The OWL and the Architect

A Framework for Knowledge Based Systems

Formalizing REA and Built Upon OWL and SCA

Thomas A. Tinsley

The OWL and the Architect: A Framework for Knowledge Based Systems Formalizing REA and Built Upon OWL and SCA / Thomas A. Tinsley, - 1st edition.

ISBN-13: 978-1519586032

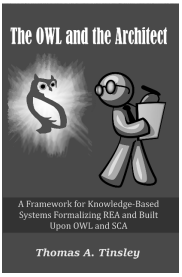
ISBN-10: 1519586035

1. Semantic Web
2. Enterprise Architecture
3. Ontology
4. Information Technology

Library of Congress Control Number: 2015919936

Chief Editor: **Rod Tinsley**
Cover by: **Rod Tinsley**
Produced by: **Tinsley Innovations**

Cover: The Owl on the cover is the symbol for the OWL language used by the W3C. The image has been flipped horizontally so the owl can look directly at the architect.



The architect is looking at OWL to gain knowledge, and he records the enlightenment emanating from the OWL on his tablet.

Forward

All throughout the universe of Information Technology, the Architect found structure and logic. All except in one uncharted sector. Everywhere he looked in this sector he saw **What** and **Where**, but there was no sign of **Why**. It wasn't until the wisdom of the Owl shone forth that the important **Why** was revealed.

I'm the architect whose visit from the Owl led him to investigate information structure. At first, I only found technical specifications embodied in database technology. Going deeper into my research, I found that wisdom and logic had been brought together in a language called OWL. In this language, philosophy and mathematical proof had gelled and brought wisdom and understanding to computers in the Semantic Web.

After an introductory time with OWL, I found simplicity and quality derived from knowing **Why**. After further research and completing a prototype project to prove to myself that what I had learned could actually work for all automated systems, I wrote this book to document the outcomes.

This book presents a proven approach to solving one of today's most significant and costly issues in utilizing computer systems and linking them together in an ecosystem. If you run a business, work in a business, investigate the world we live in, or just want to learn how and why things work, your future is dependent upon this new direction for Information Technology.

We pride ourselves on our knowledge. It gives each of us our unique view of the world and provides a basis for making decisions. Starting

at a very early age, we attend school to absorb the knowledge collected by our ancestors, and to gain the tools that allow us to communicate and share ideas. It takes each of us years to absorb the necessary knowledge to work in the world today. The more professional an individual is, the greater amount of education is required for them to perform their various tasks and responsibilities.

Our computer systems, on the other hand, do not attend school. They are entirely dependent upon the ability of programmers to infuse them with knowledge and understanding. This approach to computer learning has worked for decades, but now the demand for bigger and better computer systems has become so complex that large systems require years to develop, often fail when deployed, and once in production they continue to have significant deficiencies.

The basic problem is the complexity of these systems. With the latest advances in computer technology, we have huge amounts of storage available, unbelievably fast processing power, and incredible communications speed. At the same time, our current generation finds computers are critical to their daily lives, and their expectations of what computers can and should provide continue to grow. This is leading to a demand for complex systems that go well beyond just managing accounts and sharing information. The customers are ready for a revolutionary advance, and the technical resources exist to build one, but the architectural approaches are currently unable to deliver anything but more of the same old same old.

The World Wide Web Consortium (W3C) recognizes the need to capture and utilize knowledge that computers could understand. They refer to this area as the Semantic Web. The languages of RDF (Resource Description Framework) and OWL (Web Ontology Language) have been developed within this subject area, and they have also defined other languages which extend the use of the knowledge captured.

International interest is being shown by the large number of conferences held throughout the world to advance the application of knowledge to information. Far from just theoretical review, they address the specific and current uses of knowledge systems dealing with today's

most demanding information technology challenges.

Knowledge-based systems are the new approach to the mystifying complexity of networked servers. Major organizations working with complex information are enthusiastically embracing this concept. These organizations include health-care, medicine, astronomy, meteorology, and government.

Each utilizing their own existing standards, these organizations have all developed their own unique frameworks for systems construction. Unfortunately, having separate framework solutions results in difficulties in sharing information. It also becomes a barrier for individuals and smaller organizations that cannot justify the expense of developing their own framework. What is needed is a common framework.

This book describes that approach and describes a prototype of a common framework. It is based upon OWL for sharing knowledge, and Service Component Architecture for sharing processes. It takes full advantage of the quality checking that is part of capturing information, one which has proven accurate through layered knowledge of the decades. It also introduces, describes, and shows how a common socket component can be used for networked communications in an ecosystem, where humans and servers act as nodes within the network. It is further proven by running a simulation of multiple pizza stores based upon the REA accounting model[1].

If you are a programmer, you're not being displaced. The core of knowledge-based systems take on the complexity issue, yet these systems will still require programmer creativity to provide the kind of user experience needed to access a knowledge-based system. Programmers must also provide the high-performance services to utilize the repository of knowledge. Fortunately, these are the most enjoyable areas for professional developers to work on.

Having a common framework will result in a major expansion in the use of computing systems to handle greater and more complex

[1]REA- (Resource Event Agent) Introduced by William E. McCarthy of the University of Michigan

systems. Having quality computing systems relying upon our accumulated knowledge will far exceed the limited and localized applications of today.

As an architect that has experienced the great value of relational database design, Object-Oriented programming with exception processing, and the standardization of web-based communications, I'm ecstatic about what is happening now. There is logic and structure throughout the entire universe of Information Technology in every sector. It only took the OWL to let me see that there is also a **Why**.

Don't believe me? I encourage you to read this book and I trust you will think differently.

Preface

Information Technology has become so complex, most projects of any size require months, sometimes years, before they're completed. And even after the projects are finished, flaws in the end product can often result in serious setbacks. Lately it seems as if every large system developed is a Titanic just waiting to find its iceberg.

Many valiant and competent professionals have targeted this problem by focusing on having well-trained and gifted project managers (PMs). These managers address both the project management methodologies and the development methodologies. They have been attempting to drive more projects towards successful completion.

Current wisdom dictates that there should always be one overall responsible PM. This individual needs to coordinate all aspects of a project and be aware of all the needs of the stakeholder. This includes the business functions and engineers across a vast range of business and technology professions.

For very large projects, the central PM might be the coordinator for multiple PMs. This is certainly the case for development functions that are outsourced to other organizations, and even when purchasing a system where the vendor coordinates the installation processes.

The industry has addressed the PM's needs through education and certification. Even still, competent PMs can be hard to find. It takes a special, well-organized individual to take on the role of a PM. It also requires someone with a sophisticated understanding of the business domains involved in the project.

Another big area of change has been to improve the communi-

cations between the developers of a project, and the customers for the technology which will be provided by the project. Agile project management methods have been invented which provide continuous feedback, allowing difficulties to be recognized early on and adjustments made. It's an approach of seeing the icebergs well enough in advance to change course and avert disaster. Unfortunately, developers generally do not have the knowledge level of business domains that domain experts possess, so like the watchers on the Titanic, they don't have the binoculars to see what is coming.

Building an application requires the chaining together of many parts, and a system will only be as good as its weakest link. The bigger the system, the longer the chain, and the higher the risk of having a dire project failure.

The IT industry's answer to this problem is to build reusable components. Reusable components bring greater quality to the applications which incorporate them, and they come in multiple forms. Even complete applications may be components for satisfying specific business needs. This is a major approach to increasing quality through reuse.

As Enterprise Architects know, adding or replacing entire applications is usually a major project. Application components must be plugged in to work with all of the other existing applications. They must also fit the needs of the organization or else they may need significant modification.

This book describes another approach that is gaining attention worldwide. This approach is based upon the computerization of knowledge. In a nutshell, the best approach to having higher quality applications is to directly share our knowledge with computers, rather than the current method of sharing knowledge with programmers, who must then take information they may not fully comprehend, translate it into programming languages, and then feed a potentially distorted and likely abbreviated result into the computers.

This book presents a prototype of this approach, and also outlines the knowledge base applied. The parts that make up the approach

are all in service today. The prototype simply brings them together to prove how the knowledge approach will change applications forever.

As knowledge is captured across multiple domains, the cost and time needed to develop complex applications will substantially decrease. At the same time, the quality will substantially increase. Wouldn't you like to know how this can and will inevitably happen? Read on to discover the answer.

Contents

Part I

Current & Future State of Knowledge-Based Systems

Chapter 1

The Current State of KBS

Knowledge-based systems are changing our society, but they are hardly a fresh innovation. Educational systems that preserve and share the archived knowledge of the past have been around since the very first student was given a lesson by a wise elder. In modern times, we spend a sizable portion of our lives in learning institutions working hard to attain varying levels of education. We receive diplomas as recognition of our efforts, and our worth on the job market is measured by our degrees and experience. We value the gathering of knowledge above almost any other human pursuit, and we have always sought faster and more efficient ways of collecting and utilizing it. Capturing knowledge is simply nothing new.

What is new are computers. Fantastic machines which can do calculations and step through processes faster than any human, and also mimic our ability to do logical deductive reasoning. Because of this capability, knowledge captured in a form understandable by computers can also be used to perform complex logical analysis by computers. Computing systems built with the intent to use captured knowledge to drive processes are called knowledge-based systems (KBS). We humans have to attend school for years to reach some level of understanding of the world we live in. Computers get theirs from us without any such effort or time investment. It all comes down to computers hav-

ing a capacity to understand, not an ability to learn. KBS is about humans capturing knowledge and providing it in a computer-readable form.

Automated systems execute their tasks by running programs, which are written in a code understood by programmers and computers. All the knowledge required to run a program must be included in the code. When a program has to reflect a domain of knowledge, the programmer must possess a full understanding of that knowledge. If that knowledge is of a complexity that takes years of training to acquire, it can be nearly impossible for the best of programmers to write adequate code within a reasonable time frame. Consider a programmer challenged to write a program describing reactions in the knowledge domain of chemistry. Without a complete understanding of chemistry, the programmer has little chance of producing a worthwhile program without first gaining that understanding, but odds are the deadline for completing the program will occur long before the programmer can earn a full chemistry degree.

With the almost unimaginable speed of computers today and the vast amount of storage available, programming has quickly advanced beyond being a mere vehicle to automate business processes. Now computers are used for major research and tracking of events. To meet these new challenges, we don't need to just write bigger and better programs, we need to capture knowledge in such a fashion that programmers can use and build upon that knowledge without having to have a full understanding of it themselves.

This is where KBS comes in. KBS is not about programming knowledge. KBS implementations separate the encoding of knowledge from the coding used by programmers. Those that have the knowledge can encode it without needing to understand the complexities of programming, and the programmers do not need to have the depth of learning required to utilize the knowledge domain.

Storing knowledge separately from program code is not a new approach. In the last few decades, there have been many successful implementations of systems by capturing rules external from the pro-

gram code. More recently, the advancement is to define ontologies containing the logic of knowledge. Provability of the logic can then be accomplished by automated reasoning. In very complex environments where no single person can comprehend the full knowledge of an environment, ontologies are becoming the norm to give computers the ability to mathematically prove the consistency of the structure of the encoded knowledge of ontologies.

There are thousands of very intelligent people worldwide who strongly believe in the value of KBS, and are proving its value with important implementations. Examples range all the way from finding a cure to cancer, to tracking the movement of everything on earth and in the universe.

1.1 Ontology and Information Technology

1.1.1 Conferences

Have you ever walked out on the street of a major city and noticed a crowd of people standing and looking up at the sky, while others on the street kept their heads down and continued moving on to their destination? If you have, did you look up or did you continue on your way? Perhaps you never even noticed the crowd looking up because you had your head down and were so focused on reaching your destination.

If your head is down, look up. There are many crowds forming and looking up at the potential of KBS. They see tremendous opportunity to help mankind by fighting disease, predicting natural disasters, and explaining our universe. It's truly worth pausing a moment to look up and see what they're seeing. If not, you will miss out on one of the greatest potential applications in the use of computers.

In 2011, I had the enlightening opportunity to attend IC3K 2011, International joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. This was the third meeting of this international group. The IC3K in preparation for this meeting received 429 paper submissions from 59 countries in all continents.

Along with these papers discussing KBS, many went beyond theory by also presenting computer-based prototypes.

The Appendix A includes an overwhelming list of knowledge engineering conferences. Thousands of dedicated individuals on an international scale attend these conferences. They recognize the need to share and collaborate on the capturing and use of knowledge by building knowledge-based systems.

1.1.2 Standards

Defining and storing knowledge has presented a challenge to develop new languages. Today, computers need precision and clarity. They don't deal well with paradoxes. For example, a paradox attributed to the Greek mathematician Zeno is the proposal that a runner can never finish a race, because the runner always has to complete half the distance to the finish line first, then half the distance remaining from the midway point, then half the distance remaining from the next midway point, and so on and so on, forever whittling down the distance by halves. As there would always be another half to complete, the runner could never reach the end, since there would forever be some fraction of the distance remaining. We, of course, understand that Zeno was only illustrating a point and knew full well the runner would eventually cross the finish line no matter how the distance was sliced, but presented with this same nonsensical formula, a computer would endlessly halve the distance as instructed and agree that the runner could never reach the goal. Computers need languages that will allow them to consider illogical ideas without being tripped up by them.

I have not counted them all, but there were more than a few episodes of the original Star Trek series where Captain Kirk challenged a massive computer with a paradox and caused it to blow up. Even back in the 60's, people understood that computers are not good at handling a paradox, though in real life, a computer is highly unlikely to become so engaged trying to solve a paradox - or any problem, really - that its circuits heat up and explode. This, luckily for those

of us who spend much of our lives sitting in front of a computer, was one prediction of the future that Star Trek got completely wrong.

The W3C (World Wide Web Consortium) has stepped up to the challenge of providing these new languages. They have categorized them in the general context of the Semantic Web. The idea behind the Semantic Web is simple. Data on the web should be easily accessible through automated means. Where HTML is designed for sharing information for humans, the Semantic Web is intended to share information with computers.

The structure selected to capture knowledge-based information is in the form of a graph. A graph is not a chart like a bar chart or a line chart. These charts are simply a form to visualize data. Graphs in semantics represent the structure of knowledge-based information, not the visualization.

Graph structure is not hierarchal nor is it relational. It is a network. It is not hierarchal because it does not have a top or a bottom. It is not relational because it does not have rows and columns as in a relational database.

The network structure has only two components, nodes and edges. Nodes represent things and edges represent the relations between things. Nodes are usually viewed as objects and edges are usually viewed as lines connecting the objects. A real life view of a graph resembles a network having no beginning and no end. It is simply a set of things connected by lines.

The term graph made up of nodes and edges is based in mathematics. Mathematicians have for hundreds of years recognized the nature of graphs and their application to real world challenges. They have defined proven algorithms we use every day. Some algorithms are used to find the shortest path through a network or how to connect nodes with the minimum cost to reach a goal.

GPS systems use the shortest path algorithm to find the route to a destination where roadway intersections are nodes and the roadways are edges. These algorithms are also used in business in many ways. In project management, shortest path is applied to find the critical path

within a project. Nodes are the project activities and the edges are the dependencies of one activity upon another. It is used in logistics to find the best route for delivering products in a supply chain. The nodes are the providers, warehouses, and destinations. The edges are the routes available. In network design, the algorithms are used to find the minimum distance to connect multiple remote systems. The remote systems are the nodes and the connections are the edges.

The W3C first introduced the RDF (Resource Definition Framework) language to define things and relations. The language OWL (Ontology Web Language) was introduced later that built upon RDF and provided additional axiomatic capability. Both the RDF and OWL languages provide syntax to define nodes (things) and edges (properties). Things are defined using class structures supporting subclasses, multiple inheritance, and equivalence. Properties are defined using axiomatic qualifications.

Both RDF and OWL also provide the syntax to declare individuals that exist within the structure defined. Individuals can be thought of as the data where the structure is the metadata (data about the data). The combination of the metadata and the data into single documents represents a complete component of knowledge. Having a mathematical base, RDF or OWL documents can be validated to be axiomatically sound. This is an important capability and will be covered later in more detail.

In addition to RDF and OWL, the W3C developed the language SPARQL (SPARQL Protocol and RDF Query Language) to provide the ability to query information stored within RDF. SPARQL provides a means for defining queries to retrieve individual data from RDF in a similar manner as using SQL to retrieve data from a relational database. The W3C also developed a language in which to define rules, SWRL (Semantic Web Rule Language). SWRL provides a means of externalizing rules that control events. The language syntax is appropriate for those that maintain business rules.

1.1.3 Ontology repositories

Sharing knowledge is the very noble driving force behind establishing repositories. It is a modern version of a public library. These libraries contain great works of art that attempt to describe the universe we live in. These attempts are based on scientific facts and on observations that may lead to facts.

All of those that participate in capturing knowledge and sharing are cultural heroes seeking to make life better for us all. They have devised these ontology repositories containing thousands of ontologies. Each of the repositories provides basic functions to add and update ontologies. The ontologies are usually organized within categories to make it easy to see the scope of the knowledge provided. Many offer search capability based on keywords.

In this section, three well known repositories are mentioned:

- OntoHub - Contains ontologies and other repositories
- Tones - A central location for ontologies that might be of use to tools developers for testing purposes
- BioPortal - A repository of biomedical ontologies

OntoHub OntoHub is described at `http://wiki.ontohub.org/index.php/Main_Page` as:

> Ontohub Ontohub is an open ontology repository which supports organisation, collection, retrieval, development, mapping, translation, and evaluation of a wide array of ontologies formalised in diverse languages.
>
> Key features of Ontohub:
>
> - OntoHub is an ontology repository that supports the ontology development and maintenance along the whole ontology lifecycle.

- publishing & retrieval: OntoHub is a free ontology repository that allows you to publish your ontology and find existing ontologies that are relevant for your work.
- development: OntoHub supports ontology development. Since OntoHub is based on Git repositories, OntoHub supports ontology versioning, branching, and merging.
- evaluation: OntoHub is designed to be a platform for ontology evaluation tools. The goal is to support all kinds of evaluation; including syntactic validation, best-practices evaluation, and regression testing.
- multilingual: OntoHub supports a wide variety of languages and logics. The same ontology may exist in more than one language. OntoHub supports the automatic translation between languages.
- open: OntoHub is based on open source software.

