Ontological Modeling for Integrated Spacecraft Analysis

Erica Wicks

Mentor: Yu-Wen Tung

Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr., Pasadena, CA 91109

Abstract

Current spacecraft work as a cooperative group of a number of subsystems. Each of these requires modeling software for development, testing, and prediction. It is the goal of my team to create an overarching software architecture called the Integrated Spacecraft Analysis (ISCA) to aid in deploying the discrete subsystems' models. Such a plan has been attempted in the past, and has failed due to the excessive scope of the project. Our goal in this version of ISCA is to use new resources to reduce the scope of the project, including using ontological models to help link the internal interfaces of subsystems' models with the ISCA architecture.

I have created an ontology of functions specific to the modeling system of the navigation system of a spacecraft. The resulting ontology not only links, at an architectural level, language specific instantiations of the modeling system's code, but also is web-viewable and can act as a documentation standard.

This ontology is proof of the concept that ontological modeling can aid in the integration necessary for ISCA to work, and can act as the prototype for future ISCA ontologies.

Table of Contents

Abstract 2
Table of Figures 4
Introduction 5
Introduction to ISCA 5
Earlier Versions 6
MSAS 6
ISCA-1 6
Current ISCA 7
Introduction to Ontologies 7
History of Ontologies 7
Ontology Basics 8
Ontology Languages 10
Ontologies for Information Science 10
Analysis and Design Phase 10
Deployment and Run-time Phase 12
Early Designs 12
Initial Design 12
First Draft 13
Second Draft 13
Problems 14
ISCA Proof of Concept 15
SPICE 15
APGEN 15
Integration Plan 15
Creating the SPICE Function Ontology 16
Viewing the SPICE Function Ontology 18
Using the SPICE Function Ontology 20
Results 21
Future Plans 22
Works Cited 23

Table of Figures

Figure 1: An Example of Two Related Concepts in the Example Domain 8
Figure 2: A second Example of Two Related Concepts in the Example Domain 8
Figure 3: A Class Containing an Element 9
Figure 4: A Class Can Act as a Concept within an Ontology 9
Figure 5: A Partial View of the First Attempt at Creating a Slewtooth Ontology 13
Figure 6: A Full view of a Basic ISCA Ontology with the ACS Subsystem Specified 14
Figure 7: SPICE Functions Converted into Excel Spreadsheet 16
Figure 8: Initial XML Representation of SPICE Function Information 17
Figure 9: HyperGraph Hyperbolic view of the SPICE Function Ontology 19
Figure 10: The XML View of the Treebolic Ontology 20
Figure 11: Treebolic Ontology with Web Interface, Tooltips Visible 20
Figure 12: The Edited Version of the Treebolic Viewer, with Search Function and Tabs 21

Introduction

Current spacecraft work as a cooperative group of a number of subsystems. These subsystems include but are not limited to Navigation, Attitude Control System, Command and Data Handling, Thermal Control, and Telecommunications, which work together to form the spacecraft as we know it. Each of these individual subsystems is a complex structure on its own, however, requiring modeling software for development, testing, and prediction. It is the goal of my team to create an overarching software architecture called the Integrated Spacecraft Analysis (ISCA) to aid in deploying the discrete subsystems' models (1). In order to aid in the development of this software, I have examined the use of ontological modeling to enhance our software architecture and aid in integration.

In the remaining portions of this paper, we will look at ISCA and its development more thoroughly. First, we will examine the history of ISCA in the form of earlier versions and designs, current operations concept, and, most importantly, where the two former differ. Next, we will examine ontologies – their origin, uses in information science, and eventually the benefit that they can give ISCA. Lastly, we will examine a proof-of-concept ontological model useful to ISCA, the procedure used to implement it, and the effectiveness of such a model.

Introduction to ISCA

To understand the need for an Integrated Spacecraft Analysis (ISCA) project, it is first necessary to gain a basic understanding of the role of modeling in spacecraft operation and prediction. It is beyond the scope of this paper to fully flesh out the uses and benefits of modeling components of a spacecraft, nor is it possible in such a brief amount of time to fully explain the complexity of the interactions of the subsystems nor the design and modeling process. Therefore, the following information on these topics is exemplary but by no means exhaustive.

Before sending any commands to a spacecraft, all commands must go through a verification and validation phase where models are used "to predict the effect of proposed plans and sequences on critical…resources" (1). Modeling allows command designers to make sure that commands will not endanger the spacecraft in any way. Once a command has been carried out by the spacecraft, modeling is used to "predict the state of the [spacecraft] in such a way that it can be compared with actual data obtained from downlinked telemetry" (1). We can see, therefore, that spacecraft modeling software is intricately worked into the mission operations of a spacecraft.

Despite the fact that modeling spacecraft subsystems is necessary and often used, there are a number of problems with current modeling system software. As the designer for the initial version of ISCA, Mark Kordon, reports:

> JPL currently uses a variety of homegrown simulation-based spacecraft (S/C) prediction and health assessment simulation models and tools to assist in operating its space vehicles. These applications are often developed in an ad-hoc manner (i.e. do not follow the JPL Software Development Requirements) and are frequently maintained through the initiative of individual engineers on a project-by-project basis. They are usually

tailored for a specific mission and typically require significant modifications by the owner (generally due to lack of documentation) for use on other projects. This often leads to different operations processes using different implementations of the same simulation models (2).

To try to mitigate the effects of these problems, the ISCA project was designed to "provide a better methodology for designing and deploying the [Modeling System] that is available today" (1). However, ISCA itself has gone through a number of design changes since its earliest implementation. In order to understand how we arrived at the design of ISCA today, it is necessary to understand a little more about its predecessors.

Earlier Versions

ISCA developed as a part of a larger, multi-mission design system called the Advanced Multimission Operation System (AMMOS). ISCA is an evolution of the Spacecraft Analysis (SCA) subsystem that was included in the initial design of AMMOS (3). The most notable instances of an attempt at an integrated version of the SCA are MSAS, the initial implementation of ISCA, and the version of ISCA that is currently in development. Descriptions of these various systems follow.

MSAS

Multimission Spacecraft Analysis Subsystem (MSAS) was a proposal in the mid to late 1990s (1) that operated as part of the Advanced Multimission Operating System (AMMOS). A few of the key ways in which MSAS attempted integration among models of the spacecraft's subsystems include:

1. The use of common analysis software reused many times within the system
2. The use of the State Table as a common communication data type…
3. A front panel incorporating the Hewlett Packard Hpvue
4. A common GUI design for all applications
5. A catalogue and catalogue system for storage and retrieval of data
6. A 'Batch Mode' or command line execution capability
7. The use of Application Programming Interfaces (API's) based on abstract data types (4)

This design strategy fed directly into the design for the first implementation of ISCA when the scope of MSAS was determined too large for time and budget constraints (1).

ISCA-1

In 2006, Mark Kordon and his team started working on a different method for creating interoperability between models of the various spacecraft subsystems (2). Their proposal was the original Integrated Spacecraft Analysis (referred to here as ISCA-1). ISCA-1 had the goal of creating a "plug-and-play" framework that could incorporate different subsystems' modeling programs, and could exchange these models for updated ones as needed (5). This integrated framework would have its own Application Programming Interface (API), which would allow a user to initialize a simulation run through all connected models, execute a simulation (including single stepping), set and read data, write log and error files, write checkpoint files, write results files, and end a simulation (6).

Though more limited than its predecessor MSAS in that it did not encompass models for Data Management and Accountability, Activity Planning and Sequencing, or Data Monitoring and Display (2), its scope was wide enough to lead ISCA-1 to fail to deliver on its promises within the allotted budget (1). The reviewers of the operations concept claimed that the scope of ISCA-1 was "beyond what has been attempted by the team, or anyone else at JPL before" (1). With limited money and time, the ISCA-1 task force was attempting a project that the reviewers found beyond the scope of abilities at JPL with funding less than was given to the MSAS team.

