by providing information to those who are involved in the preliminary design process. Providing conceptual designers with knowledge supports a global view of the system that is also cost effective in terms of life cycle cost (Ruiz-Torres et al., 2006). The ontology will be a central repository with shared knowledge of the experts. Through reusing the ontology, the knowledge and expertise can also be reused and save a significant amount of time and cost.

Practically space industrial personnel practise the trend of following the previous successful missions as there are involvements of risks of failure. Previous successful mission knowledge that means pre-existing knowledge is vital in mission design unless very new ideas are tested. The ontology presents the behaviour of existing concepts; supports reusing the pre-existing knowledge for future design. It also helps to analyse the explicit information needed during the design process, speeding up the process and improving the overall quality of the project.

The ADCS ontology consists of hundreds of related terms or concepts of ADCS subsystem. All the concepts were broken down with the properties and relationships with other concepts of this domain. Before starting ontology design, it was not only important to decide which concepts are going to be presented, it

was also crucial to think why they are presented. As an ontology is the specification of concepts, therefore analysing the ontology one could know about any concept, its properties and the relations that it holds with other terms of this domain. For example the ontology presents the properties and characteristics of the wheels, how the wheels are used in the system, why they are used, where used and how they are used? The physical properties of the sensors are also defined in the ontology to support any comparison analysis. Thus the second objective was fulfilled by showing, how each and every concept exists in the system with properties and constraints and how they are related to each other.

The third objective was to develop a model for spacecraft system design, mainly emphasising the preliminary design procedure of the attitude determination and control subsystem, and to evaluate and validate the system using one or two previous missions. The example FireSat ADCS subsystem design procedure using the ontology presents the possible supports during design by extracting the required knowledge from a vocabulary of domain knowledge. Though space technology is quite young compared to other modern technologies such as aircraft technology (Sidi, 2000), control concepts are based on vast choice of physical phenomena such as different stabilization methods. It is essential to understand the details of each before deciding any engineering choice. Therefore various concepts need to be analysed, and compared in depth to make the correct choice. It is often required to evaluate alternative methods for meeting end users and operators needs: this can be illustrated using the ontology as it is a composed knowledge repository. Four possible options for FireSat ADCS subsystem control design have been illustrated and compared to show the simplicity of using ontology which also minimizes the amount of human efforts.

Basic understanding of the structure of the ontology is required to design an effective model. A complete ontology for spacecraft system design could consist of hundreds to thousands of concepts (classes) and the interface is increasingly complicated as the model grows big. After establishing the model for any mission it is often required to evaluate the model to the high level managers, communities whether the concepts, initial design, and potential requirements meets the mission requirements. Even the dataflow into an ontology requires complicated analysis for reusing it. Graphical representation assists the complex design study and support the interactive data exploration. The graphical visualization tools OWLViz and Jambalaya are used to visualize the model. It is not only useful for understanding the ontology but also helpful to explain the ontology or the interface to others who are not even familiar with the system.

The main objective of the ADCS subsystem to keep the orientation of the spacecraft to the desired location usually affected the disturbances and control requirements. The modelling of those disturbances is the unavoidable part of ADCS design. The defined terms of the ontology was used to calculate the

disturbances using simplified equations (Wertz and Larson, 1999) in an Excel workbook. The size of the actuators was calculated primarily depending upon the size of the disturbances and the slewing requirements of the mission. Simple macro programming in Excel was done to do these calculations quickly and simply. Though the expressability of OWL DL is extended using SWRL rules, very limited basic calculations were also possible to do in Ontology using mathematical build-ins of SWRL. And this was the forth objective of this project. The possible way of sharing the knowledge from the ontology to other software tools like Matlab has been shown

5.1 Ontology evaluation and validation

Before making any suggestion of completeness of the ontology it is worth evaluating the ontology. The development tools should assist the evaluation of the ontology to avoid common mistakes. Most modern programming languages support error checking by the compiler button to find the errors instantly. Another approach of detecting error is *test-case* (Knublauch et al., 2004a). But an ontology is different from a piece of code. Nicola Guarino says "*We can't evaluate ontologies in terms of their correctness with respect to a given process specification, described, for instance using an I/O function. Indeed ontology are not software processes-rather, they belong to the class of data models*" (Sure et al., 2004). There are several controversies and proposed ideas for evaluating the ontologies. For this project the recommendation by Gomez-Perez, has been followed (Gomez-Perez, 2001). Two types of evaluations: technical evaluation and user evaluation have been done for this ontology.