At the time of this writing, the OntoHub contained 2,950 ontologies and 56 repositories and organized into the following categories:

- Agriculture Forestry Fisheries Veterinary
- Arts and humanities
- Business administration and law
- Education
- Engineering manufacturing and construction
- Health and welfare
- Information and Communication Technology
- Natural science mathematics and statistics

- Services
- Social science journalism and information
- Space Time and Process
- Standard method and research technique

As an example, in the category of "Natural science mathematics and statistics" the repository called SpacePortal is listed in the sub-category of "Mathematics". It contains 63 annotation properties, 3,534 classes, 32 data properties, 1,226 individuals, and 326 object properties.

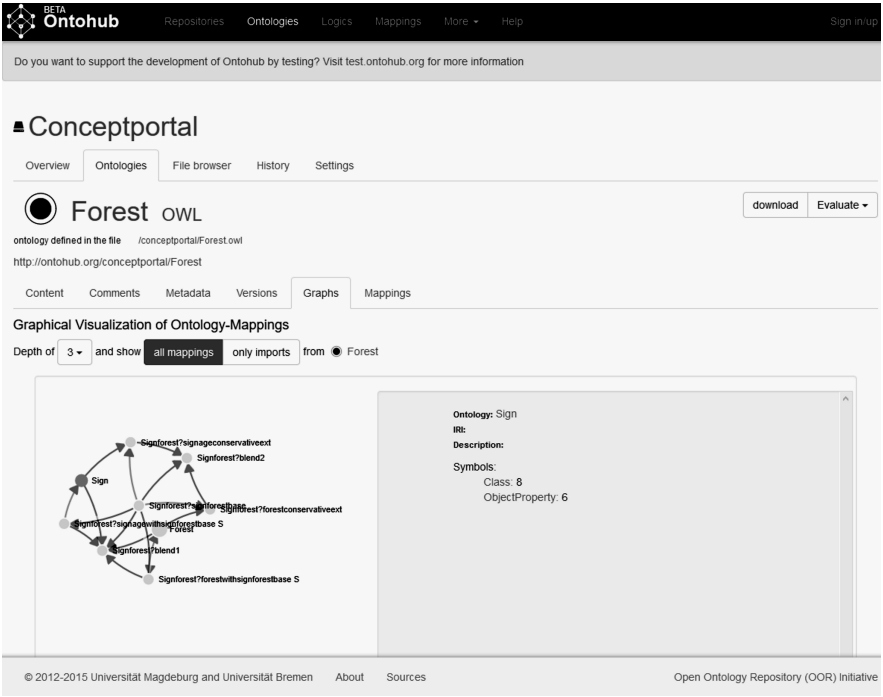


Figure 1.1: OntoHub ConceptPortal Graph View

Another example, the ConceptPortal that is described as *"populated with (common sense) ontologies in particular from the mathematics and music domains that shall be used to support and enrich the blending process"*. When selecting this repository, the OntoHub provides multiple views of the ontologies.The graph view is shown in Figure 1.1.

The graph view shows the classes and the relations between the classes. Figure 1.2 is a focus on the graph. The classes are nodes and the relations are lines. When the class "Sign" is selected as in Figure 1.1 the right side of the view shows the more detailed information about the class as shown in 1.3.

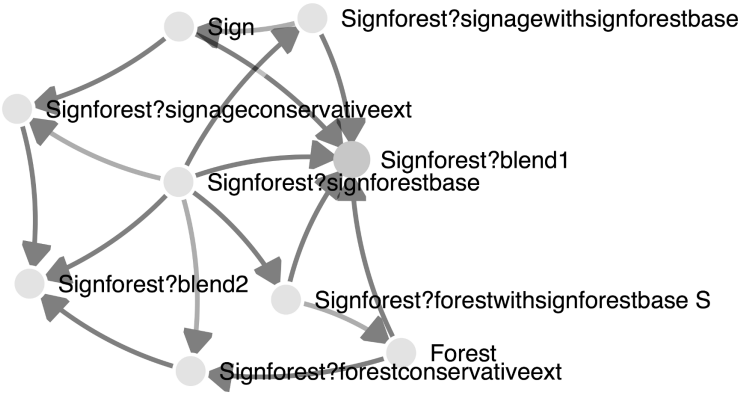


Figure 1.2: OntoHub ConceptPortal Graph

Ontology: Sign
IRI:
Description:

Symbols:
Class: 8
ObjectProperty: 6

Figure 1.3: OntoHub ConceptPortal Ontology

The OntoHub brings together many of the ontology repositories available on the web. In fact, the next two repositories are included in the OntoHub.

Tones The Tones repository is provided by the University of Manchester in the UK and supported by the Tones project. The University of Manchester is known for their work in ontology especially for the Protégé integrated development tool for OWL documents.

The Tones project is described at `http://www.tonesproject.org/` as:

> 
> TONES (Thinking ONtologiES) is an Information Societies Technology 3-year STREP FET project financed within the European Union 6th Framework Programme under contract number FP6-7603.
>
> The aim of the project is study and develop automated reasoning techniques for both offline and online tasks associated with ontologies, either seen in isolation or as a community of interoperating systems, and devise methodologies for the deployment of such techniques, on the one hand in advanced tools supporting ontology design and management, and on the other hand in applications supporting software agents in operating with ontologies. This site contains useful informations about the TONES Consortium Organization, and about the ongoing and completed project activities.

The repository contains 219 ontologies that can be searched using filters.

BioPortal BioPortal is described on the website (`http://www.bioontology.org/BioPortal`) as:

BioPortal is an open repository of biomedical ontologies that provides access via Web browsers and Web services to ontologies. BioPortal supports multiple ontology formats. It includes the ability to browse, search and visualize ontologies as well as to comment on, and create mappings for ontologies. Any registered user can submit an ontology. The NCBO Annotator and NCBO Resource Index can also be accessed via BioPortal.

BioPortal Features:

- Browse, search and visualize ontologies.
- Support for ontologies in multiple formats (OBO format, OWL, RDF, RRF Protégé frames, and LexGrid XML).
- Add Notes: discuss the ontology structure and content, propose new terms or changes to terms, upload images as examples for terms. Notification of new Notes is RSS-enabled Notes can be browsed via BioPortal and are accessible via Web services.
- Add Reviews: rate the ontology according to several criteria and describe your experience using the ontology.
- Add Mappings: submit point-to-point mappings or upload bulk mappings created with external tools. Notification of new Mappings is RSS-enabled and Mappings can be browsed via BioPortal and accessed via Web services.
- NCBO Annotator: The Annotator is a tool that tags free text with ontology terms. NCBO uses the Annotator to generate ontology annotations, creating an ontology index of these resources accessible via the NCBO Resource Index. The Annotator can be accessed through BioPortal or directly as a Web ser-

vice. The annotation workflow is based on syntactic concept recognition (using the preferred name and synonyms for terms) and on a set of semantic expansion algorithms that leverage the ontology structure (e.g., is_a relations).

- NCBO Resource Index: The NCBO Resource Index is a system for ontology based annotation and indexing of biomedical data; the key functionality of this system is to enable users to locate biomedical data linked via ontology terms. A set of annotations is generated automatically, using the NCBO Annotator, and presented in BioPortal. This service uses a concept recognizer (developed by the National Center for Integrative Biomedical Informatics, University of Michigan) to produce a set of annotations and expand them using ontology is_a relations.
- Web services: Documentation on all Web services and example code is available at: BioPortal Web services.

As an example, a query on the RxNORM ontology shows the results in Figure 1.4. The query results includes details, metrics, visits, reviews, submissions, views, and projects using this ontology.

What is outstanding is the number of visits and projects. Visits to this ontology reached a peak of nearly 30,000 in November of 2014. The projects are using the information to better understand medical imaging, integrated genomics, French biomedical data resources, and artificial-intelligence.

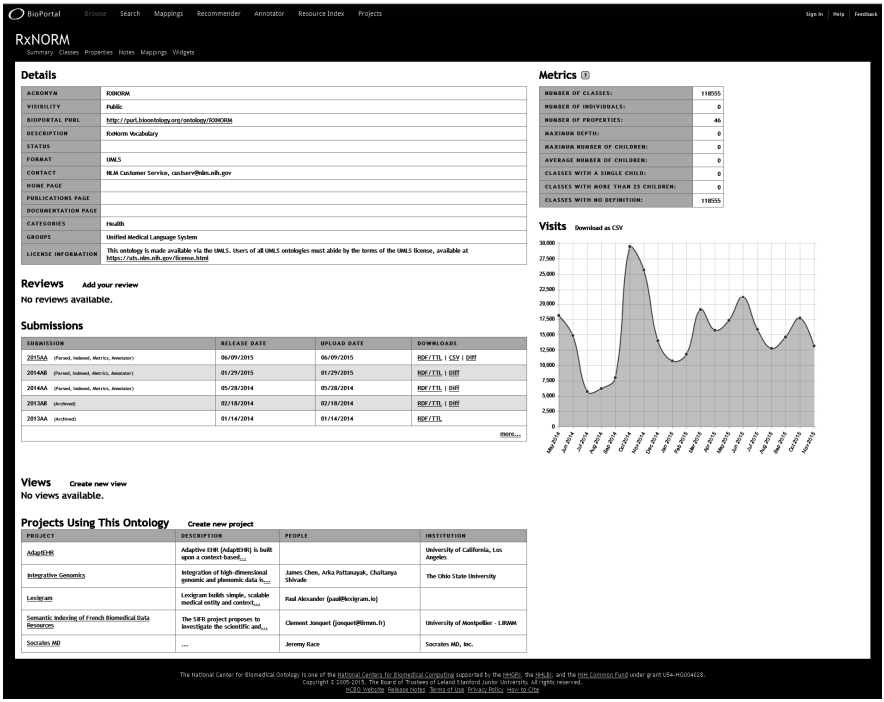


Figure 1.4: BioPortal RxNORM

1.2 High Profile Applications

This section introduces three of the most futuristic and aggressive applications of knowledge-based systems: EarthCube, Gene Ontology Consortium, and NASA - NExIOM.

1.2.1 EarthCube

About EarthCube at `http://earthcube.org/info/about` describes this massive effort as:

Spend more time doing science and less time searching for and manipulating data.

EarthCube is community-led cyberinfrastructure that will allow for unprecedented data sharing across the geosciences. Its aim is to develop a framework over the next decade to assist researchers in understanding and predicting the Earth system from the Sun to the center of the Earth. EarthCube will:

- Transform the conduct of data-enabled geosciences research
- Create effective community-driven cyberinfrastructure
- Allow global data discovery and knowledge management
- Achieve Interoperability and data integration across disciplines
- Build on and leverage exiting science and cyberinfrastructure

This project is significant and will result in the development of a large number of complex ontologies and the capturing of big data. It should also result in new understandings of the world we live in. This hopefully will allow us to be better stewards and also help predict events and avoid major catastrophes.

1.2.2 Gene Ontology Consortium

The Gene Ontology Consortium is described at `http://geneontology.org` as:

The Gene Ontology (GO) project is a collaborative effort to address the need for consistent

descriptions of gene products across databases. Founded in 1998, the project began as a collaboration between three model organism databases, FlyBase (Drosophila), the Saccharomyces Genome Database (SGD) and the Mouse Genome Database (MGD). The GO Consortium (GOC) has since grown to incorporate many databases, including several of the world's major repositories for plant, animal, and microbial genomes. The GO Contributors page lists all member organizations.

The GO project has developed three structured, controlled vocabularies (ontologies) that describe gene products in terms of their associated biological processes, cellular components and molecular functions in a species-independent manner. There are three separate aspects to this effort: first, the development and maintenance of the ontologies themselves; second, the annotation of gene products, which entails making associations between the ontologies and the genes and gene products in the collaborating databases; and third, the development of tools that facilitate the creation, maintenance and use of ontologies.

The use of GO terms by collaborating databases facilitates uniform queries across all of them. Controlled vocabularies are structured so they can be queried at different levels; for example, users may query GO to find all gene products in the mouse genome that are involved in signal transduction, or zoom in on all receptor tyrosine kinases that have been annotated. This structure also allows annotators to assign properties to genes or gene products at different levels, depending on the depth of knowledge about that entity.

Shared vocabularies are an important step towards unifying biological databases, but additional work is still necessary as knowledge changes, updates lag behind, and individual curators evaluate data differently. The GO aims to serve as a platform where curators can agree on stating

how and why a specific term is used, and how to consistently apply it, for example, to establish relationships between gene products.

The future of medical science appears to be based upon our gaining a better understanding of our own genome. So far, studies are beginning to find connections between diseases and cancers with particular genes. Preventative methods and cures are developing as a result of this understanding.

1.2.3 NASA - NExIOM

National Aeronautics and Space Administration (NASA) has constructed many ontologies in different engineering domains that are applied in space exploration. Recognizing a need to pull together all of the ontologies developed and the data collected across these domains, NASA has started the NExIOM project.

This project is described in a report entitled *"Large-Scale Knowledge Sharing for NASA Exploration Systems"* by Sidney C. Bailin, Ralph Hodgson, and Paul J. Keller. In this report, the following conclusion is given on the purpose of the project:

> 
>
> NASAs goals for the next generation of manned exploration systems are ambitious, and effective knowledge reuse will be a key component of meeting the challenges. The NExIOM project represents an ontology-based approach to knowledge reuse, pushing the limits of current technology such as OWL to formalize knowledge, while employing widely accepted technologies such as XML to disseminate and collect the knowledge.
>
> The effort is in a relatively early stage, but we have already created a comprehensive set of ontologies, generated several controlled vocabularies from them, and disseminated the resulting XML Schemas to mission projects. We have

begun to receive input from those projects with which to populate the ontologies with instance data. Thus, the ongoing process of knowledge evolution and ontology maintenance begins.

NASA continues to lead the world in providing the information needed to understand our universe. These efforts in knowledge-based systems will continue to lay the foundation for exploration into space.

1.3 Current Knowledge Sharing

Although sharing of knowledge is the purpose behind each of the repositories and implementations, each operates within its own set of domains. Sharing across the domains is limited by the technologies utilized and the structure of the ontologies.

1.3.1 Limited Sharing of Technology

In each of the examples of repositories and high profile applications there is little sharing of technology.

Heterogeneous architectures Each project has its own architects and technical staff that gather requirements and construct the solution to meet the needs of the specific users. These architectures are not interchangeable. They are heterogeneous and apply only to the identified usage of the ontologies. This restricts sharing.

Expensive to build Storing, organizing, and providing access to large complex ontologies is no small effort. It requires a large dedicated staff of highly-trained professionals. Only large organizations can justify this expense. Building large applications always requires a significant investment.

Lack of integration Combining knowledge from one domain application with the knowledge of another domain application, requires an understanding of both domains and their applications. Even though most applications provide methods to extract information, the integration of information from multiple sources defeats sharing. The intent of sharing is to share the formalization of logic not just the data stored in a repository.

1.3.2 Limited Sharing of Knowledge

As indicated in the previous section, sharing of data is not equivalent to sharing knowledge. This limitation on sharing occurs due to the design of the applications constructed to share the information.

Recorded to operate within a specific architecture For an architecture to provide value, each ontology must be defined according to the governance rules set forth in the design of the application. In other words, if the ontology does not meet the governance rules, it will not provide the value desired. Unfortunately, each application has its own governance rules, such that an ontology may be perfectly valid in one application and not in another.

Defined for a specific usage Just as an ontology must meet the governance rules to be useful within an application, it must also be defined with a specific usage intent. Even if the ontology may have multiple potential usages, its construction will only address the purpose of the application that will manage the ontology. For sharing, an ontology should not be restricted to a specific usage.

Mapping required Applications often provide some form of mapping of data from an ontology to another format. Other formats give the user of the data the ability to input the data into another application. Mapping requires the user to know how to operate both applications and have an understanding of the data provided and how it will be used.

1.4 Conclusion

The current state is:

- Thousands of people attend conferences each year to share ideas on how knowledge can be captured and shared. This is a strong indication of the direction of complex systems.
- The current standards developed give a provable mathematical base to information captured.
- There are multiple large repositories and applications that freely share information and are continuing to be expanded.
- Knowledge sharing is within specific application domains.
- Sharing across application domains remains a limitation to sharing knowledge.

This conclusion is a heads-up for all those that only focus on reaching a single destination. Join the crowd, look up, and recognize that KBS is changing our society. It is the best approach we have for building very large complex systems on which our society depends. But, there is a big problem with today's implementations. Our society wants fully integrated information, yet it is difficult to integrate the knowledge from multiple domain applications. The solution will be addressed in the next chapter.

Chapter 2

KBS Future State

Many things in life are repeated and many follow a pattern. Our society is shaped by the patterns we follow and the patterns are shaped by our knowledge of the world in which we live. In the future we will apply more inclusive patterns for the knowledge we have gained. The next biggest change to affect our society will be the application of a framework for KBS software development.

We are seeing the beginnings of this change by having our electrical grid moving towards a "smart grid", having tracking sensor networks for manufacturing, transportation, and home appliances. There are robots used from the factory to vacuuming our homes. Robots roam Mars and do experiments. An artificial intelligence computer can win on the TV show Jeopardy. We carry smartphones that each have more computing power than the total computing capability NASA had when they landed a man on the moon.

Our future lies in our software development which falls into many patterns. Some of these patterns can be abstracted and frameworks can be developed. These frameworks then become reusable in the software development process. Massive reuse of these frameworks merges these patterns into our society.

Here are a few well known examples of historical events of computing history that resulted in frameworks that have changed our society:

Situation	**Resulting Framework**
When computers became available, programs were developed that would be loaded (bootstrapped) to perform all input/output functions, task management, and memory management.	Frameworks defined as operating systems resulted to load programs and control input/output, task management, and memory management separate from the programs executed.
When direct-access storage became available, software applications were developed with their own storage and methods of access.	Frameworks defined as databases resulted to provide storage and methods of access.
When computer games were developed, individual games created their own methods for visualization and controls.	Frameworks for game development were created for specific hardware game machines to provide common methods for visualization and control.
When the Internet was invented for sharing, universities devised multiple centralized applications.	Frameworks for building browsers based upon HTML allowed for sharing documents worldwide.

All of these frameworks have enhanced our society. A framework is brewing that will change society again by making it even easier to build KBS solutions.

Knowledge in software, in the form of RDF and OWL, has begun to be used to bring axiomatic quality to information. As previously shown, applications are being developed where each one defines and develops its own methods for utilizing knowledge.