Coupled with additional problems that the review team found with the clarity and maturity of the design of ISCA-1 as well as the responsiveness of the requirements, the excessive scope eventually led to the discontinuation of ISCA-1.

Current ISCA

The failures of both MSAS and ISCA-1 did not mean that the goal of integration among the models of the spacecraft's subsystems was not worth working toward, merely that the angle of approach of the earlier versions was flawed. Because the plan for the current instantiation of ISCA is broad and interacts with most of the subsystems of the spacecraft, the plan for its architecture is likewise broad. Because the current ISCA is still in the design phase and currently developing its operations concept, it is my intention here to introduce just the very basic idea of the current ISCA, as well as the ways in which it differs from the previous attempts.

The current implementation plan for ISCA aims "to provide a system-wide, integrated interface to the modeling capabilities needed in Mission Operations" (3). The ISCA team would accomplish this by creating and adopting a Uniform Modeling Interface (UMI) that would allow users to have access to all subsystem models and enable communication between these models. While at initial glance this seems fairly similar to the design of ISCA-1, this new implementation plan takes into account the full complexity of the Mission Operation System and reduces the scope of the project substantially. One of the ways in which the scope of the project would be reduced is by incorporating some mission specific models into our architecture by linking the internal interfaces with the ISCA architecture. Such linking can be implemented by the use of ontological modeling.

Introduction to Ontologies

To understand how an ontology might be useful in the design of ISCA, and the process by which one would be built, it is necessary first to understand where ontologies came from, how they are created, and most importantly how they can be used within the realms of information and computer science.

History of Ontologies

Ontologies in information science are an evolution of a branch of Metaphysics known as Ontology. The name "Ontology" stems from the Ancient Greek words ὀντος (ontos), which is the present participle of the verb "to be", meaning "being" or "that which exists", and λογία (logia), meaning (more-or-less) "study" (7). This philosophical-Ontology is literally the study of existence and reality and defining what each of those means, and perhaps most importantly, for our purposes, the study of the fundamental

divisions of things in the world (7). The concepts of fundamental divisions and defining the domain of existence are what are pulled into the information science version of an ontology[1].

In the middle of the twentieth century, as the study of Artificial Intelligence was coming into its own, researchers began to understand the need for defining a system of knowledge (7). If an Artificially Intelligent system needs as input a knowledge base from which to extrapolate (based on input-rules) additional information or courses of action, then there must needs be a way to define such a knowledge base. The more rigorously one is able to define a knowledge base, the more robust and powerful the A.I. system that can come from it (8).

In 1993, T. Gruber released a paper that is widely credited with offering the first formal definition of an ontology from the computer science perspective (7). Initially, Gruber describes a formally defined knowledge base as a conceptualization. He claims that "[e]very knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly" (8). Based on this definition and use of conceptualizations, Gruber defines an ontology as "an explicit specification of a conceptualization" (8).

Ontology Basics

Simplified from Gruber's formal definition, an ontology is, basically, a set of concepts and a set of relations on those concepts that together define a domain (or knowledge base). For instance, you could define your (very tiny) domain as containing the concepts "Erica", "A car", and "A Buick", and the relations "Is" and "Drives". More complicated rules defining the domain can be built up using these concepts and relations. For instance, we can link "A Buick" and "A car" together with the relationship "Is" to define "A Buick Is A Car".

Figure 1: An Example of Two Related Concepts in the Example Domain

Alternatively, we can link together "Erica" and "A Buick" together with "Drives" to form "Erica Drives A Buick".

Figure 2: A second Example of Two Related Concepts in the Example Domain

[1] There is a natural language problem here when referring to the abstract, non-count noun "Ontology" when it refers to the philosophical study of existence as opposed to the counted, concrete noun "an ontology" when referring to a described knowledge base in the realm of computer science. The count noun "an ontology" is newer and not universally recognized, though it will continue to be used within this paper.

It can be helpful to think of the concepts and their relations as, respectively, the nouns and verbs of a sentence. In these two instances we have created the sentences “A Buick is a car” and “Erica drives a Buick”.

Of course, not all ontologies are as simplistic as the preceding demonstration. Most have sets of relations and concepts that are much larger than the one described, depending on the complexity of the domain needing to be described. The same basic premises apply, however, regardless of the size of the concept and relation sets. In most ontologies, though, the sets of concepts are further divided into a number of classes. These classes contain concepts that have a certain degree of similarity, and can themselves contain further sub-classes. Classes also can act as elements within the ontology, and are able to relate to other concepts via a relation. For example, in the above domain we specified the concepts “A Buick” and “A Car”. Though it might be evident that, in the real world, a Buick is a type of Car, such a relationship has not been formally specified. We can to represent this within the domain by declaring that “A Car” is a class that contains the element “A Buick”.

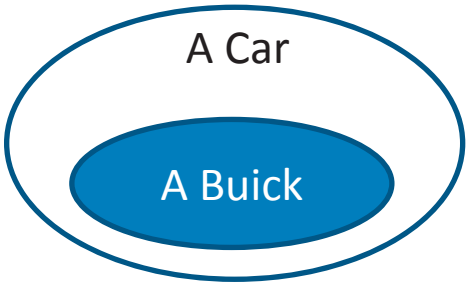

Figure 3: A Class Containing an Element

Implicit in the use of classes within a set of concepts is the relation “Is a” or “Are”. In Figure 3, we have visually denoted that “A Buick” is an element of the class “A Car”, but this can just as easily be said as “A Buick Is A Car”. In essence, we can replace the statement X ∈ Y with X is a Y, using either membership in a class or a formally specified ontology relationship “Is a” as a definition.

It is important to remember that these classes containing concepts can themselves act as concepts, meaning that they can be linked to other concepts via relations. Therefore, while “A Car” is a class, it can still act as a concept in something like “Erica Drives A Car”. This also means that classes containing concepts can aact as members themselves of other classes.

Figure 4: A Class Can Act as a Concept within an Ontology

Of course, there are other levels of complexity that can be included in an ontology. Concepts can have attributes helping to define the concept, and classes within classes can create varying levels of hierarchies within the domain. The set of relations can also come with rules and restrictions limiting

how they can interact with concepts, and the entire domain can come with axioms defining the system. However, the basic idea of a set of concepts and a set of relations defining how they interact can be viewed as a simple outline of an ontology.

Ontology Languages

There are, of course, many different languages with which to write an ontology. Which one is chosen ultimately depends on how the ontology is intended to be used (on the web, in conjunction with existing programming, whether you have a great deal of axioms, etc…)

One of the more popular web-based ontology languages is the Web Ontology Language (OWL[2]). OWL is "a computational logic-based language such that knowledge expressed in OWL can be reasoned with by computer programs either to verify the consistency of that knowledge or to make implicit knowledge explicit" (9). Unpacking this a little, OWL is designed to represent an ontology in a logical (that is, mathematically logical) way that enables artificially intelligent agents and programs to both access the ontology and draw conclusions from it. OWL is also a notoriously web-based ontology language that relies upon (but is not limited to) XML (eXtensible Markup Language (10)), RDF (Resource Description Framework (11)), and RDFS (Resource Description Framework Schema (12)) structures for its syntax. OWL's use of web-languages allows it to "represent machine interpretable content on the web" (13).

One of the most interesting ways in which OWL implements this web-based interoperability is by associating every concept with an International Resource Identifier (IRI), which are generalizations of Uniform Resource Identifiers (URIs) that may contain any Unicode character (14). By associating concepts with IRIs, OWL provides each concept with a unique identifier that can be used or mapped to by many different documents or agents, ensuring a similar definition of each concept among users and collaborators.

Ontologies for Information Science

It turns out that there are myriad uses for ontologies within the realm of information and computer science beyond aiding in defining Artificial Intelligence knowledge bases. Without going into detail on them all, I would like to talk about the uses that have the most impact and import on the task of creating an ontology for ISCA, specifically the use of ontologies during the Analysis and Design and the Deployment and Run Time phases.