Technical evaluations are carried out by the developers and the content of the ontology should be evaluated on the basis of the following ideas (Sure et al., 2004):

1. The content of the ontology should be evaluated during entire ontology life cycle.
2. The ontology building tools should support the evaluation during the ontology building process.
3. The ontology content evaluation fully depends upon the knowledge representation theory in which the ontology is implemented.

The contents of the ontology have been checked and the following conclusions:

1. The ontology is syntactically correct for the Protégé environmental test (Knublauch et al., 2004a).
2. The ontology is concise, doesn't contain redundant knowledge.
3. The ontology consistency was checked by RACER.

The consistencies of an ontology establish the logical completeness and increase the confidence of conflict avoidance among the logics. By passing these evaluations, it is not proved the absence of problems but makes it safer to use and confirms that there are no logical errors.

Natalya F. Noy stated that "*the only true way of evaluating an ontology is to use it in applications and assess the applications' performance*" (Sure et al., 2004). So users of an ontology actually determine the validity of that ontology. All other parameters like completeness, consistency, absence of redundancy, and correctness are nothing to do with the ontology user, unless the ontology can fulfil the requirements of the project. For instance an ontology might be complete, consistent and free of redundancy but does not contain enough concepts that are needed to describe the situations of ADCS; then that ontology will not be suitable. An example of ADCS subsystem design of FireSat mission has been done using the developed ontology to validate the usefulness of this ontology.

5.2 Limitation of the software

A few drawbacks due to the limited capability of OWL have been experienced and are discussed below:

A serious limitation of OWL-DL in using numeric range constraints for representing for example "the inclination of a retrograde orbit is greater than 90 deg" or "the eccentricity of an elliptical orbit is between 0 to1". This makes it impossible to define some classes using only OWL DL. SWRL rules have been added to solve this problem to some extent. For example the following SWRL rule will classify the orbit to the retrograde orbit if the inclination is greater than 90 deg.

Orbit(?x) ^ hasInclination(?x, ?y) ^ swrlb:greaterThan(?y, 90) -> Retrograde_Orbit(?x)

Another way of doing it is to add the property *hasInclinationGreaterThan90deg* for defining the class *Retrograde_Orbit*. Then the user has to be careful to assign the value to this property because the reasoner will not restrict the value, which might not be greater than 90.

Though the required calculations have been done using Excel, some fundamental calculation facilities are essential to define some concepts. For example "Orbit radius = orbit altitude + radius of centred body", which is not supported by only OWL DL. Again using some build-ins in SWRL, this facility has been provided indirectly (as discussed in Section 4.2.1).

For three axis stabilization method the maximum, minimum and intermediate moment of inertias can be aligned with any of the axes but must be different, depending upon the mission requirements. For example these rules have been added for three axis stabilization method using three reaction wheels:

hasIntermediate_Moment_Of_Inertia_Align_with some
(Spacecraft_body_axis and (hasControl_by some Reaction_Wheel)

hasMaximum_Moment_Of_Inertia_Align_with some
(Spacecraft_body_axis and (hasControl_by some Reaction_Wheel)

hasMinimum_Moment_Of_Inertia_Align_with some
(Spacecraft_body_axis and (hasControl_by some Reaction_Wheel)

Now if the maximum moment of inertia axis is aligned with X axis then minimum moment of inertia axis or intermediate moment of inertia axis must not be aligned with X axis again. But it is not possible to restrict by using OWL only. Again for example, for three axis stabilization method the axis can be controlled by reaction wheels, momentum wheels or CMGs, but for the above example the rules say that the axes must have to be controlled by some reaction wheel. But this field will not indicate error if reaction wheel is not selected. So during assigning the value to a property it is important to see the rules to avoid the errors.

SWRL supports monotonic inference only (O'Connor, 2008). Hence, SWRL rules cannot be used to modify information in an ontology, it is also not possible to retract or remove information from an ontology using SWRL. For example, the initial mass of the spacecraft is 215kg.