For knowledge to be shared across multiple KBS applications, there must be a common technical base. The technology must provide a common method of sharing knowledge.

Knowledge for humans is shared over the internet with a common

technical base. Information is shared by servers that respond to requests and the knowledge is presented either in HTML in a browser or by an application operating on an intelligent device.

In the future, all applications will be constructed using a common KBS framework. It will become the default standard for "computer to computer" communications as HTML did for "people to computer" communications. Like writing HTML, it will be simple and have easy to use tools available. Just imagine what today's generation of Lego experts will do with this kind of capability so accessible.

People will communicate with the KBS applications in the same manner as today using browsers and personal devices. The difference is that the applications for an enterprise will be consolidated. A single enterprise would only have one KBS application to provide all the computer automation needed. An individual within the enterprise will be provided access based upon their identity and profession. The total focus will be on supporting each individual in their area of contribution to the enterprise.

An overview of the framework and the benefits to be gained are described in the following section.

2.1 A KBS Framework

There will be a broad scope framework for building KBS applications. The need is so great and the benefits are so significant, a solution eventually must be developed. It is just a matter of time before the best approach to a framework will find its way into the open market, and be made available to anyone with a great idea that builds upon existing knowledge.

This book presents the results of a prototype framework that could be used in a broad scope. This prototype is proof that a solution can be developed. It is proof that there can be a single concept for a framework so that many will be able to share knowledge and take advantage of stored knowledge.

A broad scope framework will deliver a common technology. Just

as the technology of HTTP servers provides the sharing of human information over the Internet, a KBS framework must provide the technology to share knowledge among computers. Having a common technology will result in even greater benefits than sharing HTML.

2.1.1 Common Technology

A common technology should not be based upon any particular vendor product. It should be based upon a well defined set of specifications that can be achieved by many vendors. The use of Unix is an example of a common technology specification that is not a vendor product.

Unix is a computer operating system written in the C programming language. To execute a program written in C requires that it be translated into a specific instruction set of a vendor's computer. Once a C program is translated, it can then be executed. The translating program is called a C compiler. It can translate a C program into any targeted instruction set included in its processes. Consequently, a C program compiled on one vendor's computer can create a program that will run on another vendor's computer. This means that the Unix operating system (written in C) can be compiled on one machine and used on another with a different instruction set.

This ability to target Unix to any computer provided an operating system framework that could be utilized throughout the world. Having this framework allowed more vendors to sell computers to users already with training on how to utilize the features of the Unix operating system.

The KBS framework should be open and not vendor specific in a manner similar to the use of the Unix operating system.

Metadata Metadata brings an understanding of the data. For example, a value having the name of a color provides no meaning without indicating what it is a color of. Other meaningless data such as "yes" or "no" takes on meaning when it is the answer to a particular question such as: "Are you married?"

The purpose of metadata is more than just to name things. It exists to classify information and define the nature of each relation a class has with others. In the Semantic Web, particularly in the languages of RDF and OWL, relations are described as properties. Properties are qualified using axiomatic statements that can be used by the computer to determine the logical validity of a set of interrelated definitions.

The framework will utilize the Semantic Web standards of RDF and OWL to define metadata.

Services The framework must support the concept of services driven by domain knowledge. Where metadata is generally static, services are generally dynamic. Services can perform the traditional functions of create, replace, update, and delete of the content of knowledge. Even the metadata would be subject to change through services.

Services are self-services. Just as when you purchase gas for your car, you are required to select your payment method and octane needed. Then you can pump fuel into your tank. Going through the drive-thru window at a fast-food provider, you first make your selection, pay, and then you are given your completed order. Services for KBS are similar. Usually, some information is provided to a service to qualify what is needed and the service returns with a response.

Some services may require very little information to be initiated and others may receive information and not return anything. When mail is delivered to your home or office, the Postal service only requires a name and address to deliver and usually does not return anything to the sender. Of course the Postal service does offer tracking and may require a signed receipt for some mail.

Services must support complex algorithms and processes. Consequently, the framework must execute program code used to implement algorithms and run process controllers. Program code is written is program languages such a Java , C Sharp, PHP, etc. These languages provide the means for performing conditional data processing utilizing many high-performance data structures. Processes are written in

languages such a BPEL (Business Process Execution Language). Process languages often utilized multiple services to complete a function. They are equivalent to a service in how they are referenced, but are defined using process languages with the ability to execute over long periods of time and call upon multiple services within a network of KBS applications.

Data Storage The framework must support the storage of data and high speed access to the data. All functions must be controlled by the definitions set down in the metadata and services of a KBS domain.

The data storage is for graphs and there are products beginning to emerge to meet these requirements. These graph databases are just the next evolution of a transition that began with hierarchal databases and then transitioned to relational databases. Other specialized databases have included Object-Orient databases, search-engine databases, and no-sql databases. Now, the challenge is to provide high-performance graph databases.

Security Accessing the data and services of a KBS application must be secured. Individuals, either human or computer, must have their identification verified and their access authorization controlled. Security in automated systems is a serious issue that has a changing set of requirements as new threats are uncovered.

Security must be integral to the framework rather than an add-on. Integral means that it draws identification and control from the knowledge contained in the KBS application's metadata.

Networked Data storage and processes would exist across the network as an ecosystem of domains. Accessing a single node, should provide automatic access to all the domain nodes needed. Working with the entire network should be as easy as using an Internet browser.

2.1.2 Tree of Knowledge

Today's KBS is a forest of knowledge domains. There are multiple trees. Each tree addresses one or more knowledge domains. Some trees are of the same domain but each is different due to having a dissimilar purpose in its usage. Other trees combine domains that others separate. It is no wonder why so many projects are looking for ways to search and locate needed ontologies.

The KBS conferences held each year puts the spotlight on new trees. Everyone presenting is trying to show something new. It appears that building upon existing understanding takes away from the value of individual ideas. The outcome creates a larger forest with no limit in sight.

Libraries of recorded material (books, journals, etc.) have had to organize their forest of knowledge by providing a bibliographic classification system to give each item in the library a unique call number. Before computers, card catalogs provided access by title, author, and subject matter to a list of call numbers. The card catalogs are now automated and usually available over the web with powerful search capability.

The contents of the library are the trees. What's missing in the library systems is the layering of knowledge. Only by examining a book's bibliography or an articles references is it possible to determine what other trees a particular tree is based upon.

There needs to be library that unifies knowledge into the structures of a single tree, a tree of knowledge. The tree should have roots, a single trunk, limbs, and leaves. As knowledge grows, the tree grows, the roots would expand, the trunk would get larger, and more limbs and leaves would appear. This structure is shown in Figure 2.1

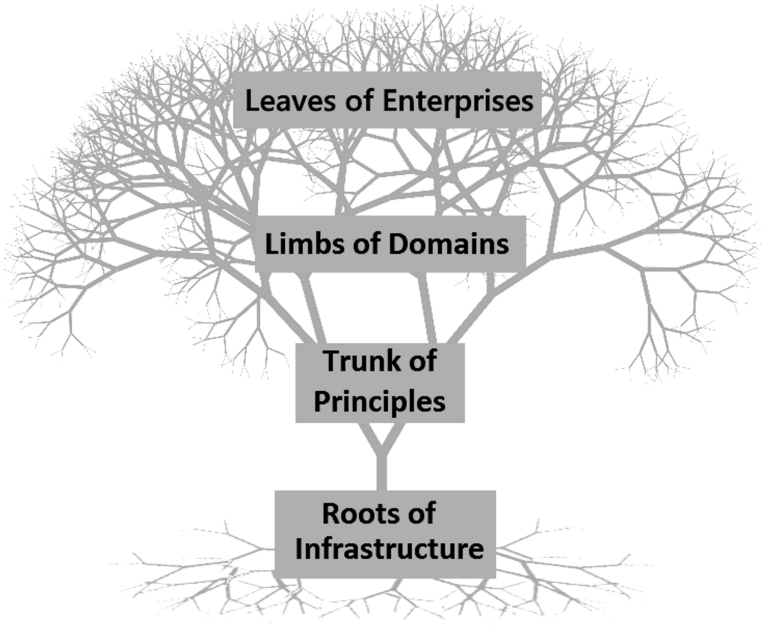


Figure 2.1: Tree of Knowledge

Roots of Infrastructure A tree gets its nutrients from the roots. For the framework, the roots are the infrastructure ontologies on which the framework operates. These ontologies provide the link to the framework's functionality which are made available to the rest of the tree. They are the nutrients that address organization, services, and security. They become the control to have consistency in the rest of the tree.

Trunk of Principles The trunk of a tree must be sturdy. It must stand strong and withstand the test of time. For the tree of knowledge the trunk is the principle ontologies on which all the

limbs of the tree are based. These principles are some times referred to by ontologists as upper-ontologies. They are the ontologies from which others derive meaning. These include the principles of the sciences (chemistry, physics, mathematics, etc.), engineering (architecture, transportation, electronics, etc.), biology (health, disease, DNA, etc.), and all others like social sciences and political science.

These principled ontologies may be viewed as the basic and absolute truths that other ontologies can depend upon. Over time, there will be new principles to add to the truck and sometimes old principles will need replacement or improvement.

Limbs of Domains Primary limbs of the tree are the specific knowledge domains that depend upon one or more of the principles of the trunk. Each primary limb is a domain category that branches out into sub-domains as secondary limbs.

Domains have two layers: occupations and industries. The occupations are the professional and clerical roles that utilize knowledge-based systems to accomplish their needs. Computer automation is a tool for those that perform in these roles. Industries require the talents of those in occupations and combine those talents for each specific type of industry.

For example, there would be a branch for each of the major business functions such as accounting where accountants would be the profession. Branch limbs would come off of accounting as the various types of accounting appropriate for an industry with needs such as managing accounts and having more limbs describing budgeting, transactions, reporting, etc.

Leaves of Enterprises Any ontologies having no other ontologies dependent upon it could be a leaf on the tree. Each leaf would be a usage by a unique enterprise where an enterprise would consist of multiple leaves.

As with any tree, the leaves provide the external view and processes that define a specific business. The leaves have both inputs and

outputs as do most industries.

Growing the Tree Over time the tree of knowledge will grow very large and at times will need trimming. For that reason the framework must contain the methods to manage the tree whether it is done by organizations or by crowdsourcing. The root ontologies which define the functions of the framework should be independent of vendors or at least not a single vendor. Organizations made up of businesses, universities, or professional business groups could take on the responsibility for managing the trunk content and all the limbs and leaves coming off them. Another potential could be crowdsourcing where knowledge is controlled in a manner similar to Wikipedia. There could even be some combination of defined groups and crowdsourcing depending upon the domain.

Unification Avoiding the forest and focusing on the tree will allow knowledge to be reusable. Each new ontology must fit into the tree somewhere. The change to the tree resulting from where it fits may be minor or significant. But either way, the power of computing logic would be applied to find any discrepancies. Having this ability to find issues is the primary reason for having a tree that unifies knowledge.

Some may be concerned that a unified tree would be too big. This thinking is old school. Today we can think big. We have the computing power and the storage capacity to think as big as we want.

2.1.3 Categories of Information Sharing

Current knowledge applications are being developed in almost every domain of business, science, and government. These efforts represent complex and successful utilization of ontology. Although, each is an independent effort that does not add to a tree of knowledge for a broad application of reuse.

Business Systems There are many large and costly efforts to define ontologies to be used by business systems. Here are a few examples:

- Insurance (`https://www.acord.org/resources/framework/Pages/default.aspx`)
- Manufacturing (`http://www.nist.gov/manufacturing-ontologies-p cfm`)
- Products and Services (`http://www.heppnetz.de/projects/eclassowl/`)
- Copyright, Multimedia, eBusiness, News (`http://rhizomik.net/html/`)
- Financial Reporting (`http://financialreportontology.wikispaces. com/`)

Internet of Things The Internet of Things (IoT) was the theme of the Ontology Summit 2015 Symposium. The keynote speakers, Mark Underwood and Michael Grüninger, described the purpose of the conference as:

- The emergence of networks of intelligent devices which are constantly sensing, monitoring, and interpreting the environment is leading to the Internet of Things (IoT).
- Ontology Summit 2015 has provided an opportunity for identifying challenges (seamless semantic interoperability, recognition and reaction within situation awareness) for the Applied Ontology community within the broad range of IoT applications.
- We have taken the first steps towards demonstrating the utility of ontologies to the IoT community.

Specific applications of ontologies used in sensor networks were discussed and demonstrated. One example described by Holger Neuhaus and Michael Compton is of a semantic sensor network ontology by a Commonwealth Scientific and Industrial Research Organisation in Australia.[1]

The conference completed by drafting a communique emphasizing the importance of ontology to IoT.[2]

Big Data Big data, as its name indicates, is a volume of information issue. With business organizations capturing every transaction such as a phone company recording every call or NASA tracking the universe, big data is a volume management challenge.

Large retailers need to analyze sales data to make forecasts. NASA needs to track every satellite and junk orbiting the earth to avoid collisions. To forecast the weather, models must be developed by understanding the data from the past. To use a navigation system to move on our highways, every road, intersection, and conditions must be recorded.

Current technology, particularly data warehousing based upon relational databases, is not efficient for the large volume processing of big data. Technologies such as Apache's open source tool Hadoop have been developed specifically to process large volumes of data. Many vendors have extended the Hadoop software to produce even better tools.

There are several efforts underway to uncover approaches where ontology can enhance big data solutions. The Ontology Summit of 2014 addressed the topic and organizations providing services and tools have been formed to move this effort further.

Storage of big data information is being addressed through the implementation of graph databases. High profile vendor solutions such

[1] http://www.esdi-humboldt.eu/files/agile2009/Neuhaus2009_Semantic_Sensor_Network_Ontology.pdf

[2] http://ontolog.cim3.net/file/work/OntologySummit2015/2015-04-13_14_OntologySummit2015_Symposium/OntologySummit2015_Communique.pdf

as the Oracle Spatial and Graph product and the IBM System G Native Store are available today. Many other vendors and open source products have also entered their solutions to the big data challenge.

In review, the current efforts recognize that processing big data is both a storage and a retrieval problem that is best handled using the ontology modeling approach. This only proves that if ontologies can be used to solve big problems, they can certainly handle smaller problems.

Artificial Intelligence Computers are popularly thought to have the ability to grow as smart as we are or even smarter. This has long been a favorite inspiration for technophobic movie plotlines, which depict advanced machines deciding they should be giving the commands to us rather than the other way around. More often than not, the self-aware robots imagined by Hollywood instantly declare all-out war against humanity to seize control of the planet, though it's rarely explained what greater purpose they intend to occupy themselves with after they've gotten us out of the way.

In reality, while we have seen giant strides towards our computers becoming more like us, it's generally always led to them becoming more helpful to us. The first major advance was developing software that not only could translate text into a spoken language but could also speak. With a smartphone there are several applications available using the analogy of an assistant that can hear your questions and find you solutions. If you're hungry, you can say "Find me a pizza," and have your phone list and give directions to the nearest places to get a pizza. Some are much more capable and can assist with making phone calls and organizing schedules. One used by many each day is to get a route to a location.

In 2011, IBM entered a computer named Watson to play Jeopardy on TV and it won. The artificial intelligence applied by IBM's machine performed a far more complex operation than simply looking up answers. Watson had to evaluate multiple potential answers, and at the same time, play the game to know when to answer. It may not

have been as impressive as one of Hollywood's sentient robots that can declare "I think, therefore I am," but at least Watson can say, "I think, therefore I won."

Linked Data Linked data is giving access to publicly available data in a consistent form following the guidelines of the W3C Semantic Web initiatives. From government agencies to business groups, there is a multitude of data available for access. Pick a topic and search the web for linked data on that topic. You will more than likely find more than one source.

There is no single method to access the linked data information. Some provide the entire set of data, others offer selection of sub-sets of the data. Some even support SPARQL, the W3C query language for RDF.

The amount of linked data is constantly growing. With the availability of cloud storage and computing power, this trend will continue.

Research Research is probably the most important application of KBS. As presented previously, research is mapping the human genome and finding cures for disease. Cancer and other incurable diseases will be a thing of the past as the research continues to uncover prevention and cures.

One day we will be sending colonies to Mars. KBS will play a significant role in preparing for this due to the enormous amount of research information being captured today and organized using ontologies.

2.2 A New Development Paradigm

If you have an idea to use computer automation, but lack any of the following:

- The detailed knowledge to fully explain your idea
- Access to needed information repositories to prove your idea

- The technical skills needed to demonstrate your idea

then you are a candidate for using a knowledge-based system framework and the library of knowledge.

In the future, detailed knowledge will be found in the layered ontologies of the library of knowledge built upon the knowledge-based system framework. These will be ontologies that have been recorded by those with a detailed understanding. These ontologies would be publicly reviewed and verified by reuse.

Information would be easily accessed. Repositories of publicly available big data would be mapped into the framework. The Mapping process translates the information into well-defined ontologies.

In-depth technical skills would not be required to use the framework. Although the framework will apply the current and most innovative technical skills in its construction, they would not be required to use the framework. It would be like driving a car without needing to know exactly how it operates.

The framework and the library of knowledge would be for dreamers to visualize and prove their ideas.

2.2.1 Reduced development time and expense

Reuse is the greatest approach to saving time and expense. This will be true for the KBS framework. From large organizations to single individuals that have an idea, using the framework and library eliminates the need to design from the ground up. Projects that could previously not be funded could be initiated. This will be of a significant benefit to individuals that would like to try out an idea.

2.2.2 Knowledge controls quality

One of the biggest efforts in any development project is in managing quality. Primarily, the quality of knowing the system delivers what the client wants.

Approaches such as agile project methodologies have emerged with the idea of development in small steps so that the client can validate

the outcome of each step. This is a major improvement over developing a massive requirements document used as the specifications for a new system. Unfortunately though, whether agile of some other method is used the issue is still whether the IT staff and the client both understand the requirements the same way.

In a KBS built on a framework, the content of the knowledge controls the quality of the system. The ontologies describing the knowledge are the requirements. These requirements are read by the computer and they must be logically sound. The quality of the system is primarily in the hands of those that prepare the ontologies which could and should be the clients themselves.

2.2.3 Provides executable ontology requirements

To gain access to the framework's functionality, there will be requirements that must be met in the structure of the ontologies. It is the requirements that allow an ontology to become executable.

Having these requirements for the ontologies, sets the standard for sharing them across multiple implementations of the framework.