Analysis and Design Phase

In looking at the uses of ontological modeling during the Analysis and Design phase, we will examine specifically the uses in requirements engineering, component re-use, and documentation.

Requirements Engineering

One of the big constraints on working on ISCA is the need to incorporate models from most (if not all) of a spacecraft's subsystems. Requiring the design team to become experts within the domain of each subsystem is infeasible. Likewise, being able to accurately model domains that require experts is itself

[2] The creators of OWL thought that the acronym "WOL", though technically correct in representing "Web Ontology Language", was more difficult to remember than "OWL", and so chose the latter as their acronym.

an undertaking. This is not a problem inherent only to the creation of ISCA, but rather a recurring theme in requirements engineering. Traditionally, requirements information is gathered and used in natural language, which leaves us with the problem that natural language is replete with inconsistencies and ambiguities[3] (15). To ensure that we can accurately incorporate all models, a shared formal (i.e. non-natural language) understanding of the domain must be in place.

To this end, we can use an ontology that "describe[s] requirements specification documents…and formally represent[s] requirements knowledge" (16). This allows us to model domains that require expert knowledge without requiring the designers to become experts themselves, and offers the assurance of accuracy not necessarily provided in natural language.

Component Re-use

An interesting characteristic of ISCA is that there have been multiple past attempts at its creation. For us, this means that, while the goals and specifications have changed since past attempts, there exist already the beginnings of a coded implementation. By creating an ontological model that formally describes the function of components at the design phase of this ISCA, we can see (both structurally and visually) instances where older implementations can be re-used. Also notable, we can likewise see instances where any new code we write can be re-used. Component re-use, whether between project implementations or within one, is generally desired since it helps "avoid rework, save money and improve the overall system quality" (16).

Documentation

Additionally, creating an ontology during the design phase of ISCA can lead toward better documentation of the program itself. The initial ISCA Operations Concept claims that most spacecraft modeling software "are usually tailored for a specific mission and typically require significant modifications by the owner…for use on other projects" (2) due to lack of documentation. Maintenance on existing software systems is necessary and is in fact "one of the most dominant activities in Software Engineering" (16). If we model the ISCA architecture using an ontology to formally explicate the relations between subsystems and code, we will have additionally created a documented model of the functionality of our code – one that is easy to evolve for future modifications and uses. Furthermore, the resulting documentation will be programming language independent and readable even by those who know no programming languages.

Such ontological documentation can also lead to benefits in bug-finding, updating, and testing software code. An ontology modeling the way in which the code interacts can lead to better predictions about the location of a bug having an effect on a certain aspect of the program. The same holds true for fixing the bug or updating the code; the ontology allows the programmer to see what other aspects of the code are affected by a certain module. An ontological model can also help in testing the code by generating "basic test cases since they encode domain knowledge in a machine process-able format" (16).

---

[3] Often in field of linguistics, this same concept is referred to as "the richness" of language. Unfortunately, in the realm of requirements engineering such richness can be tolerated only at the cost of accuracy.

Deployment and Run-time Phase

Though most benefits from ontological modeling occur during the analysis and design phase, there are also possible benefits during deployment of the software. This is especially pertinent if one is able to view ISCA as a form of middleware between the user and the various models of the subsystems of the spacecraft. Such "middleware" can become complex when trying to manage different component dependencies (17). An ontology modeling this middleware, powered by formal logic, creates a conceptual model that "may be queried, may foresight required actions…or may be check to avoid inconsistent system configurations" (18) that can be used to mitigate this complexity. Additionally, we can foresee a time when the pre-existing models of various subsystems will change or update. We can ontologically model these updates, and then map the resulting ontology to the old one or into the ISCA ontology itself, limiting the amount of work necessary to incorporate model updates.

Early Designs

The initial idea for using ontological modeling in ISCA was to create a model of an abstract, overarching structure of the various subsystems of a spacecraft and individual domain ontologies of the various subsystems. This would create a shared vocabulary between the subsystems and within the program architecture, allowing information to be shared more easily. Such an ontology would make use of the aforementioned benefits of ontological modeling in the following ways. During the requirements engineering phase, ontological modeling allows us to represent information given to us by domain experts in a rigorous, formal way. During the design of the software, ontological models of the domains allow us to see areas where we can reuse code from prior versions of ISCA or code that we have already written ourselves. During the run-time of our software, we could rely on mappings within the ontology to associate various aspects of the subsystems' models, and eventually use these mappings to incorporate new models. The resulting code would then come with its own ontological documentation, allowing our successors to update or fix the code with ease.

Initial Design

Whereas in previous versions of the ISCA attempts were made to create a "plug-and-play" architecture, fitting different models directly into the ISCA code, our initial design of the ontology would allow us instead to map into ISCA, limiting the amount of necessary code refactoring. This allows for a greater freedom of design in the individual subsystems' models, which is necessary especially as some of the models have already been created and defined.

To demonstrate this capability, we initially decided to model the domain of the Attitude Control System (ACS), a particular subsystem of the spacecraft in charge of keeping and changing the spacecraft's attitude. Currently, ACS is using a modeling system called Slewth, or Slewtooth. Slewtooth was designed to model and predict the attitude behavior of NASA's Dawn spacecraft[4] and monitor for constraint violations (19). Because Slewtooth is, at this point, specific to the spacecraft Dawn, the programmers

[4] Dawn is a deep space mapping mission heading to two asteroids, Vesta and Ceres, in the asteroid belt between Mars and Jupiter. It was launched in September of 2007 and will reach its first target, Vesta, before this paper is finished.

were able to include the actual flight software for that spacecraft. This allows them to reference the actual power steering code used by the Dawn spacecraft for more accurate predictions.

By starting with an ontological model of Slewtooth and the ACS modeling software, we can start to see where individual subsystems' models are dependent on information from other subsystems (like flight software or Navigation output). By creating a set of generalized abstract relations, we can allow for different serializations at these focal points, allowing for greater use of ISCA.

First Draft

My first attempt at modeling Slewtooth was an ontological model of every module, input, output, or mode contained within Slewtooth. Such a model resulted in 25 concept nodes, 4 relations, and 30 propositions (that is "concept-relation-concept" groups) without including Slewtooth's access to the flight software. See Figure 5: A Partial View of the First Attempt at Creating a Slewtooth Ontology to view what such an ontology looked like.