Spacecraft(?s) ^ hasMass(?s, ?mass) ^ Commom_Input_Parameters(?c)-> hasMass(?c, ?mass)

This rule will add a property *hasMass* to the class *Commom_Input_Parameters* with the value of mass 215kg from *Spacecraft* class. If, for example, the spacecraft mass is changed to 300kg, after running this rule again the previous assigned mass (250kg) in *hasMass* property in *Commom_Input_Parameters* class will not be changed. Hence it is required to delete the value of hasMass property manually to reassign the correct value.

The ontology is for only ADCS but all the subsystems are dependent on each other. For example the information of centre of mass, inertia constraints come from the structure subsystem. As this ontology does not contain other subsystems, some classes are included purposely like class *Common_Input_Parameters* which contains all common parameters shared by all subsystems and might contain some redundancy.

The basis requirement of an ontology development process is the involvement of expert knowledge and multiple resources. Very fundamental presentation of ADCS ontology has been made due to the lack of practical design knowledge, lack of information about the attitude control devices and time restrictions.

Though the ontology ADCS is a small one, as the ontology grows larger it becomes beyond the capability of a single person to maintain it in a logically correct state. A full ontology includes all the subsystems of a space mission, a team of multiple ontology engineers is required to build and maintain such a large ontology. Protégé OWL provides an extensive support to work simultaneously on the same database, without waiting for weekly cycles of editing, facilitating real time collaboration (Robert et al., 2005). Nevertheless there are a few disadvantages of multi user mode of editing. This mode introduces the possibilities of mistakes and conflict of knowledge presentation if multiple persons are allowed to edit the ontology simultaneously. A discussion of relative advantages and disadvantages of collaborative ontology editing has been presented in (Seidenberg and Rector, 2007) and several solutions also have been raised for these problems. To perform large scale ontology engineering there is a clear need for a multi-user ontology development methodology.

Chapter 6 Conclusion

The potential of a knowledge based system of attitude determination and control subsystem design has been analysed in this project. Ontology is a powerful approach of capturing, integrating the domain expert's knowledge allowing reusability of existing knowledge. The state of the art of ontology engineering is different from the formal software engineering, and permits to present the human knowledge linguistically as computer understandable form with logical correctness. The major effort was modelling and implementing the ontology building process in this project. Some significant conclusions can be stated:

Greater understanding and practical knowledge are required for representing the knowledge of the domain. During presenting the knowledge of an object in an ontology in a declarative form, the formal and detailed understanding of the object's behaviour and characteristics are required in a suitable manner within its domain. A major issue in knowledge representation is the amount of information.

Expert's involvements are necessary for a long term basis during the ontology design phase for more detailed formal knowledge representation. This involvement has the advantage that it helps to retain knowledge after a particular expert leaves a role, for example. But this may of course make the experts wary of giving too much of "themselves" and their knowledge to go into an ontology: they may fear that it makes them less valuable themselves.

Logical adequacy, the representation should be capable of making all the required distinctions. For example, unless the representation is able to distinguish between the particular illumination of a solar panel that gives a particular power output, and the general statement that "for any illumination of a solar panel, there is a level of output power associated with it", it is impossible to represent the idea that illumination of a solar panel produces power.

Completeness and distinction are the keys of knowledge presentation, for example if a statement is written that "The inclination of a Prograde orbit is less than 90 deg" then equatorial orbit (inclination 0 deg) will also be a Prograde orbit unless completely specified that " The inclination of a Equatorial orbit is 0 deg". For this example the classification "The inclination of a Prograde orbit is less than 90 deg" is not incorrect; it just doesn't necessarily give all the information that is required for Equatorial orbit. It is not possible to express around or nearly 90 deg because it depends on a binary relation.

An ontology does not mean to present all features/definitions of an object that it can have, only the related knowledge of its domain of interest are expected to be

presented in the ontology. For instance, how much power a sun sensor will consume is not the concern of ADCS designer; must be very important information for power system analyst. But a complete ontology for all subsystems obviously needs both information. It actually depends on the area/scope that needs to be covered to define the object completely.

Expressability of data restrictions is necessary within the ontology editor environment for an engineering field like space science. For example the comparison facilities of numerical values are essential.

Continuous inconsistency checking to verify the consistency of a particular design by using automated reasoning is mandatory to retain logical accuracy.

Some automatic classifications using SWRL rules can be used to infer further valuable knowledge.