2.2.4 Common easy to use tools

There are many tools available today to work with ontologies written in OWL and RDF. There are even tools to create and edit ontologies within specific knowledge domains.

Just imagine how many more tools there would be if a framework were used. Because the framework would be the default method of defining a KBS, tool developers would have a much larger population of potential users. With a larger population there would be more users to provide feedback and more tool developers with ideas for new and improved tools.

2.2.5 Reuse of knowledge for everyone

As presented in the section describing the tree of knowledge, ontologies should have consistency with all other ontologies within a taxonomy.

Adding to the tree would be straight forward by expanding on one or more existing ontologies. The methods of accessing the knowledge within the tree would be consistent for all information.

2.2.6 Provable Information Modeling

The most powerful aspect of using ontologies is the strict form of logical specification required by the current languages of OWL and RDF. The KBS can then take these specifications and process them through an automated reasoning process to uncover any discrepancies in the model.

Without this provability, systems today are totally dependent upon complex testing approaches. Provability does not eliminate the need for testing, but it does eliminate logic errors in the implementation of the specifications.

2.2.7 Redefines Enterprise Architecture

Large organizations that require individuals in the role of Enterprise Architect (EA) to maintain the Enterprise Architecture, will experience major changes as KBS becomes common place.

EAs provide the governance processes to control the infrastructure and business domain deployments for their organization. Today, this function requires the development of models that reflect the organization's technical and functional processes. These models only reflect the actual organization's computing structures and must be manually updated as changes take place.

When all systems are KBS, there is no need for business models. As already presented, the ontologies are the executing specifications of the KBS. EAs can automatically produce visuals of the current architecture from the KBS repository of knowledge.

Governance is moved to the tree of knowledge. Each organization will have their own combination of ontologies from the library of knowledge. Only the ontologies needed would be included. The

results would be a unified lattice of ontologies dependent upon the unique requirements of an organization.

The EAs will be responsible for their organization's lattice by including only the ontologies needed. If their business expands, then the lattice would have the appropriate ontologies added. If their organization contracts, then the appropriate ontologies would be removed.

All business development work would be in maintaining the lattice since the ontologies provide the executable functionality of the organization's computing resources. This places governance on the logical processing of the KBS rather than upon the EAs.

Producing visuals from the executing ontologies could show the structure of the information, the services, the domains, the customers, the providers, the products, and just about anything else desired. Since the framework controls execution, the actual performance of components could be shown. It would even be possible to simulate or forecast the effects of changes in the business or the structure.

Using 3D graphics would make it possible to tour an organization from inside the computer where everything is real and active. Running the technology of a business will be more like playing a game. Maybe Hollywood had it right when they produced the movie Tron.

2.3 Conclusion

Society has been slowly moving up on the beginning of a roller coaster ride. Once the top of the first elevation is reached by having a framework for KBS applications and a taxonomy for the library of knowledge the fast-paced ride can begin.

The high-speed of the coaster will come from the new development paradigm for KBS as a result of a common framework. It will reduce development time for KBS applications. The quality of knowledge will control development and execution. There will be common, easy to use, tools and knowledge will be available to everyone. Anyone with a great idea will have the means to test their idea. Researchers will be able to move more quickly and easily share information across

domains.

One of the biggest changes will be in Enterprise Architecture. Executable ontologies will replace models by allowing visual inspection of all software computing resources. Governance will be turned over to the logic applied by the KBS framework.

Society will suddenly move into new potentials for having computing resources to guide and help in everyday life.

Part II

Prototyping the Future State of KBS

Chapter 3

The OTTER Prototype Overview

OTTER stands for **O**ntology **T**echnology **T**hat **E**xecutes **R**eal-**T**ime. It is a prototype project that proves that a KBS framework can provide the technology on which to develop small and large scale KBS applications.

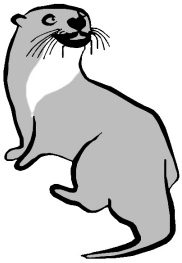


Figure 3.1: Mascot

The project has adopted the happy, easy-going otter as its mascot, as shown in Figure 3.1. Otters are found world-wide. They work hard, love their families, and have a lot of fun, and in many ways, these are the goals of a KBS framework. It should work hard for us. It should

embrace the entire family of ontology standards. And it should be fun to work with. The prototype framework has laid a foundation to reach these goals.

In this chapter, the four major aspects of the prototype are introduced:

- Architecture - Describes the concept, technology, and application of layered architecture.
- Executable Ontologies - Presents three aspects of what makes an ontology executable.
- The KBSgraph - Describes this structure as derived from an OWL class expression.
- The KBSgraph as the Cornerstone - Shows how all aspects of the OTTER implementation are in alignment with the KBSgraph.

3.1 Architecture

The architecture of the OTTER KBS framework consists of three basic components as shown in Figure 3.2.

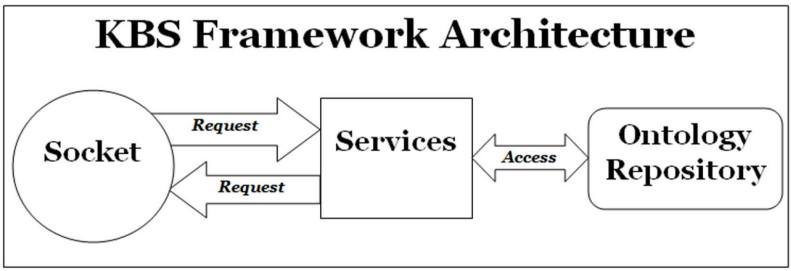


Figure 3.2: OTTER Concept

The socket is like an identification badge. The badge provides security by identifying the badge holder, determining their level of

access, and declaring the functions they provide. Also, just as a badge is only issued to those candidates who meet the right qualifications, a socket is only provided after satisfying security requirements. Once an entity plugs into a socket, a connection is made to the socket's services which determine what functions will be provided to the entity. The socket also defines the functions expected to be provided by the entity plugging in. The socket may make a request of a service and a service may make a request of the socket.

The services within the framework then have access to the ontology repository and to any services provided from the socket. Service access to the repository includes creating, reading, updating, and deleting individuals in the repository. Services may also access services of other components including those provided by other socket components.

Socket security determines what entities have access to sockets and what services have access to ontologies in the repository. Entities plugging into a socket provide proof of identity while services are given ontology access by design. Services can also limit the access of specific entities to the full content of an ontology.

3.1.1 Concept

This conceptual architecture is not like any other. The socket, the services, and the repository are all ontology based. These ontologies define and control the functions of the framework.

Ontology Repository Ontologies are all written in OWL and follow some basic rules for operating within the framework. These rules provide the patterns that the framework processor can recognize to activate functionality.

The repository is a set of ontologies brought together for a specific purpose. Any set of domain ontologies can be built upon the framework ontologies of OTTER.

Services Infrastructure ontologies are used to define services as individuals within a domain. The successful industry Service Compo-

nent Architecture (SCA) model is applied. In SCA, there are components that provide services and make references to other component's services. The flow is always from a reference to a service.

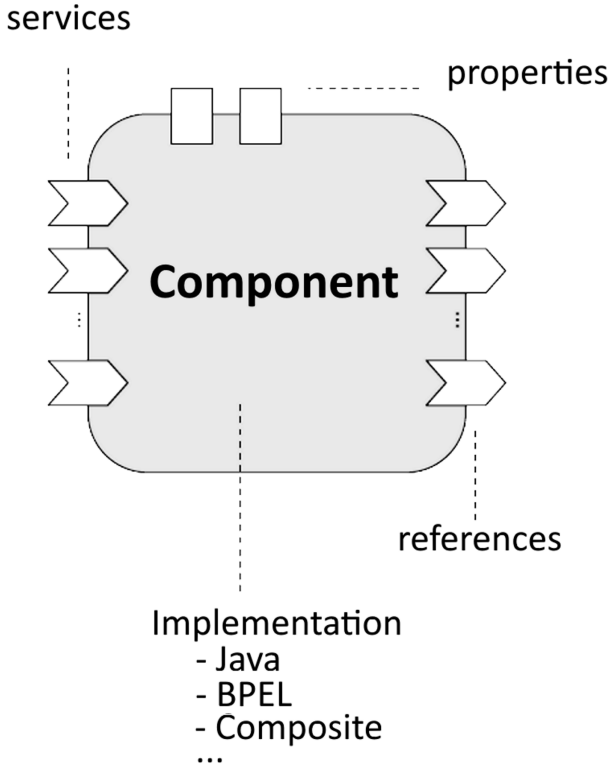


Figure 3.3: SCA Component

Figure 3.3 shows a component with services on the left side and references on the right side. In addition, properties may be provided to a specific instantiation[1] of a component. Also, components may be written in multiple programming languages. In the OTTER prototype, the services are written in Java.

[1]Instantiation: Dynamically creating an object from a specification.

In SCA, a component can be an assembly of multiple components by linking the references of services to other component services. The services and references of the assembly are those exposed from the assembled components. This is described in the framework ontology that controls services.

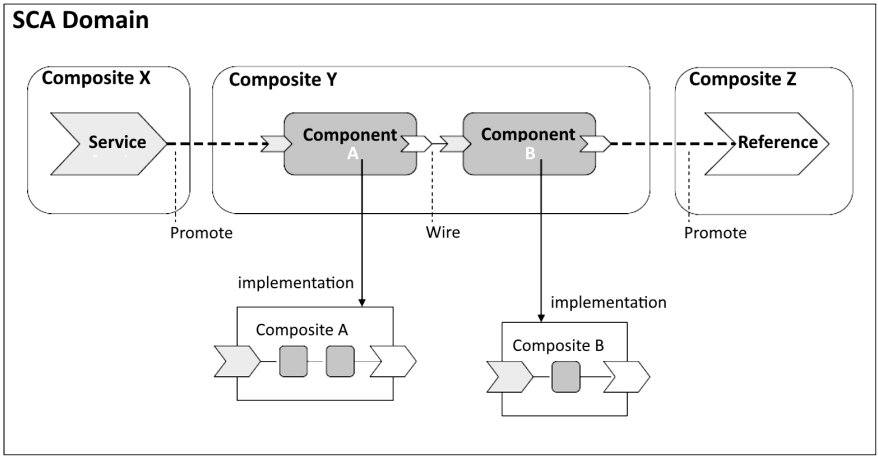


Figure 3.4: SCA Assembly

An example is shown in Figure 3.4 where Components A and B are wired together into Composite Y. The services and references of Component Y are promoted from Component A and Component B respectively. Components A and B are both composites showing that multiple levels of assembly are possible.

Sockets As already mentioned, a socket is a two-way street for an external component that plugs in. Besides being given access to services, the external component may need to provide services. The socket is therefore the gateway for an instance of the framework to provide KBS application services to external components as well as provide usage of external services.

An external component could be an Internet browser using JavaScript

to utilize the socket. It could also be another instance of the OTTER KBS framework that references and provides services. It could also be any computer application, including mobile computing.

Players Players are the entities that have security access to one or more sockets. For people, they may have an interface through a browser or a smartphone. When plugging into a socket, people are not users of an application. They are players. "Players" means they have responsibilities to complete service requests in the same manner as computer applications.

Plugging into a socket may be done by a fully automated system. After a security verification, the system connected would also be a player in the overall network where multiple systems could play.

We can finally drop the old term of "users" and recognize the more modern term of "players". All components are players, including the people.

3.1.2 Technology

The technology is based upon the OWL language utilizing the open source desktop Protégé editor. OTTER operates as a standard plug-in within Protégé. The plug-in provides an HTTP server that acts upon executable ontologies for security access to services.

Protégé Protégé is primarily an ontology editor with support for plug-ins and reasoners. Ontologies may be created or updated.

The structure supported for editing an ontology includes:

- Annotations - The ontology and any of its entities may be described.
- Including Ontologies - Knowledge can be applied from other ontologies.

- Classes - A category of entities can be described and related to other classes.
- Object Properties - The relations between classes can be described with precise axiomatic statements.
- Data Properties - Specific data associated with classes can be described.
- Individuals - Real entities may be defined as members of classes and having object and data properties.

Protégé is being developed at Stanford University in collaboration with the University of Manchester.[2]

HTTP Server The HTTP Server is a Protégé plug-in written in Java. When opened, the server begins listening on the defined port. It provides a secured login for access to service requests following HTTP standards including JavaScript Object Notation. It recognizes commands from players to begin a session, return a list of accessible services, initiate a service, retrieve the response from a service, perform a direct query on the repository, and end a session as shown in Figure 3.5.

[2]This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

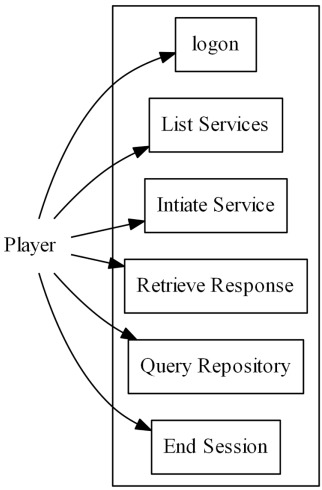


Figure 3.5: HTTP Server

When the server is started, it retrieves the definitions of all the SCA services and references, and establishes links for execution. When a service request is made, the link is located and the service is initiated as shown in Figure 3.6.

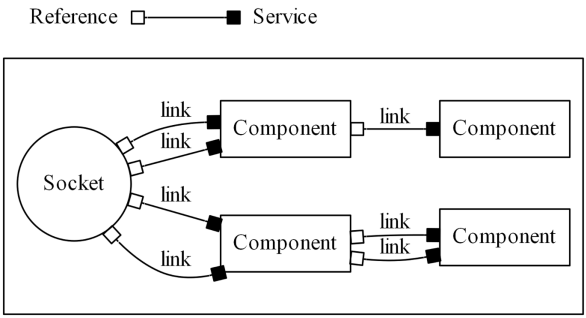


Figure 3.6: Reference to Service Links

JSON JavaScript Object Notation (JSON), a data-interchange format, is used to serialize message content to and from the HTTP server.

Web Browser For the prototype, a web browser supporting JavaScript is required. A library of JavaScript code functions are provided in OTTER to access the server and the services.

3.1.3 Layered Domain Patterns

The organization of patterns used to define the ontologies reflects the structure of knowledge as processed by the framework. The patterns used are infrastructure, academic disciplines, domains, and implementation.

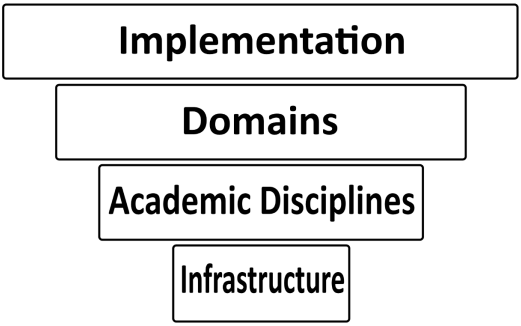


Figure 3.7: Layered Domain Patterns

As shown in Figure 3.7 infrastructure is layered by academic disciplines, then domains, and implementation is at the top. Like an upside down pyramid, the framework infrastructure is the base for a larger number of academic disciplines. Then there is a larger number of domains that build upon the infrastructure and academic disciplines. At the top, there will be many implementations.

Infrastructure The infrastructure patterns define the entities to be used in the framework's execution. These provide the structures for Service Component Architecture, organizations, and process management.

Academic Discipline Academic discipline patterns declare common truths. For example, in the discipline of Project Management, there would be definitions for activities having commitments and constraints. These could be used to define different project structures and project types, such as research or construction.

A discipline may contain service definitions as well. In the example of Project Management, there may be a service to calculate the critical path for the project as well as the slack time available for each activity.

Domain A domain is a specific area of knowledge that may have multiple sub-domains. These areas are often referred to as subject areas, each having subject-area domain experts. If the area of knowledge is software development, it may have sub-domains for project management, design, testing, quality assurance, and implementation.

Domain patterns would extend infrastructure and academic discipline ontologies, and describe the metadata for the data structures and for the services provided. The data structures are the logical patterns describing the non-transient information to be stored in the ontology repository. The metadata for the services would describe the logical patterns of the transient information of messages.

Ontologies describing the messages used within a domain may be thought of as the interface to the domain. In design, the ontology developer would expose only the domain entities needed in a particular message. This could mean an independent ontology, or it could be based upon other ontologies within the domain.

There are several structural approaches an ontology designer might follow in defining message ontologies. These ontologies could be independent ontologies in the domain, or dependent upon other ontologies in the domain.

One approach is to have a root ontology which has the structures common to both the transient and non-transient information. Then, the more extensive knowledge of both the transient and non-transient ontologies would be derived from the root ontology. In this approach, a reference would only need to have exposed to it the root ontology and the more extensive ontology for messages.

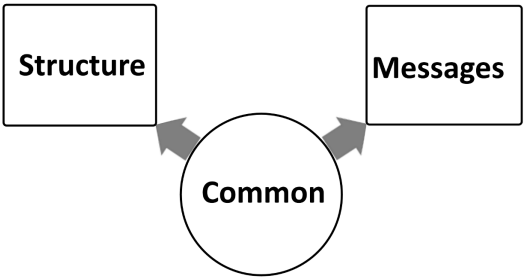


Figure 3.8: Only Expose What is Needed

This approach is shown in Figure 3.8 where the common ontology contains knowledge used by both the message ontologies for transient information and structure ontologies for the non-transient information.

Implementation Implementation is build upon the domain ontologies and includes the individuals within the ontology. Implementation ontologies in the framework are only accessible by those with security access. A service that reads and updates individuals would need to have access, while a player may only be limited to accessing services.

3.2 Executable Ontologies

There are three forms of execution in the framework:

- Logical Validation - This is accomplished by a general reasoner.

- Service Execution - Services are defined within a domain.
- Common Language - Messages are filtered by understanding.

Having each of these forms of execution provides a complete capability for constructing a KBS application.

3.2.1 General Reasoner and the Army of Axioms

A general reasoner has the responsibility to satisfy all members of the army of axioms. The reasoner aligns the axioms into a dependency of logic to determine that all axioms are satisfied. The framework will only allow changes to an ontology that can get past the general reasoner.

For hundreds of years, mathematicians have worked proofs to declare theorems. They take individual steps of logic that lead up to the final statement of the theorem. This effort takes time, many years in some cases, to find just the right combination of axioms to finalize a proof.