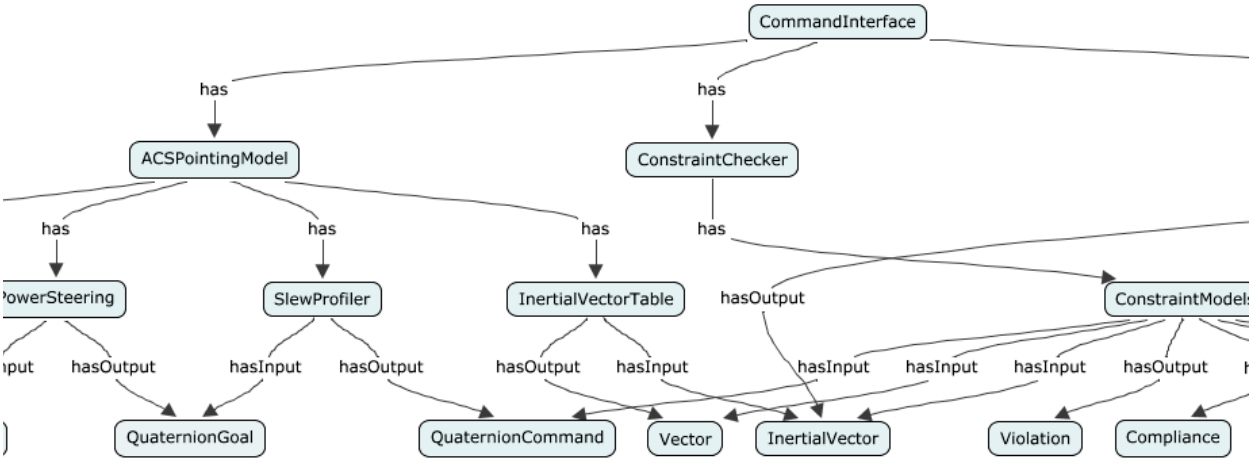

Figure 5: A Partial View of the First Attempt at Creating a Slewtooth Ontology

Such a description of Slewtooth could aid the ISCA team's understanding of Slewtooth itself and could aid in representing Slewtooth within the confines of ISCA. However, while this generalized model of what Slewtooth represents could be applied to more spacecraft than just Dawn, in order to make ISCA sustainable over multiple missions we have to allow for changes within the internal structure of Slewtooth or the abandonment of Slewtooth as a modeler all together. To this end, an ISCA ontology must have a way to mandate necessary inputs into any ACS modeler, any necessary outputs from an ACS modeler, and any points within an ACS modeler where access to information from another subsystem is necessary that is not specific to Sleewtooth.

Second Draft

This led to a second draft of an ontological modeling of Slewtooth or ACS software, which was less complex, though potentially more informative to the ISCA objective. The second draft incorporates the idea of a "black-box" model of Slewtooth, mandating only the inputs, outputs, and connections with

other modeling systems so that Slewtooth itself could be removed or its inner-workings changed without necessarily affecting the model. This new, simpler ontology resulted in 11 concept nodes, 5 relations, and 9 propositions for the Slewtooth portion alone. See Figure 6: A Full view of a Basic ISCA Ontology with the ACS Subsystem Specified to see what this new ontological model looked like.

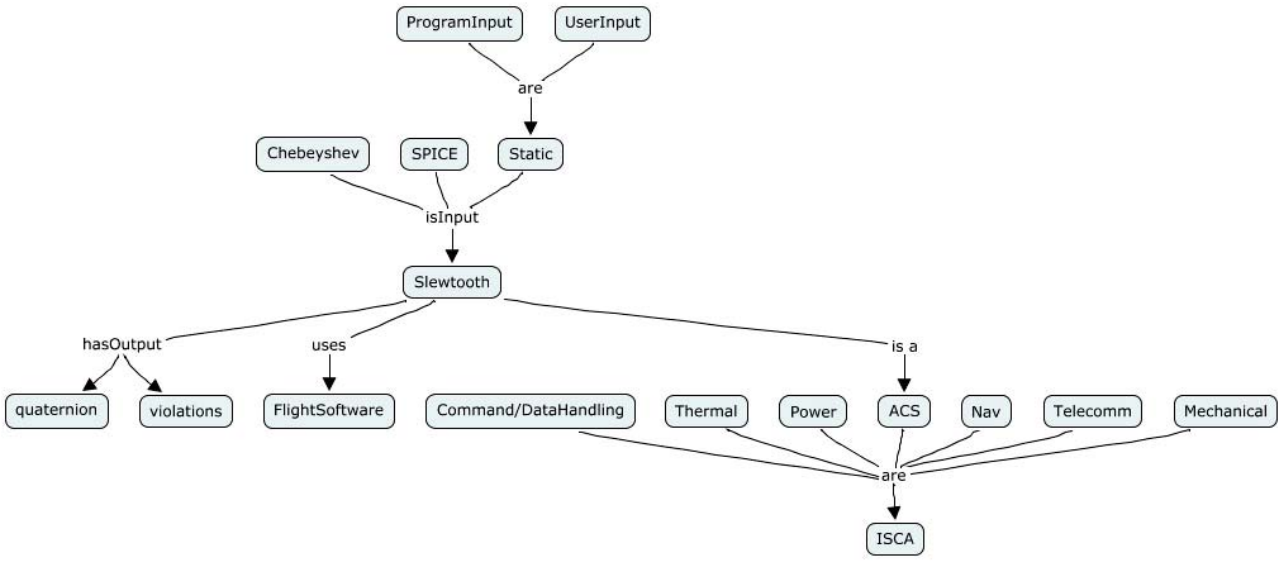

Figure 6: A Full view of a Basic ISCA Ontology with the ACS Subsystem Specified

The new ontology showed only those points where the ACS modeling system might interact with other models or subsystems, including the flight software. In this sense, we are no longer looking at an ontological model of JUST the Slewtooth or ACS domain, but rather a portion of the domain of ISCA itself. A simpler model like this allows us to see the mapping points between ontologies of different subsystems, which will turn, in the future, to the mapping points between the modeling systems of different subsystems.

Problems

Both ontological models of Slewtooth looked promising, and potentially helpful in the overall product of ISCA, but there were some inherent problems with the overall concept of these designs. These draft ontologies were both trying to model the interaction of Slewtooth with the overall spacecraft and the ISCA design, but because there is limited time to the internship and thus our work on this project, we were unable to gain access to the code of Slewtooth to see exactly how it interacted with other applications. Additionally, the new version of ISCA is still in the early stages of design, currently refining its operations concept. Because the actual architectural design of the ISCA UMI is still variable, ontologically modeling the design or ontologically linking to the design seemed dangerous. In order to still demonstrate the usefulness of an ontology within the domain of ISCA, we were forced to focus down the ontology in two ways. The first focus lowered the domain from ontologically modeling the

interaction of an entire subsystem with the ISCA UMI to modeling one aspect of a particular modeling program. The second focus changed the use of the ontology; instead of a high-level, abstract backbone to the entire ISCA system, the new ontology design would allow for web-enhanced, user-level integration. The resulting ontology would stand as a proof of concept of the ideas of integration between subsystems, as well as a proof of the concept of the usefulness of ontologies in software design. These two new foci, along with the new design of the ontology, will be explained in depth below.

ISCA Proof of Concept

The newly refocused version of the ISCA ontology aimed at making the functions of a program called SPICE more accessible to other modeling programs, in particular a program called APGEN. This approach was decided upon after observing users of SPICE and the methods by which they accessed SPICE. To better understand our goals with this ontology, a basic understanding of both SPICE and APGEN is necessary.

SPICE

SPICE is an information system built by the Navigation and Ancillary Information Facility to assist "in planning and interpreting scientific observations from space-borne instruments" (20). A difficult acronym to unpack, SPICE more or less stands for Spacecraft ephemeris; Planet, satellite, comet, or asteroid ephemerides; Instrument description; C-matrix; and Events – though in truth more navigation information than that can be contained within SPICE files. Information held in these SPICE files is used to model the location of the spacecraft in space or the locations of objects in space from the point of view of the spacecraft. SPICE is a standard that is available to the public and used by many different groups and applications.

APGEN

APGEN – or Applications Generator – is a "planning tool component of the Mission Planning & Sequencing Program Element" (20). It allows for visualization of mission scenarios, recognition of constraint violations, support in sequence building and resource and constraint analysis. Users can specify resources – power, fuel, etc. – and events that use those resources, and then plan with respect to time the order that those events occur. By providing APGEN with restraints on resources, APGEN can provide information on when the user's plan violates a resource constraint, as well as report on how long a given plan will take to operate. As an initial input, APGEN can take navigation and location information in the form of SPICE files, and can even call SPICE functions. In talking with proficient APGEN users (21), it became clear that in order to use SPICE functions, individual function wrappers that allowed the functions to be called from APGEN had to be created in user-defined files for even limited functionality.

Integration Plan

We decided to create an ontological model of the SPICE functions based on C-SPICE, or the C version of the SPICE functions, which is the most often used format. However, within the ontology each function

would be linked to its Fortran, Matlab, and IDL function counterpart. In this sense, the SPICE ontology would be populated with the abstract SPICE functions, and each of these nodes would be associated with up to four different instantiations of the functions in the different programming languages. This allows users of SPICE to access the same ontology for documentation of the SPICE functions in any language.