Visual tools permit analysis of an ontology; these are good and easy approach to understand and allow overall investigation of the domain knowledge. The various options of graphical presentation of the visual tools like OWLViz, Jambalaya make it possible to understand the dataflow into the ontology and these also allow a non-expert to start to understand the knowledge area.

An ontology is a repository of knowledge, expected to generate a greater understanding of the process of knowledge capture and representation, and to identify the state-of-the-art and methodologies used in other industries in the context of spacecraft systems design.

Building such an ontology has several benefits: it presents the knowledge about the specific domain of interest by explaining the related pre-existing knowledge about the domain, moreover the design process itself produces a large amount of knowledge by documentation of information during design process. And thus hopefully improve, enhance, or speed up the design process and prevent loss of knowledge if personnel leave at the end of a project where useful lessons were learned. Again it is also reasonable that some new ideas will arise during the design process and will add extra bits in the domain knowledge (Pazienza et al., 2005).

Also, some way to make changes made in one area (e.g. spacecraft mass, as mentioned earlier) propagate through the ontology and databases would be very useful over other programming languages.

But building an ontology needs a great involvement of experts and the users, is a time consuming process. As it depends on the description logics, it is difficult to maintain the ontology in a logically correct state as the system grows larger and

need continuous checking the consistency. Ontology editing tools should be more flexible for engineering field and it is a crucial to add the mathematical capability to make it really useful for space technology.

6.1 Future works

Currently the simulation of control design using satellite control toolbox has been done in Matlab by taking the inputs from the ontology. Though Matlab and Protégé are currently not integrated together, they need to be combined to make it a useful decision making tool. Two ideas could be useful for future progress in this context.

1. The Matlab Database Toolbox allows exchange of data with relational databases, and can be used to acquire knowledge from the ontology (saved in database format) to the Matlab environment for further analysis.

2. Building a Matlab plugin for the Protégé environment will be the best option for making the ontology and Matlab linked. A library called JMatLink, connects Matlab with Java, and allows using the functions available in Matlab using the Java programming language (Güemes, 2006) .

A Bioinformatics Toolbox to analyze microarray gene data using the Gene Ontology Database already exists in Matlab to shift from ontological theory to application easily (The MathWorks, 2008). If the ontology "Space System Design" becomes a rich one there are lots of possibilities of building such a toolbox in Matlab for design analysis.

In more advanced forms, such a knowledge base system would also be a useful tool for education and training

REFERENCES

Antoniou, G. and Harmelen, F. (2004), "Web Ontology Language: OWL", *Handbook on Ontologies,* , pp. 67–92.

Bechhofer, S., Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F. and Stein, L. A. (2004), "OWL Web Ontology Language Reference", *W3C Recommendation,* vol. 10, pp. 2006-2001.

Bray, T., Paoli, J. and Sperberg-McQueen, C. M. (2000), "Extensible Markup Language (XML) 1.0", *W3C Recommendation,* vol. 6.

Brown, C. D. (2002), *Elements of Spacecraft Design,* Aiaa.

CHISEL Authors (2008), *SHriMP | The CHISEL Group, University of Victoria*, available at: http://www.thechiselgroup.org/shrimp (accessed 06/18).

Davis, R., Shrobe, H. and Szolovits, P., (Spring 1993), *What is A Knowledge Representation?*, 1st ed., AI Magazine.

Duineveld, A. J., Stoter, R., Weiden, M. R., Kenepa, B. and Benjamins, V. R. (2000), "WonderTools? A comparative study of ontological engineering tools", *International Journal of Human-Computer Studies,* vol. 52, no. 6, pp. 1111-1133.

Fensel, D. and Straatman, R. (1998), "The essence of problem-solving methods: making assumptions to gain efficiency", *International Journal of Human-Computer Studies,* vol. 48, no. 2, pp. 181-215.

Fortescue, P. W. and Swinerd, G. (2003), *Spacecraft Systems Engineering,* Wiley.

Gomez-Perez, A. (2001), "Evaluation of ontologies", *International Journal of Intelligent Systems,* vol. 16, no. 3, pp. 391-409.

Gruber, T. (1995), "What is an Ontology", *Knowledge Acquisition,* .

Gruninger, M. and Fox, M. S. (1995), "Methodology for the Design and Evaluation of Ontologies", *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI,* vol. 95.

Güemes, A. H. (2006), "A Prototype System for Automatic Ontology Matching Using Polygons", *Computer and Electrical Engineering, Jönköping University,* .