When I was in college, on the final exam for a math course, each of us were asked to provide the proof for twelve statements. We were given three days to provide the proofs. I found ten of them to be quite easy because we had done similar proofs in class work. The last two were very difficult and did not appear to follow the patterns we were taught. I found a way to prove one of the statements, but I did not find a proof for the other.

I received an A in the class for completing the ten and extra credit for the extra one completed. I was the only one that provided a proof for this statement. Another student gave a proof for the other challenging statement. Later we learned the professor was writing a book and needed these proofs to publish.

The outcome of this story is that applying logic to prove statements is an arduous task that can take a great deal of time. It requires an in-depth knowledge of the subject area and an understanding of logical steps to reach conclusions. A general reasoner makes it look easy.

Axioms The army of axioms is in the knowledge-base. For example, a property can be defined having specific requirements. One might be that an object relation can be defined as "functional", meaning that an individual in the domain of the property can only have one individual in the range of the property. In database terminology, this would be equivalent to a one-to-one relationship.

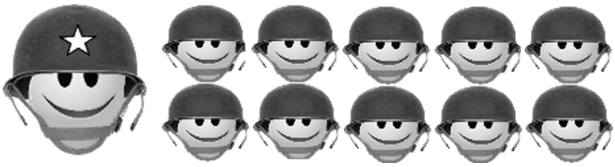


Figure 3.9: Happy Army of Axioms

Figure 3.9 shows an army of axioms, all of which are happy, which also makes the general reasoner happy.

Reasoner The general reasoner automates a logical process by using the axioms included in the knowledge-base. The reasoner operates by considering all the valid moves provided by the axioms as depicted in Figure 3.10[3]. The reasoner iterates through the moves to assure that each one is in fact logically consistent. If the reasoner finds that even one axiom is inconsistent, it stops the game.

[3]Provided by alfa-img.com.

Figure 3.10: General Reasoner

Managing Change Although classification of things is important in an ontology, it is the axioms that provide the executable knowledge. Only through providing automated reasoning can all axioms be tested.

In the framework, any change to an ontology is given over to the general reasoner to verify the change with the army of axioms. If any axiom reports an inconsistency, the change is rejected.

Figure 3.11: Unhappy Army of Axioms

Figure 3.9 shows the general reasoner is unhappy even when only one single axiom in the army is unhappy.

3.2.2 OTTER Service Infrastructure

Execution also comes from the infrastructure ontologies defined for services, businesses, and processes where the services ontology is the foundation layer for both business and process.

Services Services are defined by an ontology that is fully executed by the framework for both internal and external processing. Internally, application services are instantiated for execution to receive and send messages from sockets or other services. For externally provided services and references, message requests are also managed by the framework.

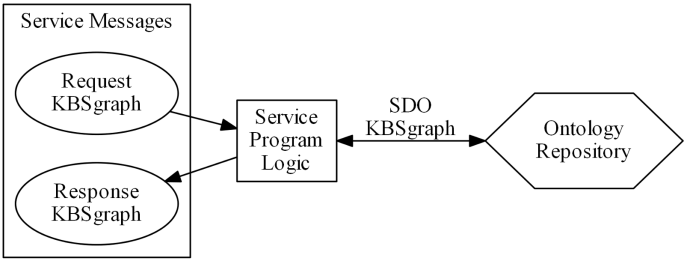


Figure 3.12: Service Messages

A service is passed the request and response KBSgraph using the standard of SDO (Service Data Object) as shown in Figure 3.12. The program logic is initiated as a task that processes the request and produces the response. The task also has access to the ontology repository.

Business The business ontology classifies things as people, organizations, or software. People are individuals with a unique identification. Organizations include organizational units of the business, the business's customers, and their providers. Software identifies computer components.

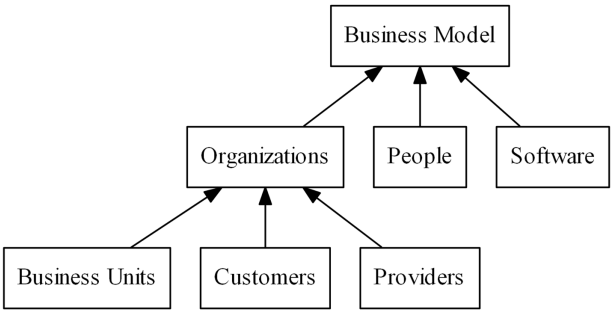


Figure 3.13: Business Model

Figure 3.13 is a high-level view of the business ontology executed by the framework. The business model separates organizations, people, and software. People get access to services by providing their identification, and software defines the applications to be loaded for service execution.

Process Process is provided at the business process level. The Business Process Execution Language (BPEL) ontology classifies the operations, their relations, and the data content that is used by the framework. Once a process is initiated, the framework steps through and executes each of the operations.

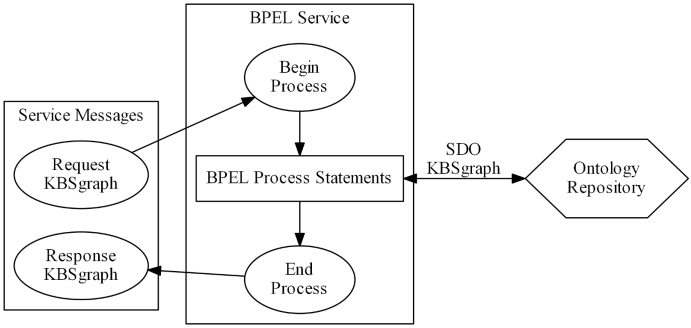


Figure 3.14: BPEL Service Messages

The BPEL processor of the framework is passed the request and response KBSgraph as shown in Figure 3.14. The processor compiles the BPEL statements at initiation and executes the compiled statements. The statements process the request and produce the response. Statements within BPEL provide access to the ontology repository.

The ontology describing BPEL provides execution sequencing by linking the statements together. Linking provides the order of execution. This includes multiple links for step wise processing, conditional processing, and parallel processing.

3.2.3 Common Language for Shared Understanding

Services always have a request message and usually have a response message. The requester has matching send and receive messages. These messages sent and received must use a common ontology. The ontology could be the same ontology or a subset or a superset of the domain ontology providing the services.

The request to a service contains what a requester knows and the response contains what the requester wants. There are three forms in which message properties are formed:

- An instance of data - Metadata and a value.

- Missing value - Only has metadata.
- Not needed - Neither metadata or value.

The receiver of a message should interpret an instance of data as important to the requester. A missing value is a recognition that the requester recognizes the importance of the information, but does not have any value to assign to it. When the metadata is not included, the requester is indicating that it places no importance upon this information.

What the Requester knows The framework execution matches the ontologies of the send and request for commonalities. The outcome of this matching is the structure of the message allowed to be passed to the service.

For example, a service that provides information on businesses would need to know the identity of the business. Although the service's identification for a business could include the business name, their address, or telephone number, the ontology for the send used by the requester may only define the use of telephone number.

In this same example, the requester ontology must define a business description in the service request such as telephone number that is the same class or equivalent to the service's request ontology. The requester must then set a value for the telephone number before initiating the service.

What the Requester Wants The framework executes the same matching process for the message of the response and receive. Although a response may have the ability to provide a broad amount of information, the requester can limit the content by the receive ontology.

Following on with the previous example of a business request, the service may be able to provide organization, financial, sales, and a multitude of information about a business in the response ontology.

The receive ontology can restrict the content of the response message by only including a need for a business address.

3.3 The KBSgraph

Knowledge is best understood and processed in a graph form. This is because graphs can be very simple to visualize, yet contain enormous complexity. Graphs are also much closer to the way we think about and view the world, as points connected with lines.

We can look into the night sky and see the three stars that make up Orion's belt. Some folks can even see all of Orion with his sword. In doing this, we are graphing the constellation of Orion. The stars are the points, and the imaginary lines we supply make up the graph.

Many forms of graphs can be derived from OWL with classes as points, and lines of relationships as subsets, supersets, equates, object properties, and data properties. Other graphs can be derived from OWL where the points are the ontologies, and the lines are the included ontologies.

The KBSgraph is derived from an OWL class expression. It is unique to the framework and represents a Knowledge-Based System graph. The metadata of the graph is extracted directly from the class expression, and the individuals that populate the graph are those that satisfy the metadata.

3.3.1 OWL Class Expression

An OWL class expression is algebraic in its form combining existing class definitions. It includes restrictions in the form of the class operations of "and" (intersection), "or" (union), and "not" (complement) and in the form of class properties and data properties.

Union Union defines a class that contains all the individuals found in two other classes. If an individual exists in both classes, it will appear only once in the result of a union. The union of fishing

boats and row boats would only have one inclusion each of row boats used for fishing.

Intersection The resulting class of an intersection only contains the individuals that are members of both of the classes in the intersection. The intersection of boats having deep keels with boats driven by wind power would only contain sailboats with deep keels.

Complement In OWL, the "not" refers to classes that are defined as disjoint classes of a class. They are refereed to as the complement. For example, the class of women may be defined as disjoint with the class of men. In other words, an individual in the class of women cannot also be in the class of men. The reverse is also true. A class expression including the statement of not women is interpreted to be the class of men.

Class Properties A class can have an object property with another class, or a data property with a data type. In OWL, a property may be restricted by the class that has the property by the existence of the property or by the number of the individuals having the property.

Restriction by class uses the keywords "some", "only", and "value". In each of these, the restricted class must have the given property with another class. For example, with the property of "has child" defined between "one person" and "one or more other persons", a "Mother" could be defined as a "Female" that has the property of "has child" of "some person".

The keywords "min", "exactly", and "max" are used to restrict membership based upon the number of individuals having a property. In the previous example, a "Female" that has the property of "has child" of "exactly 2", would define a set of mothers with exactly two children.

Data Relations Data relations may be restricted by value, datatype, and XML facets. A value restriction might be "has age" of value

"21" and a data-type restriction might be "has age" of data-type integer.

Facets provide a means of applying more specific restrictions on data-types as shown in the following table:

Facet	Meaning
`< x, <= x`	Less than, less than or equal to x
`> x,>= x`	Greater than, greater than or equal to x
`length x`	For strings, the number of characters must be equal to x
`maxLength x`	For strings, the number of characters must be less than or equal to x
`minLength x`	For strings, the number of characters must be greater than or equal to x
`pattern regexp`	The lexical representation of the value must match the requlars expression, regexp
`totalDigits x`	Number can be expressed in x characters
`fractionDigits x`	Part of the number to the right of the decimal place can be expressed in x characters

Algebraic Complex class expressions may be prepared by using parentheses to define class expressions within a class expression. Wherever an OWL class expression requires a class, a class expression can be described within parentheses.

For example, parentheses are used in the following statement:

```
Female and hasChild some (person hasAge some int[<21])
```

In this statement the outcome will be females that have children less than age 21. The parentheses are used to define the set of persons less than 21. Using "some" then selects from that set those that are in the range of the "hasChild" property.

3.3.2 Extracting Metadata

The framework extracts the metadata from a class expression to describe a KBSgraph. It extracts the domains, root classes, object properties, data properties, and range classes. Only the classes and properties used in the class expression are included in the KBSgraph metadata.

The metadata describes the component of a graph. The classes are the points or nodes, and the properties are the lines or edges. The graph may have cycles. In other words, following the lines in the graph may loop back to a point already followed.

Property values may be one or more. In OWL, a property that is defined as "functional" can only have one individual in a range. This restriction is captured in the metadata.

Domains The domains are the ontologies containing the classes and properties found in the class expression. These domain names can be given an abbreviation when using a KBSgraph to allow for shorter names for the classes and properties.

Root Classes The root classes of a class expression are those that meet all the requirements of the class expression. When a class expression is qualifying membership to existing classes, there will be one or more root classes. When the class expression defines a new class, then there are no root classes identifiable by a name in the ontology.

Object Properties Object properties used in the class expression are included in the metadata by assigning the property to the domain class and the range class of the property as defined in the ontology.

Data Properties Similar to object properties, data properties used in the class expression are included by assigning the property to

the domain class, but the range is assigned to the data-type defined in the ontology.

3.3.3 Populating with Individuals

The individuals in a KBSgraph are the information content that satisfies the metadata of the class expression. Populating or identifying the individuals that belong in a KBSgraph is a three step process:

1. Identify the individuals that are in the root classes.
2. Filling out the other individuals by following the object properties from domain to range and assigning the individuals to each individual's object property.
3. For each individual, assign its data object data-type values.

The first step finds the roots of the KBSgraph, and the remaining steps find the connected individuals and their data values. During these steps, individuals only appear once in the KBSgraph no matter how many classes they are assigned to.

Individuals may be members of one or more of the classes defined in the ontologies and in the metadata. OWL and the KBSgraph therefore supports multiple-inheritance.

If an individual does not have a value for a property, the value is given as null. This distinguishes missing values from default values.

Root Objects The root objects are the direct results found by executing an OWL class expression. These individuals are qualified by the class expression as having met all of the restrictions of the class expression.

Object Property Objects These individuals are those in the range of an object property for a specific individual in the domain of the object property. There are no other restrictions applied, even if the class expression did include additional restrictions.

Data Property Values The actual values of each individual's data properties are assigned to each individual. In the prototype framework, a data property has only one data-type.

3.4 The KBSgraph is the Cornerstone

Everything in the framework aligns with the KBSgraph derived from the OWL class expression. It is the basis for repository query and update and the foundation for service messages and socket descriptions.

3.4.1 Repository Query and Update

Service Component Architecture (SCA) includes a query and update standard, called Service Data Objects (SDO). The SDO standard is applied by the framework to query and update the ontology repository of individuals.

Service Data Objects The SDO standard defines the processes to create, read, update, and delete information in graph structures. The framework provides the implementation for the Data Access Service for creating a SDO data graph and all other objects used by the standard.

SDO was created as an abstraction that would be closer to the real world usage of data. This frees the programmer from the complexities of the underlying data repository. This abstraction is also almost the same as the abstraction of knowledge of OWL, and therefore makes SDO a good approach for accessing ontology information.

OntoDAS In the framework, the Data Access Service (DAS) used with OWL is given the name of OntoDAS. The "onto" naturally refers to ontology. Providing a DAS is the standard approach for implementing various forms of information structures to SDO.

OntoDAS is an object with methods to create an SDO data graph from a class expression. The data graph can then be loaded with

individuals from the ontology repository. Individuals can be added to the data graph, deleted from the data graph, or updated in ways where updates may change the values of any data or object property.

In the SDO approach to preventing overlapping updates to information, the originally retrieved data from the repository is compared to the information in the repository at the time of the update. If there has been no change, the update is allowed. If it has been changed, the update is disallowed.

In addition to the standard SDO approach to verifying changes, the OntoDAS invokes the ontology repository's reasoner to verify changes. If changes fail any axiom, the changes are rejected.

Data Graph The data graph object of SDO is a mapping of the KBSgraph. SDO has classes with object and data properties the same as a KBSgraph. A SDO property can have one or more values, as does the KBSgraph.

It has all of the methods defined in the SDO standard for working with graphs and controlling the update process. It does have some differences, such as there may be more than one individual in a root class. There may be more than one class in the root. Also, multiple-inheritance is supported in the framework support of SDO.

Data Object A data object in SDO is the same as a named individual in an ontology. This means that a data object has object properties and data properties just like an OWL individual.

Data objects can be created from a root class or from an object property. Once created, values can be set. Data objects may also be deleted. All of these changes are recorded in the standard SDO change log. They are applied to the repository by the update method of the DAS.

Object Properties In SDO, Object properties are assigned to a class as having the property and other classes are defined as being the subject of the property. This is the same structure as the KBSgraph

metadata. The class assigned the property is in the domain, and the subject classes are in the range.

Data Properties Data properties contain values in SDO in the same manner as in a KBSgraph. The values are of data-types defined in OWL and mapped to SDO data-types.

3.4.2 Service Messages

Service messages are KBSgraphs. They are formed from an OWL class expression that describes the content of the message.

All messages are asynchronous. This means that when a message is sent, the only initial response is whether the message was received. The sender of the message may continue on with other processing or wait for a response.

In SCA as implemented in the framework, a service can receive a request message and return a response message. A referencing component can make a request using a send message with the results returned in a receive message. So a referencing component has a send and a receive message, where a service component has a request and response message.

There are only two types of services that may be referenced by a component:

- Request / Response Services
- Request Only Services

A component references a service with a send and may have a receive message. As a result, there are only two types of communications that can take place:

- A Reference Send to a Service Request
- A Service Response to a Reference Receive

In each of these communications, there must be some root ontologies used by both the sender and the receiver. This provides the commonality needed for a receiver of a message to understand the content of a message.

Message Root Ontologies For the communications to work, both the sender and the receiver must have access to the root ontologies that define the messages. The actual classes used in a message do not need to be the same on both ends of a communication, but the classes must be understood as either sub-sets, super-sets, or equivalents.

This implementation of the framework to recognize root ontologies offers great opportunity for expanded functionality of services over time without losing current functionality.

Reference Send, Receive, and Correlation From the perspective of a referencing component, the important messages in the service communication are send and receive. The send message, filtered through the root ontologies, populates the service's request message. When the service completes its process, it constructs a response message. The response message is then filtered through its root ontologies and returned as a receive message.

A reference component may also have a correlation message associated with a send message. This provides a means for the referencing component to distinguish multiple receive messages from the same service. When a referencing component is notified that a receive message is available, it is associated by the framework with the original correlation message.

Service Request and Response From the perspective of a component providing a service, the important messages are request and response. In the framework when a request message is ready for processing, the component providing the service is initiated as a new

task to run asynchronously. Once that task completes, it returns a response message.

3.4.3 Socket

A socket contains the metadata of one or more services and references in the form of KBSgraphs. Any external entity plugging into the socket is given full access to this metadata.

For people plugging in through the web, the framework provides functionality to create messages and receive messages. For other servers utilizing the framework, plugging into a socket allows for a full knowledge of integration metadata for information processing.

Services In a socket, services should be provided by the entity plugging into the socket. The framework supports messages directed to any entity that plugs in or only to entities targeted in the message. For example, and entity might receive the next service request from a component needing a result from any analyst or an analyst may receive a message targeted specifically for the analyst.

References References in the socket are the services that the entity plugging in can access. The entity would send a message from the reference and then wait on a response from the service.

Metadata The metadata describing the classes and properties of the services and references are fully disclosed in the socket. The metadata is defined according to the ontologies made available to the socket component.