However, this does not solve the problem of needing to wrap the functions in order to use them from APGEN. To address this problem, the goal was to associate wrappers with each function node in the ontology. Eventually this proved to be unfeasible given the limited time, so instead pseudocode was attached to each function that, when implemented, would convert data types and functions into calls to an xmlrpc server version of the SPICE functions. This would allow users to call SPICE functions more easily from other applications.

Creating the SPICE Function Ontology

To create the ontology of the SPICE Functions, I took the information compiled in the CSPICE API Function Guide (22) and converted it into a comma-separated file. I then manually unembedded the URLs, added the keywords and the inputs, outputs, and parameters associated with each Function, and saved the resulting file as an Excel spreadsheet. I treated this file as my base; because most current users of SPICE use the C-SPICE version, it seemed that the SPICE ontology itself would be most usable if based on the C-SPICE version. To this, I added the function procedures for the corresponding version of each function in C, Fortran, Matlab, and IDL.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Name	Description	URL	Keywords	Input	Output	Input/Out	Paramete	Num	Nun	Nu	Procedure
2	APPNDC_	Append an	http://na	Cells	item: The	item to ap	cell: The cell to whic		2	1	0	void appndc_c(
3	APPNDD	Append an	http://na	Cells	item: The	item to ap	cell: The cell to whic		2	1	0	void appndd_c
4	APPNDI_	Append an	http://na	Cells	item: The	item to ap	cell: The cell to whic		2	1	0	void appndi_c (
5	AXISAR_	Axis and an	http://na	Matrix, Rot	axis: Rota	r: Rotation matrix correspondi			2	1	0	void axisar_c ((
6	B1900_C	Besselian D	http://na	Constants		The function returns the Julian			0	1	0	SpiceDouble b:
7	B1950_C	Besselian D	http://na	Constants		The function returns the Julian			0	1	0	SpiceDouble b:
8	BADKPV_	Bad Kernel	http://na	Error	caller: Na	SPICEFALSE: If the kernel pool			6	1	0	SpiceBoolean b
9	BODC2N_	Body ID cod	http://na	Body, Conv	code: Inte	name: A common name for the			2	2	0	void bodc2n_c
10	BODC2S_	Body ID cod	http://na	Body, Conv	code: Inte	name: String corresponding to			2	1	0	void bodc2s_c (
11	BODDEF_	Body name,	http://na	Body, Conv	name: Common name of some body, cod				2	0	0	void boddef_c
12	BODFND	Find values	http://na	Constants	body: ID c	SPICETRUE: If the item is in the			2	1	0	SpiceBoolean b
13	BODN2C_	Body name	http://na	Body, Conv	name: Bo	code: SPICE integer ID code for			1	2	0	void bodn2c_c
14	BODS2C_	Body string	http://na	Body, Conv	name: Str	code: Integer ID code correspo			1	2	0	void bods2c_c (
15	BODVAR	Return valu	http://na	Constants	body: ID c	dim: Number of values returne			2	2	0	void bodvar_c (
16	BODVCD	Return d.p.	http://na	Constants	bodynm:	dim: Number of values returne			2	2	0	void bodvcd_c
17	BODVRD	Return d.p.	http://na	Constants	bodyid: B	dim: Number of values returne			3	2	0	void bodvrd_c
18	BRCKTD_	Bracket a d.	http://naif.jpl.nasa.g		number:	Number to be bracketed, end1:			3	1	0	SpiceDouble br
19	BRCKTI_	Bracket an i	http://naif.jpl.nasa.g		number:	Number to be bracketed, end1:			3	1	0	SpiceInt brcktl_
20	BSCHOC_	Binary searc	http://na	Array, Sear	value: Ke	The function returns the index			5	1	0	SpiceInt bschoc
21	BSCHOI_	Binary searc	http://na	Array, Sear	value: Va	The function returns the index			4	1	0	SpiceInt bschoi
22	BSRCHC	Binary searc	http://na	Array, Sear	value: Ke	The function returns the index			4	1	0	SpiceInt bsrchc

Figure 7: SPICE Functions Converted into Excel Spreadsheet

The resulting data needed some preprocessing before it could be converted to a usable ontology format. Some keywords associated with the functions were present in both plural and singular forms (i.e., “Ellipse” and “Ellipses”). The decision was made to go with the singular form except in those cases where a plural keyword matched up with a section of the C-SPICE required reading documents (23). Those keywords could still be easily linked with the corresponding required reading documents, but now functions with a keyword “Ellipse” would be grouped with those associated a keyword “Ellipses”.

Additionally, there was not always a direct isomorphism between functions in one language with those in another. For instance, the decision was made to base the ontology on the C-SPICE functions, as C-SPICE is the most often used form of SPICE currently, though originally SPICE was designed for Fortran. There are 1008 Fortran SPICE functions, but only 507 C-SPICE functions. On top of that, there are 37 C-SPICE functions that do not correspond to any Fortran function. The Matlab version of SPICE (called MICE) has only 179 functions; however, 12 of these correspond to no C-SPICE function[5]. Only in Icy, the IDL version of SPICE, does every function correspond with a C-SPICE function (though, because there are only 370 function in Icy, the converse is not true). Correctly associating the different versions of each function nrequired manually matching and checking each function.

After preprocessing the data, I then used <oXygen> XML (24) editor to automatically translate the spreadsheet into an XML document containing each functions name, description, URL, keywords, number of inputs, number of outputs, and number of parameters, input and output data types, and the C, Fortran, Matlab, and IDL procedures.

```
<Function>
    <Name>REPMCT_C</Name>
    <Description>Replace marker with cardinal text</Description>
    <URL> http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/repmct_c.html</URL>
    <Keywords>Character, Conversion, String</Keywords>
    <Input>in:  Input string, marker:  Marker to be replaced, value:  Replacement valu
    <Output>out:  Output string</Output>
    <Input_Output></Input_Output>
    <Parameters>MAXLCN:  is the maximum expected length of any cardinal text</Paramete
    <NumInputs>5</NumInputs>
    <NumOutputs>1</NumOutputs>
    <NumParameters>1</NumParameters>
    <Procedure>void repmct_c (ConstSpiceChar *in, ConstSpiceChar *marker, SpiceInt val
</Function>
```

Figure 8: Initial XML Representation of SPICE Function Information

To create a viable OWL/RDF version of the C-SPICE ontology, I wrote an XSL stylesheet to convert the XML version of the SPICE Functions into OWL/RDF. This stylesheet needed to convert each Function into an OWL class, create OWL classes out of each Keyword and output data types, and associate each Function/OWL class with the correct keyword and output data type parent classes. This resulted in 62 Keyword-based classes, an additional class for those functions without keywords, 7 output data type classes, 507 leaf function nodes, and an additional 9 Data Type leaf nodes.

While manipulating the XML version of the ontology, we also added in additional information to make the ontology more robust and more usable. We included nodes that defined the SPICE data types SpiceInt, SpiceBoolean, SpiceChar, SpiceChar *, Array, Enum, and SpiceCell, and, most importantly, offered users pseudocode for converting these SPICE data types, using either c++ or Java, into XMLRPC

[5] Generally, these MICE functions that do not correspond to a C-SPICE function are special Matlab duplicates of C-SPICE functions that already exist in the MICE library.

calls. I also added label and content information, as well as links to parent nodes (where applicable), leading to required reading documents.

Viewing the SPICE Function Ontology

A side effect of this enforced hierarchy based on Keywords and Output data type associated with each SPICE function is that often nodes belonged to multiple parent classes. In fact, it is rare that a function-node is not associated with multiple parents, the average being about 2.5 parents to each child. This means that each of the 507 children will have Multiple connections leading away from it. While this is not necessarily a problem for an ontology, it gives the resulting model a less-taxonomic appearance that is neither immediately appealing nor parsable to the human mind.