Hardacre, S.P., (2008), *Emergency Sun Mode Simulation in MATLAB*, Cranfiled University, Lecture Notes.

Heflin, J. (2003), "Web Ontology Language (OWL) Use Cases and Requirements", *W3C Working Draft,* vol. 31.

Horridge, M. (2008), *OWLViz-A visualisation plugin for the Protege OWL Plugin*, available at: http://www.co-ode.org/downloads/owlviz/OWLVizGuide.pdf (accessed 06/15).

Horridge, M., Knublauch, H., Rector, A., Stevens, R. and Wroe, C. (2004), "A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools", *The University of Manchester,* .

Ikeda, M., Seta, K., Kakusho, O. and Mizoguchi, R. (1998), "Task ontology: ontology for building conceptual problem solving models", *Proceedings of ECAI98 Workshop on Applications of ontologies and problem-solving model,* , pp. 126-133.

John, S. (2000), *Knowledge Representation: Logical, Philosophical, and Computational Foundations,* Pacific Grove.

Knublauch, H., Fergerson, R. W., Noy, N. F. and Musen, M. A. (2004a), "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", in *The Semantic Web – ISWC 2004,* Springer, , pp. 229–243.

Knublauch, H., Musen, M. A. and Rector, A. L. (2004b), "Editing description logics ontologies with the Protege OWL plugin", *International Workshop on Description Logics,* .

Lambrix, P., Habbouche, M. and Perez, M. (2003), "Evaluation of ontology development tools for bioinformatics", *Bioinformatics (Oxford, England),* vol. 19, no. 12, pp. 1564-1571.

Lassila, O. and Swick, R. R. (1999), "Resource Description Framework (RDF) Model and Syntax Specification", *W3C Recommendation,* vol. 22, pp. 2004-2003.

Lintern, R. and Storey, M. A. "Jambalaya Express: Ontology Visualization-On-Demand", .

Martin, C. and Adrian, I. K. (2008), *Project One*, available at: http://dac.icore.at/one/home (accessed 06/18).

Mizoguchi, R. and Ikeda, M. (1997), "Towards Ontology Engineering", *Proc.of PACES/SPICIS'97,* , pp. 259-266.

Mizoguchi, R., Welkenhuysen, J. and Ikeda, M. (1995), "Task Ontology for Reuse of Problem Solving Knowledge", *Towards Very Large Knowledge Bases,* , pp. 46–57.

Musen, M. A. (2000), "Ontology-Oriented Design and Programming", *Knowledge Engineering and Agent Technology,* , pp. 1-20.

Musen, M. A. (1989), *Automated Generation of Model-Based Knowledge-Acquisition Tools,* Morgan Kaufmann.

Musen, M. A., Fergerson, R. W., Grosso, W. E., Noy, N. F., Crubezy, M. and Gennari, J. H. (2000), "Component-Based Support for Building Knowledge-Acquisition Systems", *Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000),* vol. 194.

Noy, N. F. and Mcguinness, D. L. (2001), *Ontology development 101: A guide to creating your first ontology.*, available at: http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html (accessed 05/27).

Nyulas, C., O'Connor, M. and Tu, S. (2007) "DataMaster–a Plug-in for Importing Schemas and Data from Relational Databases into Protégé", .

O'Connor, M. J. (2008), *SWRLLanguage FAQ*, available at: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ (accessed 06/11).

O'Connor, M. J., Knublauch, H., Tu, S. W., Grossof, B., Dean, M., Grosso, W. E. and Musen, M. A. (2005), "Supporting Rule System Interoperability on the Semantic Web with SWRL", *Fourth International Semantic Web Conference (ISWC2005), Galway, Ireland,* , pp. 974-986.

Pazienza, M. T., Pennacchiotti, M., Vindigni, M. and Zanzotto, F. M. (2005), "AI/NLP technologies applied to spacecraft mission design", *Proceedings of the 18th International conference on Innovations in Applied Artificial Intelligence.Lecture Notes in Computer Science,* , pp. 239-248.

Pinto, H. S. and Martins, J. P. (July, 2004), "Ontologies: How can They be Built?", *Knowledge and Information Systems,* vol. Volume 6, no. Number 4, pp. 441-464.

Protege Team (2008a), *getting started with protégé-owl*, available at: http://protege.stanford.edu/doc/owl/getting-started.html#projects (accessed 06/15).