3.5 Simplicity

The simplicity of the framework is found in the class expression. There is only one language used to define both queries and messages. Those that learn how to use the class expression enhance their ability in all

areas. They move closer to understanding the knowledge upon which the queries and messages rely.

The KBSgraph cornerstone, upon which all aspects of the framework are aligned, is more than a format for information. It is a simplified view that reduces the complexity of accessing and processing information within the fullness of knowledge.

Chapter 4

The OTTER Executable Ontologies

The OTTER prototype framework provides ontology execution for business, services, and processes. The business ontology is an upper-ontology, which defines a basic structure that may be expanded by other more business-domain specific ontologies. The service ontology defines Service Component Architecture, and the process ontology defines how to implement a formal process.

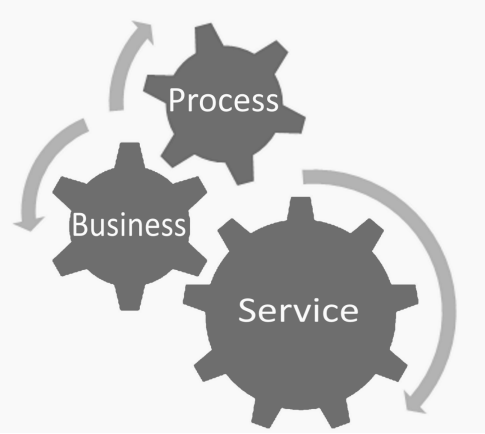


Figure 4.1: Framework Execution

As shown in Figure 4.1, the three primary ontologies of the OTTER framework operate together to provide the functionality to execute KBS applications. This chapter describes how this works.

4.1 SCA

Service Component Architecture (SCA) is a proven architectural approach to constructing large complex systems. The framework provides this capability through SCA and BPEL with the ontologies SCAlite and BPELlike, respectively.

4.1.1 SCAlite

The SCAlite ontology defines the individuals that make up the executables of Service Component Architecture (SCA). Figure 4.2 shows the main classes and properties of this ontology.

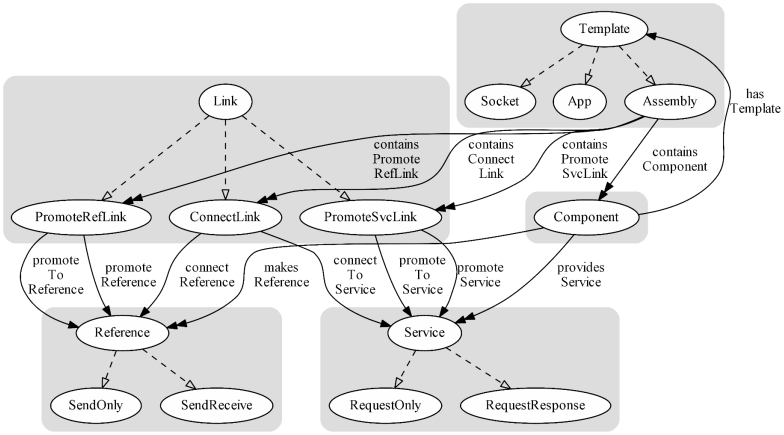


Figure 4.2: SCAlite

Components Components provide services and make reference to services. They also have a template that defines how the component is implemented. Components may provide multiple services and make multiple references, but they can only have one template. A template for a component is defined as one of the following:

- **Socket** - External entities plug into socket components to reference services and to provide services.
- **App** - These components are implemented within application code.
- **Assembly** - This component is made up of other components, promoted services, promoted references, and links from references to services.

Each of these types of templates are defined in more detail later in this chapter.

Services Services are individuals defined using the SCAlite ontology. The ontology requires that the type of services be defined as:

- **Request only** - The service is passed a request message and executed.
- **Request / Response** - The service is passed a request message and a response message. The service is then executed and the response message is returned to the requester.

The SCAlite ontology also requires a service to be given a unique name within its domain, and that service messages are defined as KBSgraphs.

References References are individuals defined using the SCAlite ontology. The ontology requires that the type of references be defined as:

- **Send only** - The reference message is sent to a service without needing a response.
- **Send / Receive** - The reference message is sent to a service and the service is expected to respond with a receive message.

Just like services and service messages, the SCAlite ontology requires that a reference be given a unique name within its domain, and reference messages are defined as KBSgraphs.

Assembly An assembly component is an individual defined in the SCAlite ontology by combining two or more components. The services of an assembly are defined as promoted services, and the references of the assembly are defined as promoted references. Promoted services are selected services of the contained components. Promoted references are those of the contained components needed by the assembly for execution.

Links are defined for the purpose of connecting the services and references within an assembly. The following links may be defined:

- **Connect Link** - Component references to service connections may be made.
- **Promoted References** - Component references may be connected to references of the assembly and therefore promote the reference.
- **Promoted Services** - Component services may be connected to services of the assembly and therefore promote the service.

Links are the wiring with which to construct an assembly, as shown in the example in Figure 4.3.

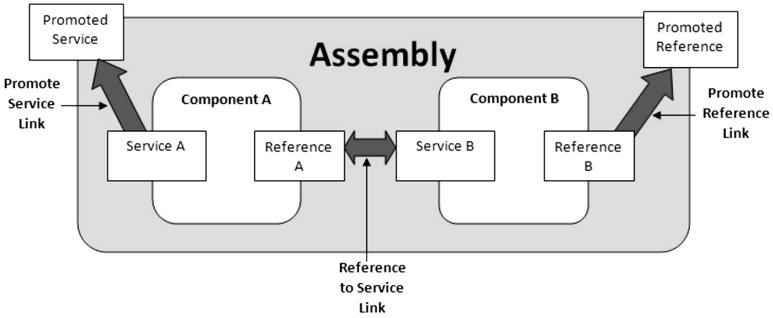


Figure 4.3: Assembly Links

In this example, “Service A” is promoted to “Promoted Service” by a link. “Reference A” of “Component A” is linked to “Service B” of “Component B”. And, “Reference B” is promoted to “Promoted Reference”. The ”Assembly” is now defined as a component with one service and one reference.

4.1.2 BPELlike

The Business Process Language (BPEL) is shaped around web-services and is referred to as WS-BPEL. The framework does not use web-services. All messages are defined using KBSgraphs, which are technically serialized using JavaScript Object Notation (JSON) and transported using Representational State Transfer (REST).

The BPELlike ontology defines commands found in BPEL. All the commands are subsets of the Activity class. Some commands are subsets of the Activity Block class which is also a sub-class of the Activity class.

Complex data structures are supported in the BPEL ontology through the use of data graphs, which are defined as KBSgraphs.

Activities The BPEL activities supported in the framework include:

Activity	Function
Assign	Set values in variables and data graphs.
DataGraph	Create a data graph or query the repository.
Terminate	End the process.
ThrowFault	Throw a fault to be caught by an activity block.
Variable	Create a simple variable to store values.
Wait	Wait for a defined period of time.

Activity Blocks The BPEL activity blocks supported in the framework include:

Activity Block	Function
CatchAll	Catches all faults including technical failures.
CatchFault	Catch only specific faults.
Flow	Initiate multiple parallel activities.
If	Selects from alternate activities to execute based upon a condition.
Invoke	Execute a reference to a service.
OnRequest	The first activity executed on receiving a service request.
Pick	Select from a group of activities to execute based upon a value.
Update	Create a data graph to update the repository.
While	Execute an activity chain repetitively while a condition exists.

Order of Execution The order in which the commands are executed is controlled by the activity links defined. A link is an object property from one Activity to another.

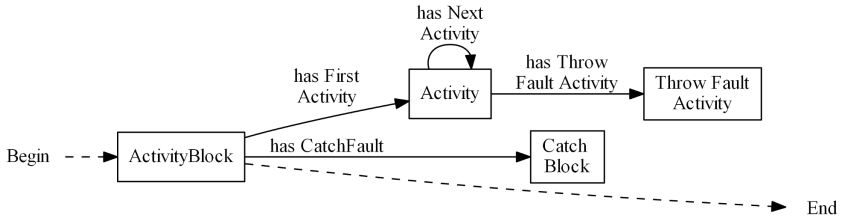


Figure 4.4: BPEL like Control Flow

The flow of control in a BPEL process is shown in Figure 4.4. A process begins by executing an activity defined by the class Activity-Block, a subset of Activity. Executing an activity block links to the first activity of the activity block, and continues with the links of the activities. Activity links are like a chain that can be followed until there are no more links remaining, as demonstrated in the example

shown in Figure 4.5.

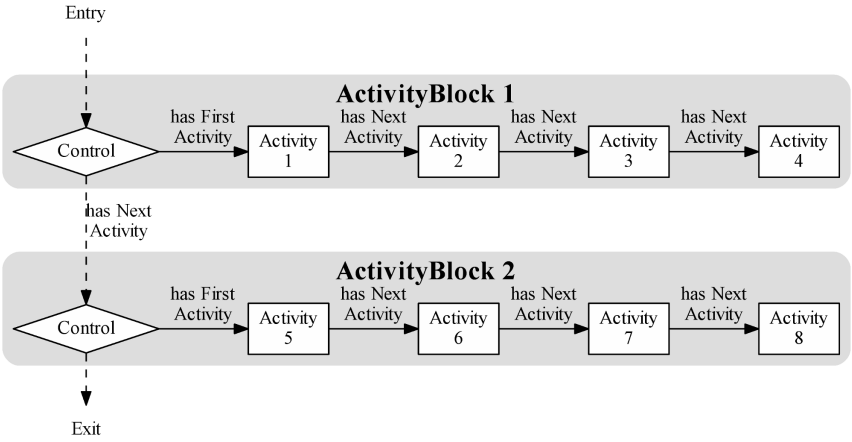


Figure 4.5: BPELlike Chain Example

In this example, "ActivityBlock 1" is executed. The first step in its execution is to perform the activity that is in the range of its "has First Activity" property. This activity, "Activity 1", does its function and then links to the activity in the range of its property, "has Next Activity". This is repeated for Activities 2 and 3 and continues on until "Activity 4", which has no value defined for "has Next Activity". At that time, execution is returned to the activity block's control and execution continues with the activity in the range of the activity block's "has Next Activity" property. The next activity is "ActivityBlock 2", which repeats the same chaining process for Activities 5 through 8 and then exits the execution since Activity block 2 does not have the property of "has Next Activity".

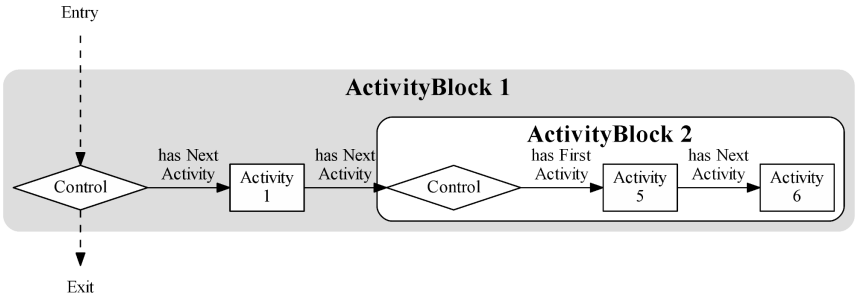


Figure 4.6: Activity Block Example

An activity block chain of activities may contain other activity blocks which also have their own chain of activities, as shown in Figure 4.6. This can form a hierarchy of active activity blocks. Any activity can throw faults, and the first activity block going up in the hierarchy that has a catch defined for the faults is executed.

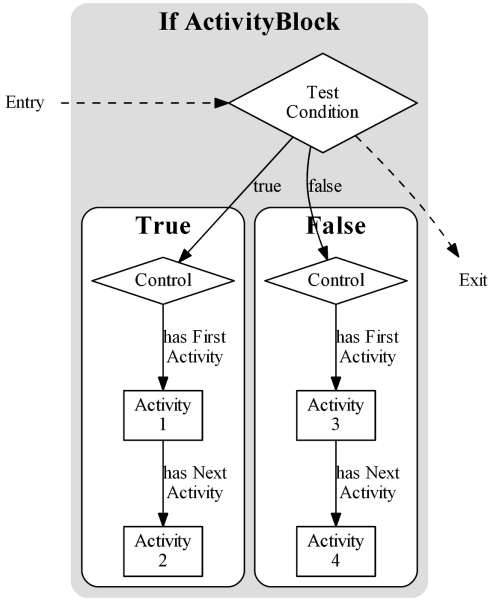


Figure 4.7: Activity Block If Example

As an example of how the framework utilizes activity blocks, Figure 4.7 shows how the BPEL "If" statement is implemented when it contains two blocks. The true block is executed if the condition is true, and the false block is executed if the condition is false.

This method of implementation is used by the framework for any BPEL statement that selects from multiple logical paths.

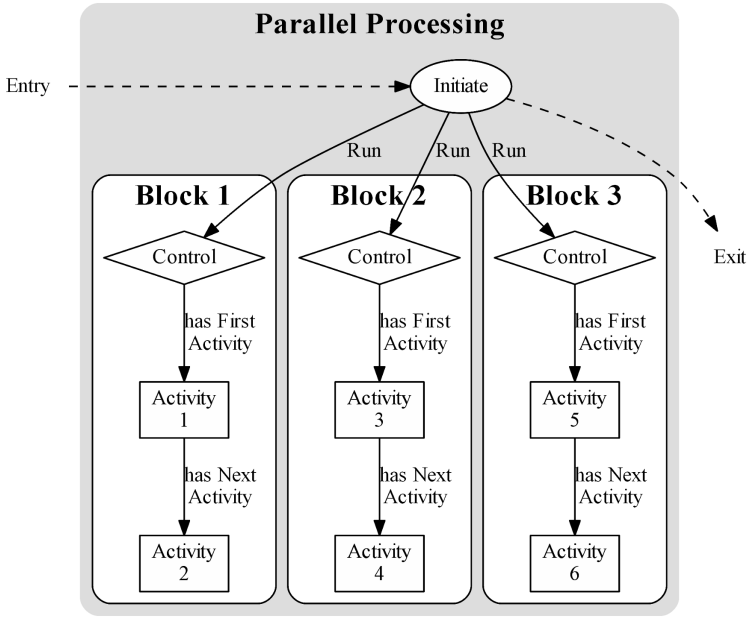


Figure 4.8: Parallel Processing Example

Another example shown in Figure 4.8 is parallel processing. This allows an activity block to initiate multiple blocks at the same time. The framework uses this method to implement BPEL statements that run parallel processes.

A full description of the classes and properties can be found in "Appendix B".

4.2 Business Model

The primary purpose of the business model is to separate internal computer processing from all other forms of information processing. Internal processing is done by computer applications executing within the framework, and other forms are performed external of an instance

of the framework by people or other instances of the framework.

The business model provides a structure for business components. These components include organizations, persons, and software as the means to show the assignment of responsibilities for performing services, as well as the resources needed from other business components.

4.2.1 Class Structure

All classes are derived from the "BusinessComponent" class as shown in Figure 4.9.

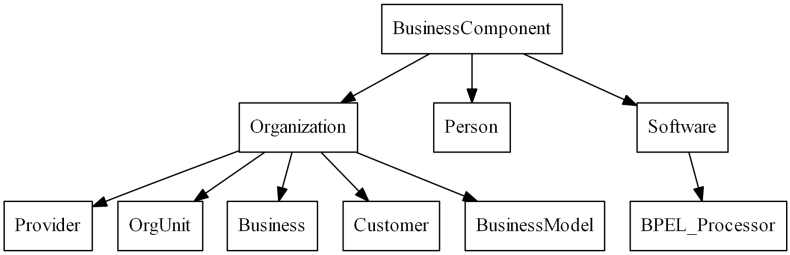


Figure 4.9: Business Model Class Hierarchy

Organization The "Organization" class has subsets that provide structure to the business model. The subsets are:

- **Provider** - An external source of resources.
- **OrgUnit** - An organizational unit usually found on a business organizational chart.
- **Business** - A representation of a specific business that provides services to customers and consumes resources from providers.
- **Customer** - An external consumer of business services.
- **BusinessModel** - An assembly that displays the flows between the business and its providers and customers.

Person The "Person" class specifically refers to an individual with an identity and security access to sockets. In the framework, identification is performed by name and password, and sockets are selectively linked directly to each person.

Software The "Software" class describes the implementation of an application. Implementation includes the name of the library that contains the application and the name of the application. The software may also be defined as a BPEL process.

4.2.2 Class Relations

In addition to the class structure, there are key object properties in the business model as shown in Figure 4.10.

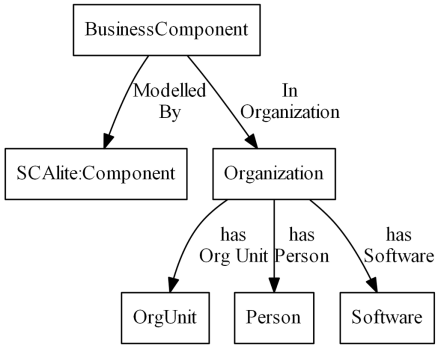


Figure 4.10: Business Model Class Relations

Business Component The "BusinessComponent" class is "Modelled By" a component in the SCAlite ontology. Consequently, a business component can be a model having a template of a socket, an application, or an assembly.

Organization The "Organization" class provides structure to an organization's responsibilities and ownership.

Organizational units (OrgUnit) follow the typical structure of a business organization, where the business is made up of multiple organizations and each of those organizations may be made up of other smaller organizations.

People make up organizations. These include both business, customer, and provider organizations where the people have responsibilities to provide services, and also have requirements for consuming services.

Organizations are responsible for software, as they are the providers to those that use the software. The intent in this model is that these organizations make all of the business decisions about the services provided by the software.

4.3 Service Execution

The framework controls the execution of services defined in applications or sockets.

4.3.1 Applications

Application code is standalone code that provides one or more services and makes one or more references. The application communicates with the framework through parameters passed by the framework to the application. By importing framework packages, the application has access to framework functionality and the library of classes for using Data Service Objects.

The framework only loads the application code once to make a unique instance, and then it passes control information to the "init" entry. The service entry is then located in the code by the name of the service.

The application code defines a method called "init", and a method for each service provided. The control information passed to the "init" entry of the code includes the access to framework functions. These functions include:

- **OntoDas** - The Ontology Data Access Service supporting the creation and updating of data graphs.
- **SocketControl** - References can be located, parameters retrieved, and other information (such as the name of the originating requester and namespaces) is exposed.