Because of the issue of multiple parent classes for each function-node, I determined that the best option for viewing the ontology would be to use a hyperbolic viewer. A hyperbolic viewer displays tree-structured hierarchies on a non-Euclidean, Poincarean plane that can re-center on any selected node (25). Such a viewer would allow us to show the vast amount of leaf function nodes from the ontology, as well as the interconnectedness between parent classes and each child function, with limited confusion. Initial concepts for this design were influenced by the display used by MultiTree: A Digital Library of Language Relationship (26), which uses a hyperbolic viewer to display large amounts of genetic language relationship data.

Initially, I chose a hyperbolic viewer called Hypergraph (27) to model the ontology because of its ease in producing Java applets and its open source nature. Hypergraph cannot read OWL/RDF, however, nor will it read straight XML; instead, it relies on XML that is compliant with the GraphXML format. GraphXML is an interchange format designed to support pure, mathematical graph descriptions as well as the needs of informational visualization applications (28). To create a GraphXML format of the spice function ontology, I used a second stylesheet, this time designed to convert from the OWL/RDF version of the ontology into GraphXML.

However, despite the use of a hyperbolic viewer, the resulting ontology view was still chaotic and lacked any clarity necessary to be used for documentation and learning purposes. Most of this chaos was due to the large number of leaf nodes and having multiple parents for each child, creating a complex, large, web-structure.

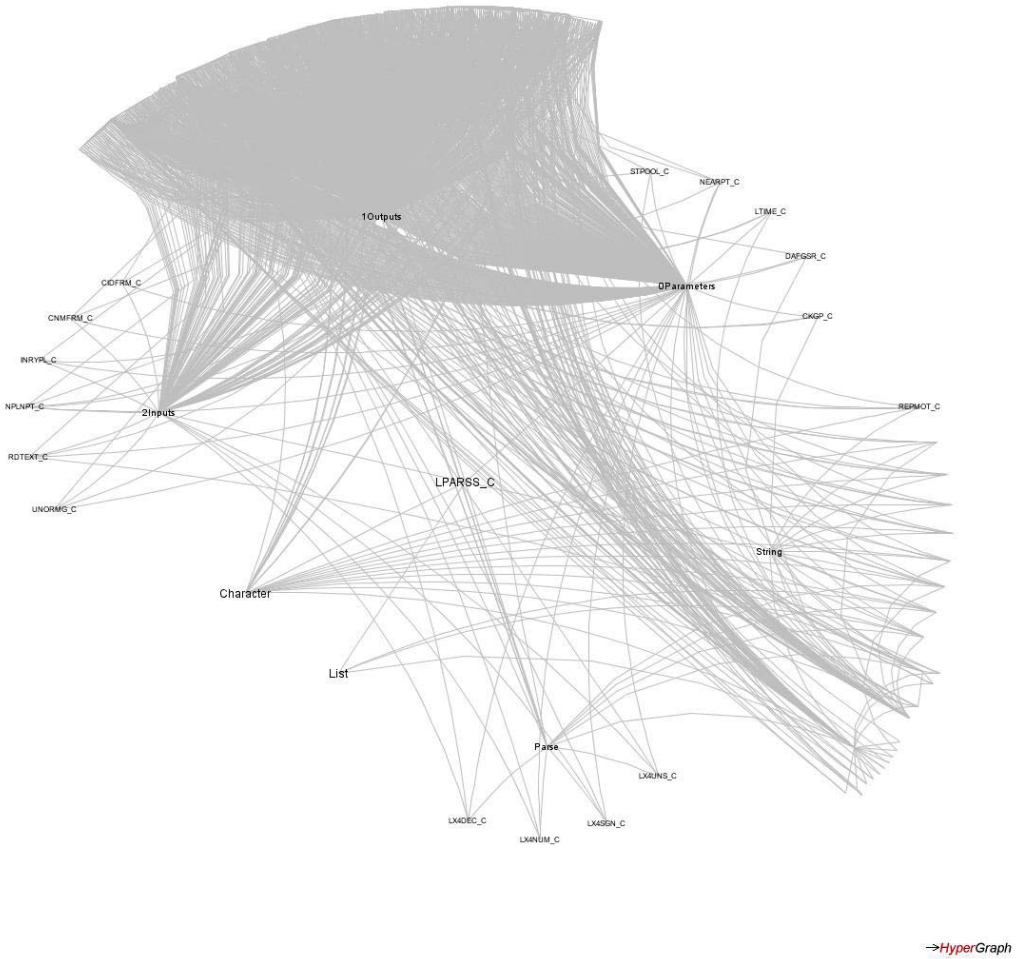

Figure 9: HyperGraph Hyperbolic view of the SPICE Function Ontology

Because of the lack of clarity and therefore lack of usefulness, my co-intern and I decided to find an alternative hyperbolic viewer. We decided on using Treebolic, a similar open source hyperbolic viewer that allowed for more editing of visualization code and that used a different, clearer visual style (29). In addition to changing hyperbolic viewers, we decided to clone leaf function nodes. In the earlier, Hypergraph version of the ontology, each function was linked to multiple parent classes (keywords, input number, etc.). In the Treebolic version, each parent was allowed to create its own version of the function children in order to limit the number of edges and aid in clarity. This meant that the original 507 functions now produced 1367 leaf nodes.

However, Treebolic does not read OWL/RDF ontologies, XML, or even GraphXML versions. I once again wrote a stylesheet to create the "Treebolic" version of the ontology, complete with cloned nodes. Each node comes equipped with tool tips outlining the function procedure and detailed inputs, outputs, and parameters, as well as a link that opens up the online documentation page associated with each function.

```
<node backcolor="006666" id="_Keywords">
   <label>Keywords</label>
   <a href="http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/"/>
   <node backcolor="cc0000" id="_Cells">
      <label>Cells</label>
      <a href="http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/cells.html" target="a"/>
      <node backcolor="3366ff" id="_APPNDC_C-Cells">
         <label>APPNDC_C</label>
         <content>&lt;b&gt;Procedure:  &lt;/b&gt;void appndc_c(ConstSpiceChar *item, SpiceCell *cell)
                  &lt;b&gt;Inputs:  &lt;/b&gt;item:  The item to append
                  &lt;b&gt;Input/Outputs:  &lt;/b&gt;cell:  The cell to which item will be appended</content>
         <a href="http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/appndc_c.html" target="a"/>
      </node>
```