Protege Team (2008b), *user documentation*, available at: http://protege.stanford.edu/doc/users.html (accessed 06/15).

Provine, R., Schlenoff, C., Balakirsky, S., Smith, S. and Uschold, M. (2004), "Ontology-based methods for enhancing autonomous vehicle path planning", *Robotics and Autonomous Systems,* vol. 49, no. 1-2, pp. 123-133.

Robert, L., Holger, K., Tracy, S., Gilberto, F., Sherri, d. C. and Noy, N. F. (2005), *Evaluating Protégé OWL as an Editing Environment for NCI Thesaurus*, available at: http://protege.stanford.edu/conference/2005/submissions/posters/poster-lintern.pdf (accessed 06/13).

Ronald J. Brachman and Hector J. Levesque (2004), *Knowledge Representation and Reasoning,* Morgan Kaufmann.

Ruiz-Torres, A. J., Zapata, E., Nakatani, K. and Cowen, M. (2006), "Knowledge based representation and operations assessment of space transportation system architectures", *Knowledge-Based Systems,* vol. 19, no. 7, pp. 516-523.

Sarder, M. B., Ferreira, S., Rogers, J. and Liles, D. H. (5-9 Aug. 2007), "A Methodology for Design Ontology Modeling", *Management of Engineering and Technology, Portland International Center for,* , pp. 1011-1011-1018.

Sarder, M.B. and Ferreira, S., (2007), *Developing Systems Engineering Ontologies*.

Schlenoff, C., Washington, R. and Barbera, T. M.,C. (2005), "A standard intelligent system ontology", *Proc.SPIE,* vol. SPIE-5804, pp. 46-56.

Seidenberg, J. and Rector, A. (2007), "The State of Multi-User Ontology Engineering", Vol. 315, October 28, 2007, .

Seta, K., Ikeda, M., Kakusho, O. and Mizoguchi, R. (1996), "Design of a Conceptual Level Programming Environment Based on Task Ontology", *Proc.of Successes and Failures of Knowledge Based Systems in Real World Applications,* , pp. 11-20.

Sidi, M. J. (2000), *Spacecraft Dynamics and Control: A Practical Engineering Approach,* Cambridge University Press, United States of America.

Smith, M. K., Welty, C. and McGuinness, D. L. (2004), "OWL Web Ontology Language Guide", *W3C Recommendation,* vol. 10.

Storey, M. A., Lintern, R., Ernst, N. A. and Perrin, D. (2004), "Visualization and Protégé", *7th International Protégé Conference,* .

Storey, M. A., Musen, M., Silva, J., Best, C., Ernst, N., Fergerson, R. and Noy, N. (2001), "Jambalaya: Interactive visualization to enhance ontology authoring and knowledge acquisition in Protege", *Workshop on Interactive Tools for Knowledge Capture, K-CAP-2001,* .

Sure, Y., Gomez-Perez, G. P., Daelemans, W., Reinberger, M. L., Guarino, N. and Noy, N. F. (2004), "Why Evaluate Ontology Technologies? Because It Works!", *IEEE Intelligent Systems,* vol. 19, no. 4, pp. 74-81.

The MathWorks (2008), *Medical University of South Carolina Applies Bioinformatics Theory with MathWorks Tools*, available at: https://tagteamdbserver.mathworks.com/ttserverroot/Download/23054_91281V00_MedUnivSC_US.pdf (accessed 06/15).

Verrier, D. (2001), *A Unified framework for spacecraft operations,* PhD and Masters by research theses (School of Engineering), Cranfield University, Cranfield University.

Wertz, J. and Larson, J. (1999), *Space mission analysis and design,* 3rd ed, Microcosm Press; Kluwer Academic Publishers, Torrance,CA; Dordrecht.

APPENDICES

A.1 How to convert to a database project?

It is eventually required to convert the project to a database format if the ontology is big enough (with a few hundreds of classes). Before starting a new project or converting an existing project to a database some prerequisites have to be completed.