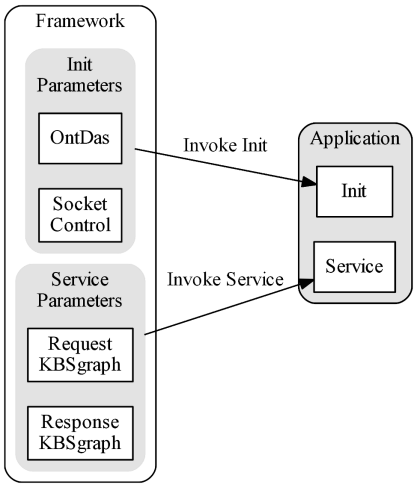


Figure 4.11: Application Execution

The execution of an application service is shown in Figure 4.11 passing parameters to the "init" method. The service entry is shown as passing the request KBSgraph and the response KBSgraph as parameters in the standard form of a method call.

4.3.2 Sockets

Sockets are uniquely initialized for each entity that plugs in as part of a communications session, as shown in Figure 4.12. When the session ends, the unique socket initializations are discarded.

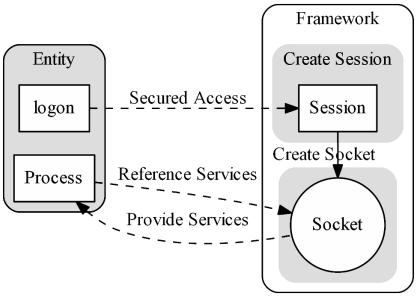


Figure 4.12: Socket Execution

The socket initialization resolves all links for services and references.

4.3.3 Asynchronous Communications

All service requests are asynchronous. The framework creates a reference control to manage the flow of control in executing a service. The reference control executes services using a thread different from the requester, then queues the response on completion, as shown in Figure 4.13.

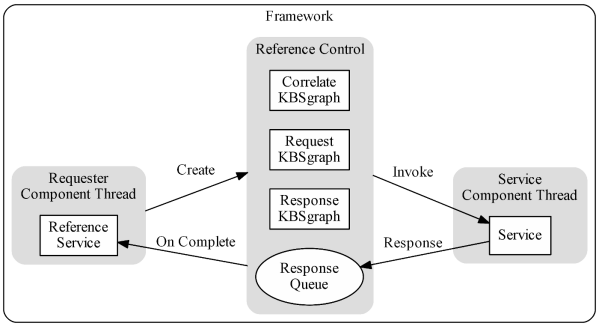


Figure 4.13: Asynchronous Communications

The reference control is made available to the requester to provide the methods of communication and the information for processing. The methods include setting the KBSgraphs for sending, receiving, and correlating, as well as testing for completion or waiting for completion. It also contains information on errors when they occur.

4.3.4 Conversational Communications

Services may have conversations with the original entity making the request. This is possible since sockets provide services as well as make references.

A service may have one or more references linked to socket services. This allows services to invoke references to socket services for conversational purposes.

4.4 Security

Security is integral to the framework, rather than a separately managed function. People's access to the server is controlled by the identification defined in the business ontology, and by a server initialization ontology that describes a valid login request. The services a person can reference is determined by the sockets to which they are given access. The information accessible by applications is determined by its domain's ontology access, which is controlled by the OWL import statements.

4.4.1 ID and Password

The framework supports the basic form of access over the Internet. This requires a person to have an identification name and a password. To protect the passwords, they are stored in an implementation ontology only accessible to the framework.

4.4.2 Composite Sockets

A single person may have more that one socket assigned to them. The framework combines all of the sockets assigned into a single composite socket, as shown in Figure 4.14.

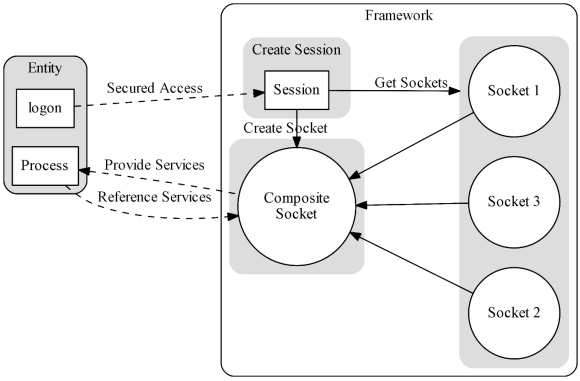


Figure 4.14: Composite Socket

When an entity is given secured access, a session is created and all of the sockets available to the entity are combined into a single composite socket. The entity can then access the composite to determine the services and references and their KBSgraph requirements.

4.4.3 Scope of access within OWL imports

An OWL document can import other OWL documents. This capability allows an ontology to access and expand upon the content of other ontologies. It is also restrictive in that it only has access to the ontologies that are directly or indirectly imported.

Directly imported documents are those included by a primary OWL document. Indirectly imported documents are those included by a secondary OWL document, which itself was directly or indirectly imported by another document.

The recommendation in using the framework is to separate the implementation ontology of a domain from the metadata ontologies of the domain. The implementation ontology would contain the individual content which can be thought of as the repository of data. When this is done, domain ontologies may include metadata ontologies of other domains, but may not include the implementation ontology where the individual content resides.

As recommended, this provides a secure separation of metadata access versus domain content access. This access restriction exists for both sockets and applications.

4.5 Message Flow

The framework manages the flow of information from references to services. This includes establishing the links between the references and services, and providing an ontology-based method of integration between domains. It also provides the identifications needed to track the flow of the messages.

4.5.1 Reference / Service linking

The linking of references to services is not dynamic. Each link is specifically defined in an SCA assembly.

To visualize the flow of information within a business, the business ontology classes may be assembled into a hierarchical form of assemblies with the business at the pinnacle. To simplify the visualization at higher level assemblies, flows can be combined at the sub-assembly level and named according to their sub-domains. The flows and the assemblies could be drilled into for more detailed levels of information.

Although the linking is not dynamic, a service may have multiple implementations. In this case, the framework follows a round-robin approach to assigning a reference to a service from a list of services. This is done dynamically at execution time.

4.5.2 Reference / Service integration

The integration of a reference to a service is based upon common knowledge. A domain referencing a service of another domain must use the language of the service domain.

The framework merges the KBSgraphs of the reference with that of the service for identical class and property definitions. Service KBSgraphs would be defined using definitions from its metadata. Reference KBSgraphs may be defined using definitions for the service metadata, or define a mapping of the service domains to the reference domain.

Merging KBSgraphs The framework merges KBSgraphs by finding the intercept of classes and properties between two KBSgraphs as shown in Figure 4.15.

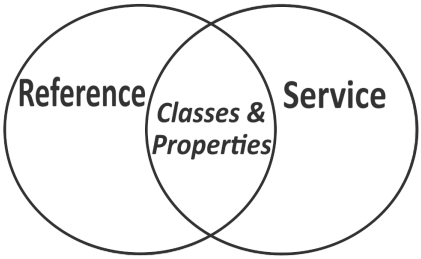


Figure 4.15: Merging KBSgraphs

These merges take place at execution time, so only the classes, properties, and individuals that have the common classes and properties are included in the messages as follows:

- **Send / Request** - The send KBSgraph of the reference is merged with the KBSgraph of the service request. Qualifying classes, properties, and individuals from the send KBSgraph are inserted into the request KBSgraph.

- **Response / Receive** The response KBSgraph of the service is merged with the receive KBSgraph of the reference. Qualifying classes, properties, and individuals from the response KBSgraph are inserted into the receive KBSgraph.

Service Domain Metadata The request and response messages of a service are defined by the service's domain metadata. In order for a reference send and receive to access the service, the reference domain must include the ontology in the domain document.

By including the metadata from the service domain, the reference domain has access to the classes and properties of the service messages. The service domain can expose its service message metadata using multiple approaches. The selection of an approach would be dependent upon future change and the amount of metadata to be exposed to referencing domains.

The selection of how a service domain provides the metadata for its services is a design issue, not a technical one. The framework performs the merge of KBSgraphs in the same manner no matter what method is chosen.

Reference Domain Mapping A reference domain may directly use the metadata provided by a service domain, or it may use a mapping approach. In a mapping approach, the reference domain utilizes OWL to define classes and properties as equivalent. Defining equivalents essentially gives two different names for the same thing.

Mapping provides the reference domain the ability to:

- Use names that are more appropriate to the referencing domain.
- Minimize change if the service provider is replaced by another.

Whether a reference domain utilizes mapping is a design issue. The framework will perform merges with or without equivalents.

4.5.3 Request identification

There are two forms of request identification provided by the framework:

- **External** - When an external entity makes a service request through a socket, its identification is tracked. The tracking occurs for initial service requests and for service requests made by those services.
- **Internal** - The requester of the service is also tracked. This identifies the SCA component making the request.

Tracking provides the benefits of an application to provide specific content security. Also, it provides the information needed to support detailed billing of services, and it may be useful in finding problems.

4.6 Summary

The ontologies of the OTTER framework include business, services, and process. Services is the primary ontology for framework implementation. The service ontology is used by the business ontology to define the business components as either providers or consumers of services. The process ontology defines a set of commands that are supported by the framework and operate within a service environment.

The framework provides the execution for these ontologies upon which KBS applications can be developed.

Chapter 5

Reality and Imagination

In the framework, the only reality a KBS application knows exists in the executable ontologies. These ontologies are documents that capture human understanding. They allow us to explain how things work together within and across domains for KBS application consumption.

Imagination is how the known parts are brought together into a visualization that allows humans to understand the knowledge contained in KBS applications. Much of this imagination will go towards understanding specific domains that may already have a means of visualization.

Visualizations peek into the reality of KBS applications. They are not blueprints of something to be constructed. They are snapshots of what has been constructed and currently operating within the framework.

Computer applications in the framework are not directly visible. They are more in the form of an explanation of how we believe things should be. They are metaphysical in nature. Any visualization is in the mind of those that have an understanding of the metaphysics embodied in the reality of the application.

In this chapter, reality is the source of all visualization. Imagination is limited to current methods of viewing data and processes. Consequently, this chapter is limited to static screen-shots of visual-

izations using the JavaScript libraries of D3.js and X3dom.

5.1 The Challenge of Visualization

A challenge exists as to how humans can see the knowledge captured in the domains. Since OWL and the framework ontologies are the forms in which the knowledge takes, the visualization must come from these sources. As such there are many approaches that are in place today to visualized OWL, SCA, and business processes in graphical forms. The development of each of these approaches required imagination.

In my book, *Enterprise Architects Masters of the Unseen City*[1], the entire automation of an organization is imagined as a city where the integrated domains were buildings with service connections running under ground. It was all done in 3D so anyone could walk the streets and enter any building.

The emphasis is that computing systems are real. We know all the software parts. We only need to assemble the parts into a meaningful visual whole. This is the challenge.

5.2 Framework Visualization Architecture

The framework infrastructure ontology, SCALiteContent, provides services to access the content of OWL, SCA, and BPEL. The content of the composite socket is provided directly from the framework.

The content information provided is used by JavaScript processes in a browser to create HTML visualization of the components as shown in Figure 5.1. All information transfered from the framework to and from the browser is serialized using JavaScript Object Notation (JSON).

[1]Tinsley, Thomas A.,*Enterprise Architects Masters of the Unseen City*, 2009, Tinsley Innovations

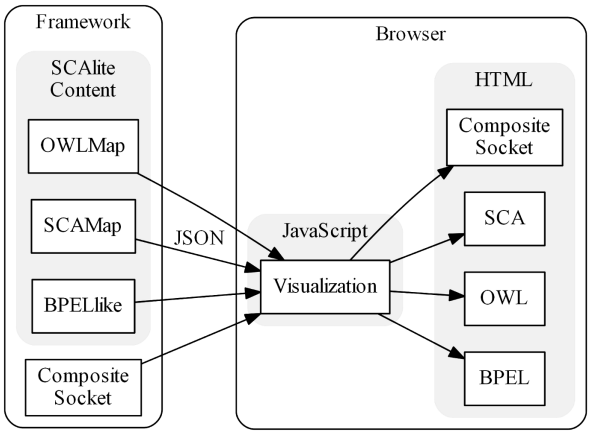


Figure 5.1: Browser Visualization

The framework includes the JavaScript components to support Service Data Objects (SDO). Utilizing SDO, the visualization components retrieve the information needed to produce diagrams to show generalities and detail.

5.2.1 Content Ontologies

Content ontologies describe KBSgraphs for accessing information on the structure and execution of ontologies. These ontologies are part of the framework infrastructure.

SCAlite Content This domain includes an application shown in Figure 5.2 supporting the following services:

- **GetOWLmap** - OWL structure
- **RetrieveComponentContent** - Services, references, and component descriptions
- **ConstructAssemblyContent** - Content of an assembly

- **RetrieveConnectLinks** - Reference to service links
- **GetSCAmap** - Link performance information
- **RetrieveBPEL** - Process content

The response of each of these services is a KBSgraph.

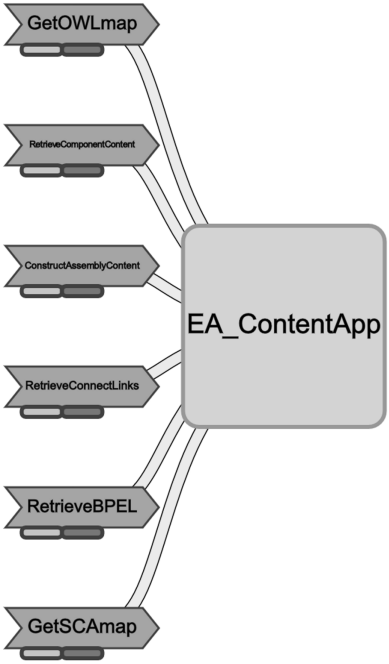


Figure 5.2: Content App

In this figure, each service is shown as an arrow connected to the application with the name of the service within the arrow. Below each arrow are two rounded corner rectangles that when clicked will bring up a view of the KBSgraph. The first rectangle is the KBSgraph for the request and the second is for the response.

Each of the KBSgraphs returned from the services are presented in this section.

Get OWLmap Service The OWLmap KBSgraph returned by service "GetOWLmap" contains the classes, properties, annotations, and data types used in each ontology in the framework as shown in Figure 5.3.

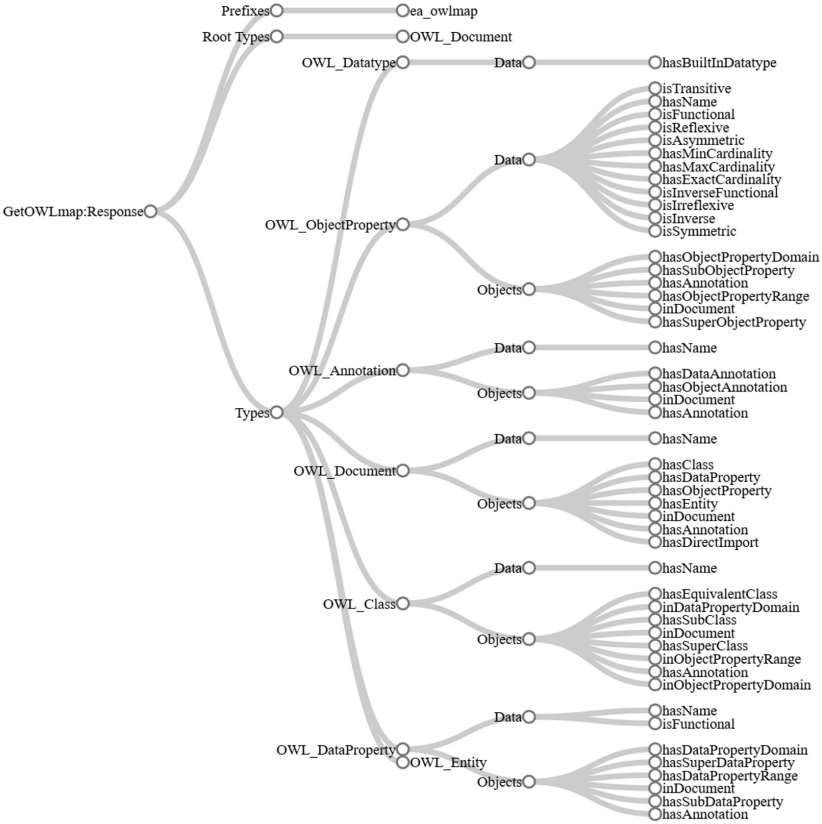


Figure 5.3: OWL Map

The OWLmap is a specialized content produced by the framework for accessibility by applications.

Retrieve Component Content Service The component content KBSgraph returned by service "RetrieveComponentContent" contains the services, references, and component description as shown in Figure 5.4.

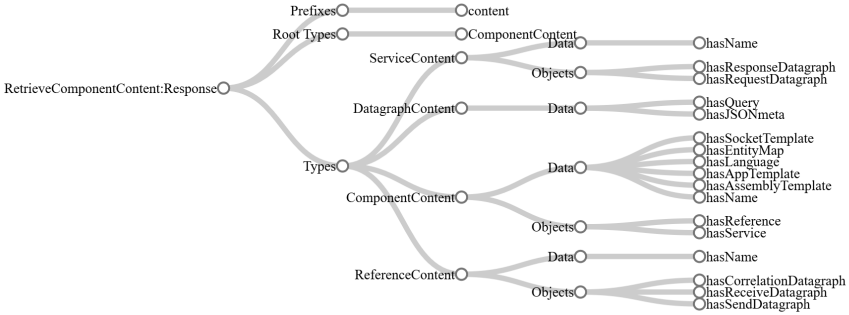


Figure 5.4: Component Content

The content information is retrieved using the framework's standard method of accessing graphs.

Construct Assembly Content Service The assembly content KBSgraph returned by service "ConstructAssemblyContent" contains the contents of an assembly including the components, services, references, links, promoted services, and promoted references as shown in Figure 5.5.