Figure 10: The XML View of the Treebolic Ontology

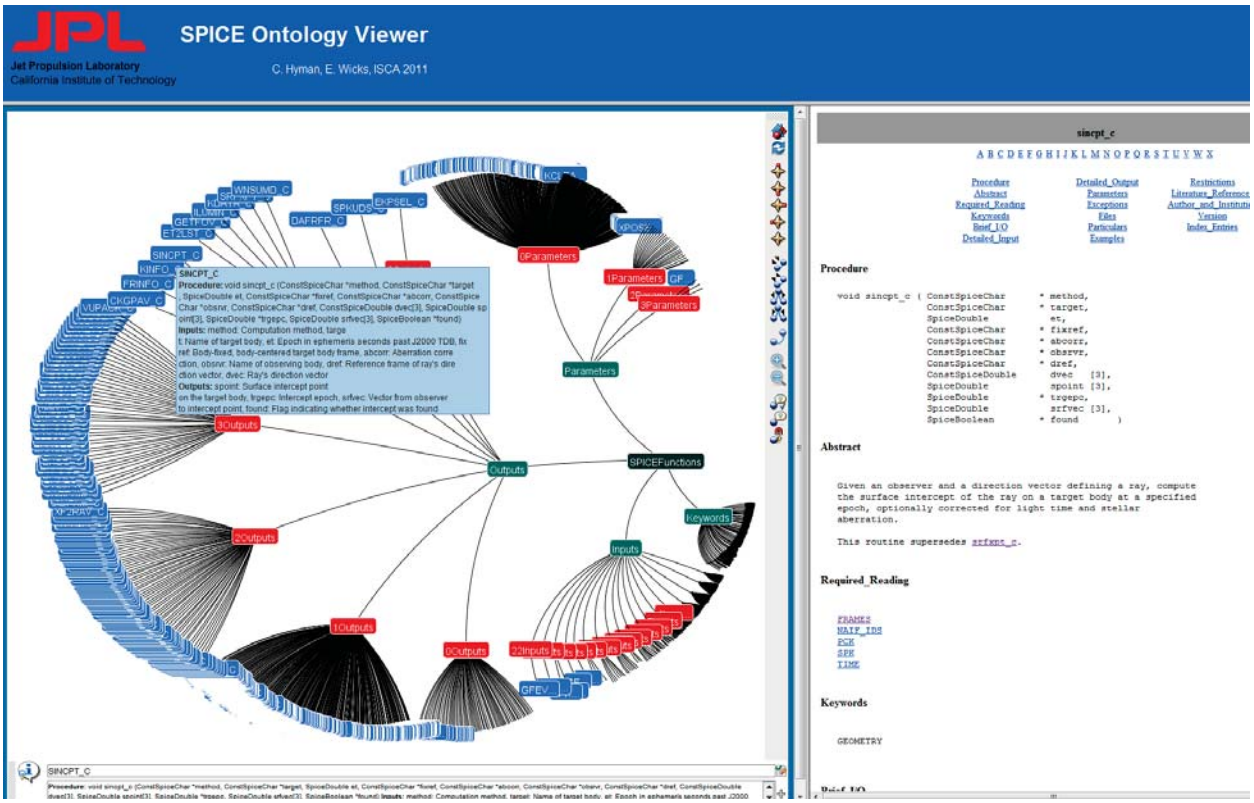

Figure 11: Treebolic Ontology with Web Interface, Tooltips Visible

Using the SPICE Function Ontology

The Treebolic visualization of the ontology allowed us to display every SPICE function, as well as additional documentation associated with each node. However, the functionality was still limited using an unedited version of Treebolic. My co-intern edited the Treebolic code to include tabbed documentation viewing, the option to output the procedure of a selected function, the ability to choose the output language, and robust search functionality. The ontology searches through the description and label for each function, and returns all the functions that contain the search query. Clicking on any

returned function will automatically center the hyperbolic viewer on that particular node, allowing for immediate visualization of the search query as well as easy access to the documentation and usability encoded into each node.

As a result of this added functionality to the interface, users can now use the SPICE function ontology to search for relevant or useful functions, output the necessary function calls in any available language, quickly generate code to convert data types, or simply as a way of viewing and learning the SPICE functions.

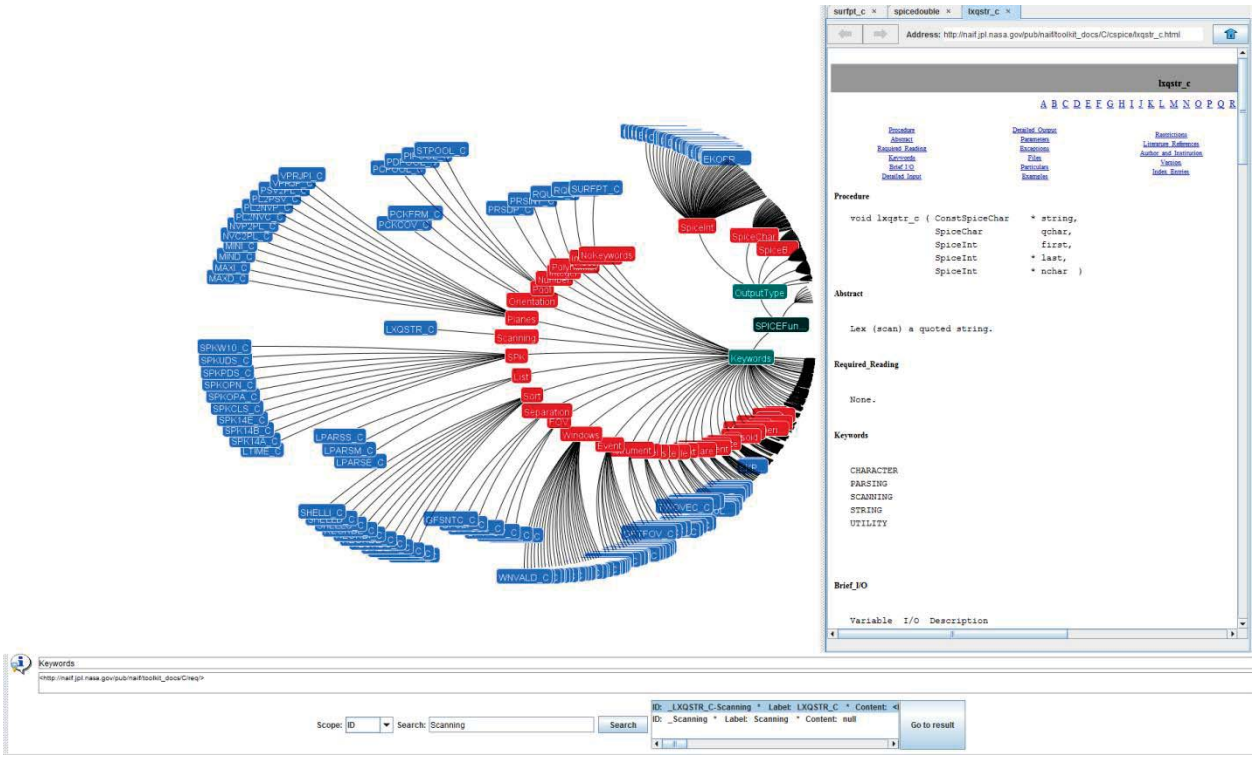

Figure 12: The Edited Version of the Treebolic Viewer, with Search Function and Tabs

Results

The proof-of-concept ontology that we were able to produce contained the entire library of C-SPICE functions, associated with their Fortran, Matlab, and IDL counterparts, as well as the pseudocode that would make them more usable from other applications, such as APGEN. Because the SPICE functions are not necessarily hierarchical, a hierarchy was forced on them that separated the functions by keywords associated with each function in the documentation as well as output type. This means that functions associated with multiple keywords or output types were repeated, to keep visualization clear and simple. The original 507 C-SPICE functions became a set of 1367 function nodes, with 63 parent keyword classes and 7 parent output-type classes. Each function node contained information about the function as well as a link to the corresponding documentation webpages. An interface was developed that allowed the ontology to be viewed in a hyperbolic tree viewer (see Figure 11: Treebolic Ontology

with Web Interface, Tooltips Visible in the Methods section to view what this looks like). The interface also included a robust search functionality, in-window viewing of documentation pages, as well as automatic output of the selected function in the programming language of the user's choice (including pseudocode.)

The resulting ontology also demonstrates many of the aforementioned uses for ontological modeling in information science. The SPICE function ontology acts as a way of modeling the SPICE knowledge domain in a way that is easy to use, even by those users who are not experts in SPICE. Users can easily search the entire ontology for words of phrases that appear anywhere within the documentation pages for each function, all from a single webpage. By associating information on input and output data types with each function, the ontology can provide the correct pseudocode for translating each data type. In this way, the pseudocode is only input once into the ontology but is reused repeatedly for each function where it is applicable. The ontology also acts as a new standard of documentation for the SPICE functions; instead of separate html files for each function as it appears in each different programming language, all of the information is held in one place.