1. Get the database software:

The developers of Protégé claim that they support most of the relational databases but the author was successful to use MySQL database. MySQL is open source relational database, very easy to use and relatively fast. A free version is available to download http://dev.mysql.com/downloads/. First of all, it is obvious that MySQL has to be installed in the system and it is required to create a user preferably with password with all accessible privileges. Using this user name and password a new database needs to be created. It is also possible to create a separate table in an existing database. The simple syntax of creating a database is:

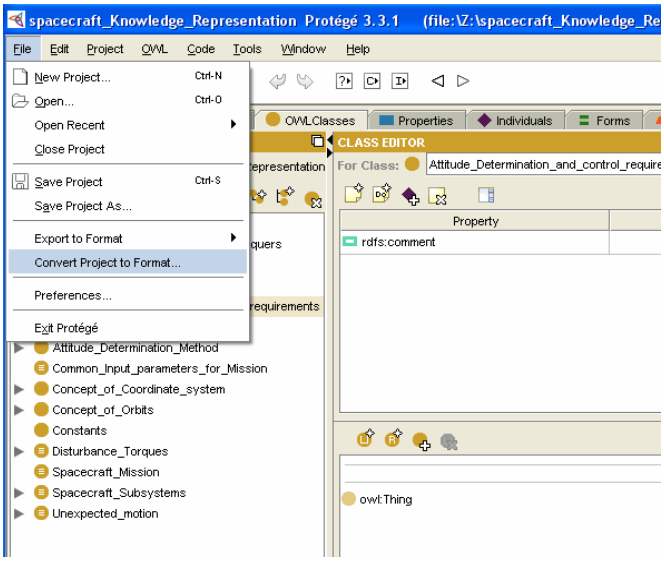
CREATE DATABASE [database name];

2. Save the driver in required location:

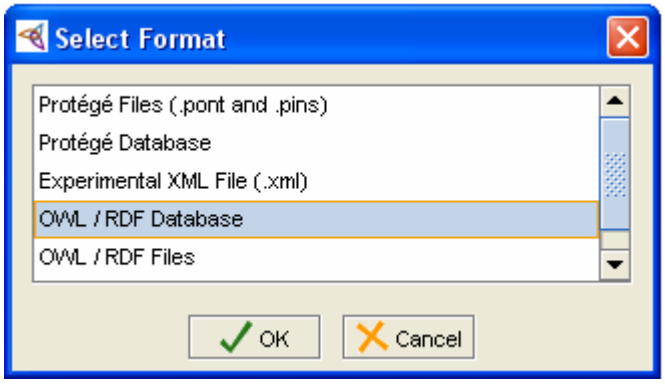
MySQL is compatible with most of the programming languages; as Protégé is written in Java some connector called a driver is required to connect MySQL from the Java environment to talk with each other. The driver is also freely available in the MySQL website and MySQL Connector/J 5.1 is used in this project. The best and easiest way to do it is to install the driver and place the driver in the root directory for Protégé, renaming it to Driver.jar. It is mandatory to rename it, because when a database project starts the Protégé looks for some file with name Driver0.jar, Driver1.jar.

After all these preparation the ontology is ready to be converted into a database project. The steps are as follows:

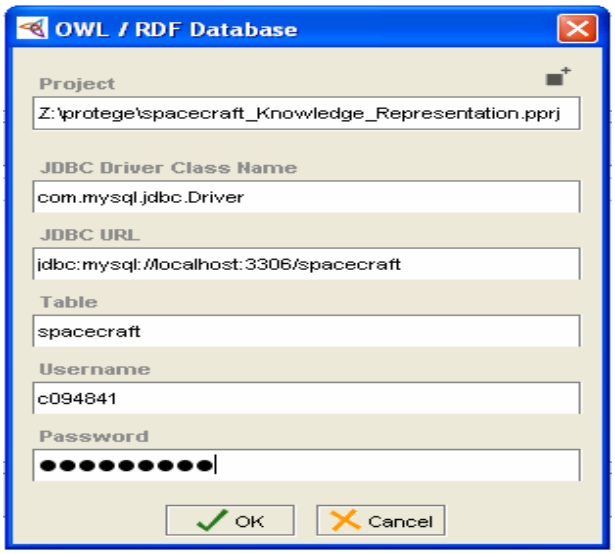
1. In Protégé environment, Select File>Convert Project to Format.