Future Plans

While we believe that the SPICE ontology and interface acts as a decent proof-of-concept of the usefulness of ontologies in integration of spacecraft models, we have also noted some specific places where the design may be improved upon, many of which are usability areas that we had intended to implement. The first of these is the inclusion of actual wrappers for the SPICE functions. It would be ideal if users were able to use our ontology to find a SPICE function and output said function completely wrapped for the program that they were using, and, while this had been our intention, such usability remains undelivered. Instead, pseudocode is provided for only the C version of the SPICE functions that would allow users to, if not immediately than at least with less effort, create the version of the functions that they need. However, adding actual function wrappers as an output from the ontology, for all four variations of the SPICE functions, would add an incredible amount of usability to the ontology and further aid in integration measures.

Another area of improvement is in the very documentation of the ontology itself. While the SPICE ontology may be viewed as a form of documentation for SPICE functions – storing information on function descriptions and components, as well as associating versions of the same function across programming languages – there is little documentation on how to use the ontology and its interface. If the SPICE function ontology were more than a proof-of-concept, if it were being used as actual documentation for SPICE or available to the public, such documentation would need to exist.

Additionally, after speaking with the group currently in charge of SPICE, it was proposed that an ontology of SPICE concepts would also by useful. The SPICE concept ontology would document the nouns used in SPICE – everything from ideas like "ephemeris" to examples of ephemerides. This ontology could be linked at the root to the SPICE function ontology and extend the amount of the knowledge domain of SPICE that the ontology is able to cover, which would in turn allow for more robust documentation and better and more thorough usability. The initial problem with the creation of

this new ontology is the data source for SPICE concepts. Whereas with the SPICE function ontology the initial list of functions was already created, no such list of SPICE concepts currently exists. However, if SPICE concepts could be compiled, then created an ontology model of them and linking the new ontology with the already existing one would prove not only relatively simple, with the creation and viewing methods already in place, but also markedly helpful.

However, despite the noted areas of improvement, the SPICE function ontology is a working example of the benefits of using ontological modeling for integration. The ontology demonstrated areas of code re-use, helped define a knowledge base without expert level knowledge, and resulted in rigorous documentation of SPICE. I believe that a similar use of ontologies can aid in the implementation of the new version of ISCA.

Acknowledgement

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Space Grant Program and the National Aeronautics and Space Administration.

Works Cited

1. **Maldague, Pierre F. and Tung, Yu-Wen.** *Integrated Spacecraft Analysis Operations Concept.* 2011.

2. **Kordon, Mark.** *Integrated Spacecraft Analysis Operations Concept.* 2007.

3. **Maldague, Pierre and Tung, Yu-Wen.** *Integrated Spacecraft Analysis Operations Concept Version 0.5.* 2011.

4. **Hill, Michael H.** *Spacecraft Analysis, MSAS -- A Multi-Mission Solution.*

5. **Kordon, Mark A.** *Integrated Spacecraft Analysis High Level Architectures.* 2007.

6. **Kordon, Mark.** *Integrated Spacecraft Analysis Application Program Interface (API) Software Interface Specification.* 2007.

7. **Gruber, Tom.** Ontology Definition. *tomgruber.org.* [Online] 2007. [Cited: June 22, 2011.] http://tomgruber.org/writing/ontology-definition-2007.htm.

8. —. Toward Priciples for the Design of Ontologies Used for Knowledge Sharing. *tomgruber.org.* [Online] August 23, 1993. [Cited: June 22, 2011.] http://tomgruber.org/writing/onto-design.pdf.

9. **Hitzler, Pascal, et al., et al.** OWL 2 Web Ontology Language Primer. *W3C.* [Online] October 27, 2009. [Cited: June 27, 2011.] http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.

10. Extensible Markup Language (XML). *W3C.* [Online] April 23, 2011. [Cited: June 27, 2011.] http://www.w3.org/XML/.

11. **Klyne, Graham and Carroll, Jeremy J.** Resource Description Framework (RDF): Concepts and Abstract Syntax. *W3C.* [Online] November 8, 2002. [Cited: June 27, 2011.] http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/.

12. **Brickley, dan and Guha, R.V.** RDF Vocabulary Description Language 1.0: RDF Schema. *W3C.* [Online] November 12, 2002. [Cited: June 27, 2011.] http://www.w3.org/TR/2002/WD-rdf-schema-20021112/.

13. **McGuinness, Deborah L. and van Harmelen, Frank.** OWL Web Ontology Language Overview. *W3C.* [Online] February 10, 2004. [Cited: June 27, 2011.] http://www.w3.org/TR/2004/REC-owl-features-20040210/.

14. **Duerst, M.** Internationalized Resource Identifiers (IRIs). *W3C.* [Online] May 31, 2005. [Cited: June 27, 2011.] http://www.w3.org/International/iri-edit/draft-duerst-iri.html.

15. *The Use of Ontologies as a Backbone for Use Case Management.* **Deridder, Dirk, Wouters, Bart and Van Paesschen, Ellen.**

16. *Applications of Ontologies in Software Engineering.* **Happel, Hans-Jorg and Seedorf, Stefan.**

17. *Developing and Managing Software Components in an Ontology-based Application Server.* **Oberle, Daniel, et al., et al.** Karlsruhe, Germany : s.n.

18. *Semantic Management of Middleware.* **Oberle, Daniel.** Karlsruhe, Germany : s.n.

19. **Vanelli, C. Anthony, Swenka, Edward and Smith, Brett.** *Verification of Pointing Constraints for the Dawn Spacecraft.* Honolulu, Hawaii : s.n., 2008.

20. NAIF. [Online] 02 09, 2009. [Cited: 15 07, 2011.] http://naif.jpl.nasa.gov/naif/spiceconcept.html.

21. **Wissler, Steven.** *EPOXI Spacecraft Team Chief.* Pasadena, July 12, 2011.

22. CSPICE API Reference Guide. *NAIF.* [Online] June 9, 2010. [Cited: July 12, 2011.] http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/index.html.

23. SPICE Required Reading Documents. *NAIF.* [Online] [Cited: July 25, 2011.] http://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/req/.

24. <oXygen/> XML Editor. [Online] Syncro Soft. [Cited: June 17, 2011.] www.oxygenxml.com.

25. Hyperbolic Trees. *InfoVis Cyberinfrastructure.* [Online] May 13, 2004. [Cited: July 25, 2011.] http://iv.slis.indiana.edu/sw/hyptree.html.

26. **List, LINGUIST.** MultiTree. [Online] [Cited: July 15, 2011.] multitree.org.

27. Hypergraph. [Online] 2003. [Cited: July 16, 2011.] http://hypergraph.sourceforge.net/index.html.

28. *GraphXML -- An XML-based graph description format.* **Herman, I. and Marshall, M.S.** Amsterdam : s.n.

29. **Bou, Bernard.** Treebolic2. *sourceforge.net.* [Online] [Cited: July 21, 2011.] http://treebolic.sourceforge.net/en/index.html.

30. **Protégé .** Welcome to Protégé . *Protégé .* [Online] 2011. [Cited: June 27, 2011.] protege.stanford.edu.

31. **Knublauch, Holger.** An AI Tool for the Real World. *JavaWorld.* [Online] June 20, 2003. [Cited: June 27, 2011.] http://www.javaworld.com/javaworld/jw-06-2003/jw-0620-protege.html.

32. **Florida Institute for Human and Machine Cognition.** COE. *Cmap tools COE.* [Online] [Cited: June 27, 2011.]http://www.ihmc.us/sandbox/groups/coe/wiki/welcome/attachments/d2a1b/COEmanual06.pdf?sessionID=af02061dea8c960803bd175ae2e54d5ea2b1ec42.

33. **Hayes, Pat, et al., et al.** COE: Tools for Collaborative Ontology Development and Reuse. *COE.* [Online] [Cited: June 27, 2011.] http://www.ihmc.us/sandbox/groups/coe/wiki/f8c65/attachments/59fa9/HayesCOE.pdf?sessionID=af02061dea8c960803bd175ae2e54d5ea2b1ec42.