2. Select OWL/RDF Database.



3. Then this following menu will appear and the parameters must be entered.

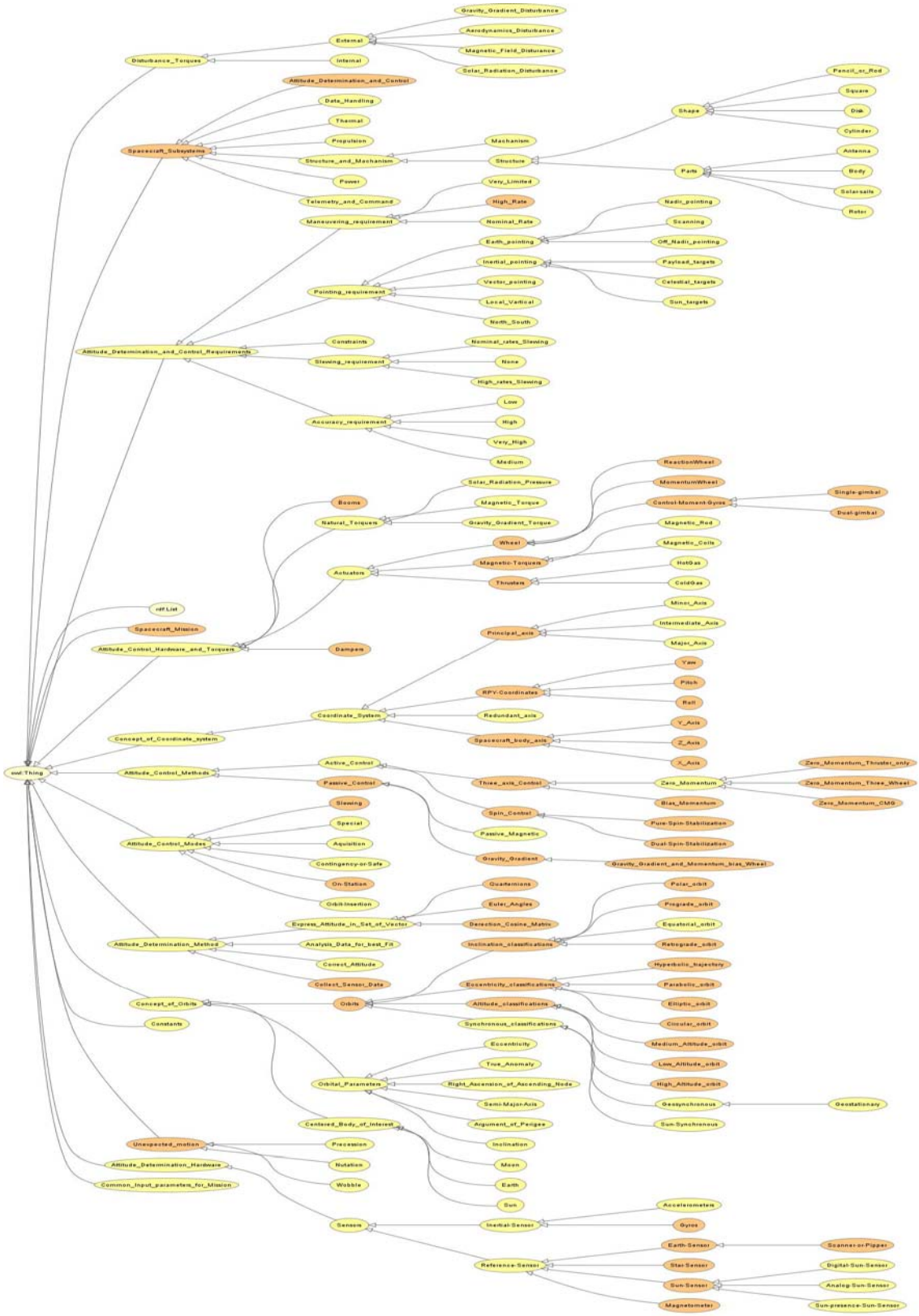


4. Enter the JDBC driver class name ("com.mysql.jdbc.Driver" for MySQL).

5. Enter the JDBC URL (jdbc:mysql://localhost:3306/[name of the database]). The name of the database for this project was spacecraft.
6. Enter the Table name. Anything can be chosen, for this project table name is spacecraft.
7. Enter the Username for the database application that is already created in MySQL.
8. Enter the Password for database application that is already created in MySQL.
9. Click OK

These few steps are required to create, save, and convert the ontology.

B.1 Asserted Hierarchy of ADCS ontology



C.1 SWRL rules

The web document is attached.