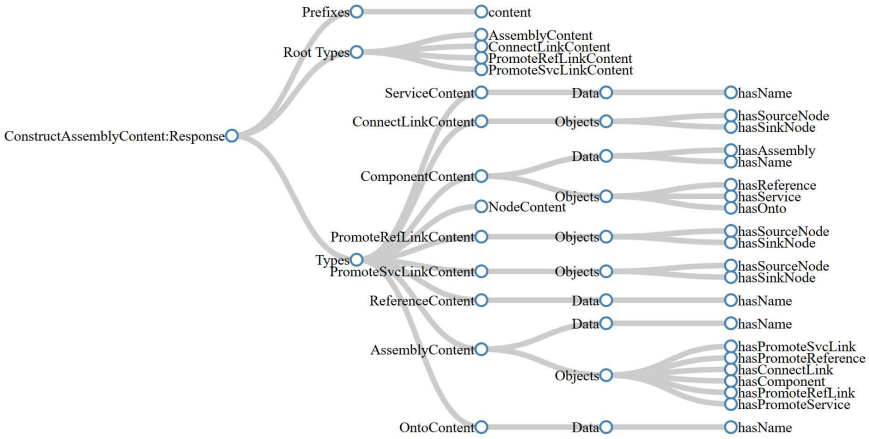


Figure 5.5: Assembly Content

The assembly content information is also retrieved using the framework's standard method of accessing graphs. It contains all the parts of the assembly.

Retrieve Connect Links Service The connect links KBS-graph returned by service "RetrieveConnectLinks" contains the components linked by references to services as shown in Figure 5.6.

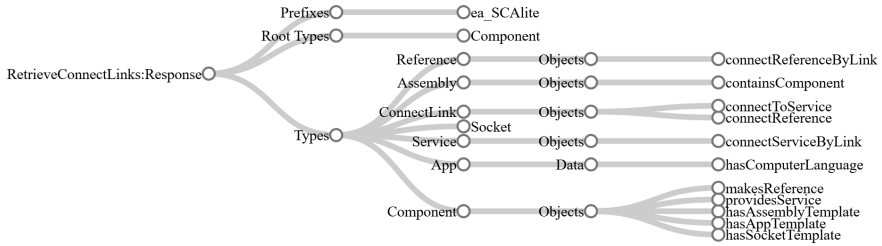


Figure 5.6: Content Links

The connect links show the references connected to services and references to services.

Get SCAmap Service The KBSgraph in the response to the "GetSCAmap" service provides performance data on the links between references and services as shown in Figure 5.7.

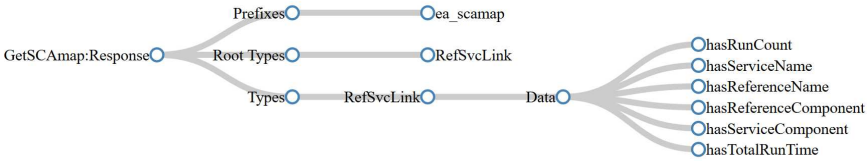


Figure 5.7: SCA Map

The links information is produced by the framework and accessible to applications.

Retrieve BPEL The service "RetrieveBPEL" returns a KBSgraph containing all of the steps of a process. There are sixteen unique processes. The diagram is sectioned into three parts as shown in Figure 5.8, Figure 5.9, and Figure 5.10. Although the diagram is difficult to read in the figures, the actual display shows a long web page that is easily read.

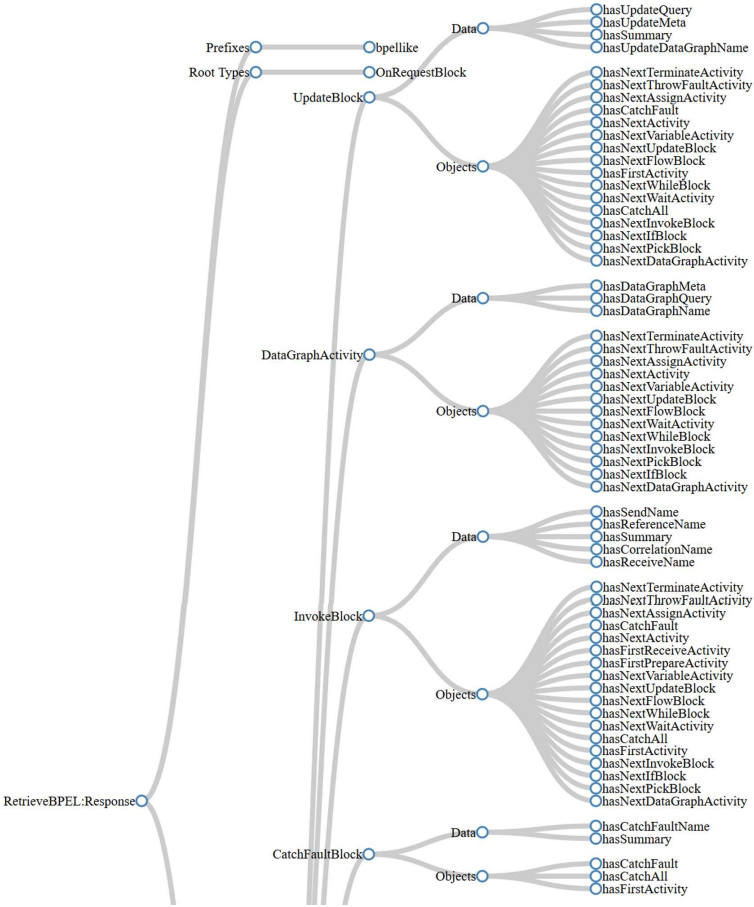


Figure 5.8: BPEL Content

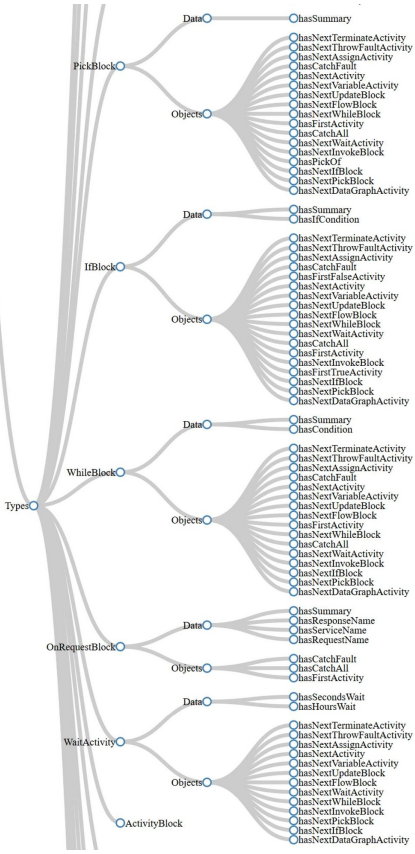


Figure 5.9: BPEL Content

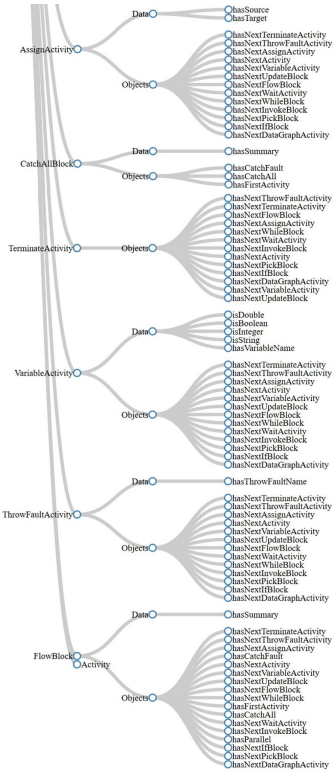


Figure 5.10: BPEL Content

This content provides the information on each step of a process and the steps of its sub-processes.

5.2.2 JavaScript

To provide visualization using a browser, the framework relies on JavaScript. It uses JSON for serialization of messages, provides support for SDO, and utilizes visualization components written in JavaScript.

JSON JSON (JavaScript Object Notation) is used for all message communications to and from the framework. This includes access to the composite socket and access to services and references.

JavaScript SDO The framework includes a JavaScript component that supports SDO by sending reference requests to services and receiving service responses in the form of KBSgraphs. The KBSgraph in JavaScript provides a direct access to the properties of each class.

The basic steps required to access a service is to execute methods of the SDO framework:

1. Add a prefix to each domain namespace to be used.
2. Get a reference to the service from the composite socket.
3. Create one or more root objects.
4. Add both data and object properties to the root and any other objects.
5. Invoke the reference.
6. Receive the response.

All service requests are asynchronous. This allows a JavaScript application to make multiple requests. The framework returns a unique number to each request. JavaScript logic can then retrieve a response to a specific reference or to any of the outstanding requests.

D3.js The D3.js software is an open source library for visualizing data. It can be found at `d3js.org`. On the website the following description is given:

> *D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS. D3s emphasis on web standards gives you the full capabilities of modern browsers without*

> *tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to DOM manipulation.*

This software package is utilized by the framework to produce many of the diagrams.

Digraph The Digraph JavaScript module is included in the framework to calculate the coordinates of components for simple diagrams. A digraph (directed graph) containing nodes and edges is provided as input to this module. The method applied by the module attempts primarily to minimize crossed lines for a more pleasing view. Coordinates are computed for x, y, and z for rendering in both 2D and 3D space.

This module is used to diagram SCA assemblies and BPEL processes.

5.3 OWL Structure and Content

In the industry, visualizing the structure of OWL documents is currently being done by multiple software tools. These are all excellent products and the framework does not intend to compete.

The framework includes a distributed, JavaScript based, approach for showing OWL structure where the focus is on the entire interrelated repository of ontologies. It takes an enterprise view.

5.3.1 Document Imports

The Sankey layout of the D3.js package is used to show the documents and their imports. The diagram is read from the left to right with flow lines on the right to each import. Since the same documents may be imported by several documents, the Sankey layout is very useful although sometimes very crowded.

To reduce the crowding in a single diagram, there is an option to select only the categories desired in the diagram. This is accomplished

by using the following filters and using the annotations provided in the framework's EA Pattern ontology as listed in the following:

Filtering Categories

- **Implementation** - Contains the individuals within the domain
- **Logical Pattern** - The metadata of the domain structure
- **Message Pattern** - The description of the SCA messages
- **Academic Discipline** - Principled based ontologies
- **Infrastructure** - Foundation domains

The Sankey layout also supports moving the rectangle depicting a document vertically to improve the visibility. When a rectangle is moved, it automatically redraws the flow lines.

Implementation Only The diagram in Figure 5.11 only shows implementation domains of the prototype test ontologies. This reduces the complexity significantly.

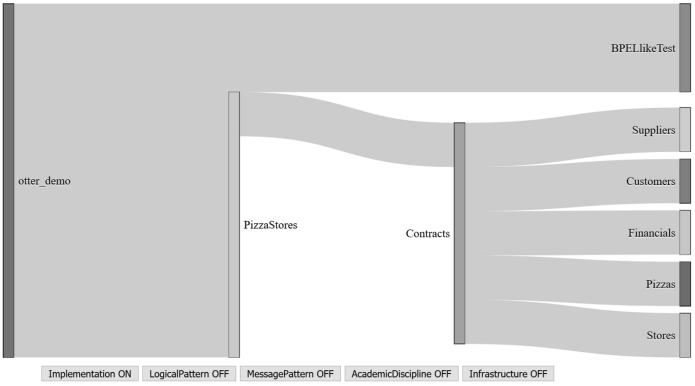


Figure 5.11: Implementation Only

As seen in this figure, the "otter_demo" imports a testing environment called "BPELLikeTest" and an environment called "PizzaStores" used to simulate the operation of running multiple stores. "PizzaStores" imports "Contracts" that defines the relationships between the imports of "Suppliers", "Customers", "Financials", "Pizzas", and "Stores".

Implementation, Meta, and Message In figure 5.12 the metadata and the message definition are included to expand on the diagram.

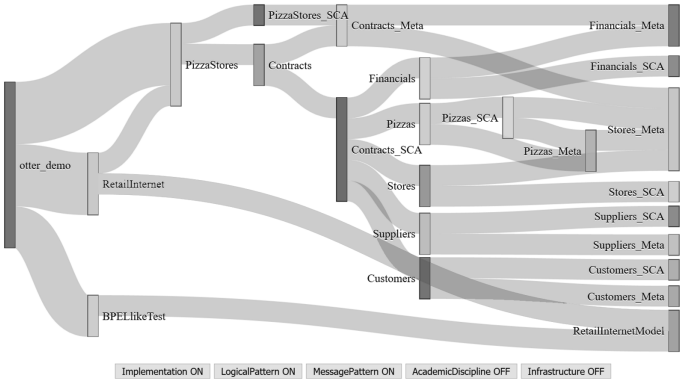


Figure 5.12: Implementation, Meta, and Message

This diagram reveals more of the complexity without overloading the content.

Mouse Over Document If the mouse is moved over a connection line, it will be darkened and a popup will appear describing the import. An example is shown in Figure 5.13.

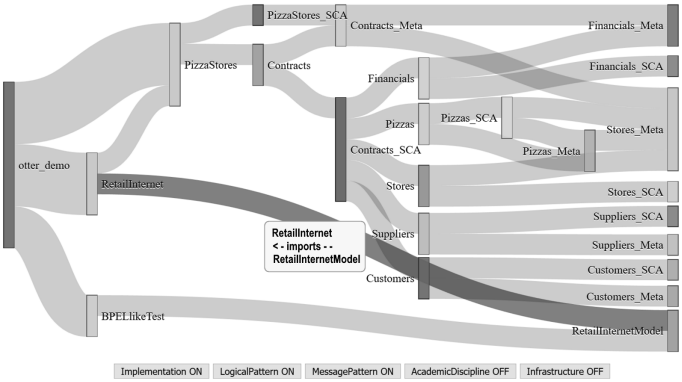


Figure 5.13: Mouse Over Connection

This ability is useful when flow lines overlap and it is difficult to tell what actual connections are being shown.

Without Infrastructure When everything is selected except the infrastructure, the diagram becomes difficult to follow as shown Figure 5.14.

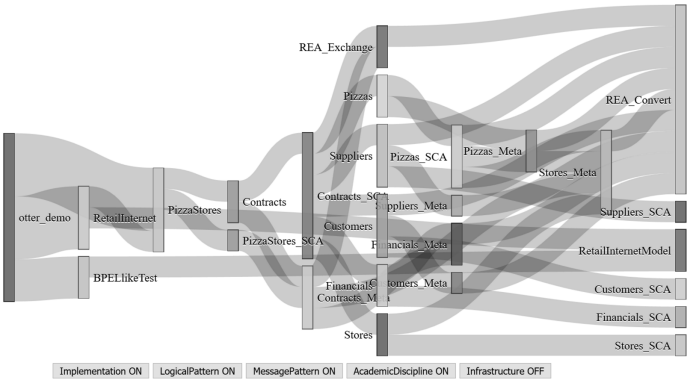


Figure 5.14: Implementation, Meta, and Message

Sometimes this can be useful in conjunction with the mouse-over capability to determine a particular import.

No Filtering When no filtering is applied, the diagram can get very difficult to read as in Figure 5.15. This is due the fact that all of the documents import one or more of the framework infrastructure ontologies.

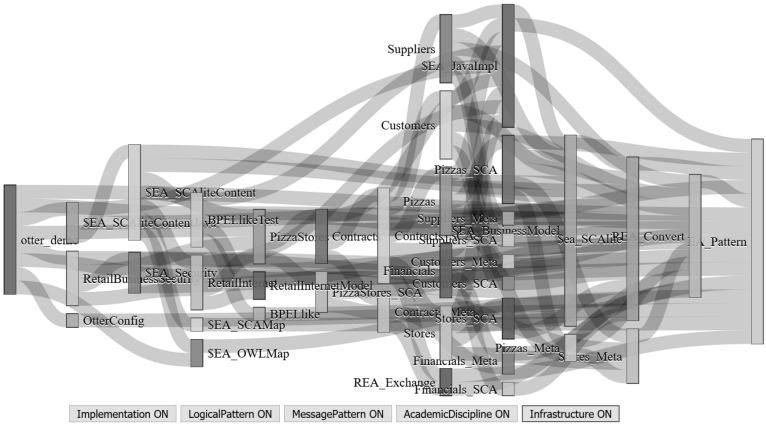


Figure 5.15: No Filtering

On the far left of this diagram is the initial document loaded, "otter_demo", and on the far right is the "EA Pattern" document imported by all the ontologies. The "EA Pattern" document contains the annotations used for identifying the layer of an ontology. This identification is used by the Sankey diagram to filter the ontologies.

5.3.2 Domain

By clicking the rectangle depicting a document on the Sankey diagram, another diagram is produced showing all the classes in the document. Then, by clicking on a class in this diagram the details of the class are shown in another diagram.

Classes When selecting a document, a Cord diagram is drawn showing the relations between all the classes within the document and classes outside the document. The Cord diagram offers the unique ability to view the links between classes in any direction as shown in the example in Figure 5.16.

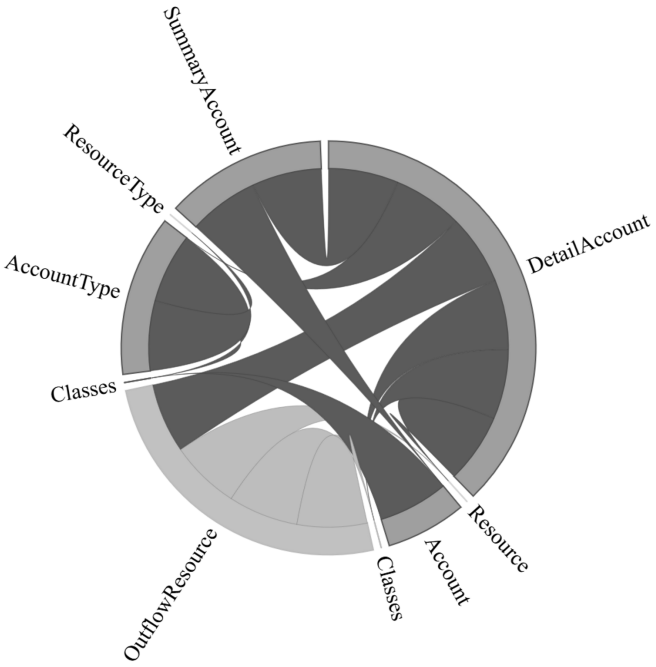


Figure 5.16: Classes

In this example, the classes having connections with each other are all shown in one color, but some also have relations with classes in another imported document. Within the same document, "DetailAccount" has a relation with "Summary Account". The class "DetailAccount" also has a relation with "OutflowResource" that is defined in an imported document.

Mouse-Over Class With many connections, the ability to place the mouse over a single class results in showing only the connections for that class as shown in Figure 5.17.

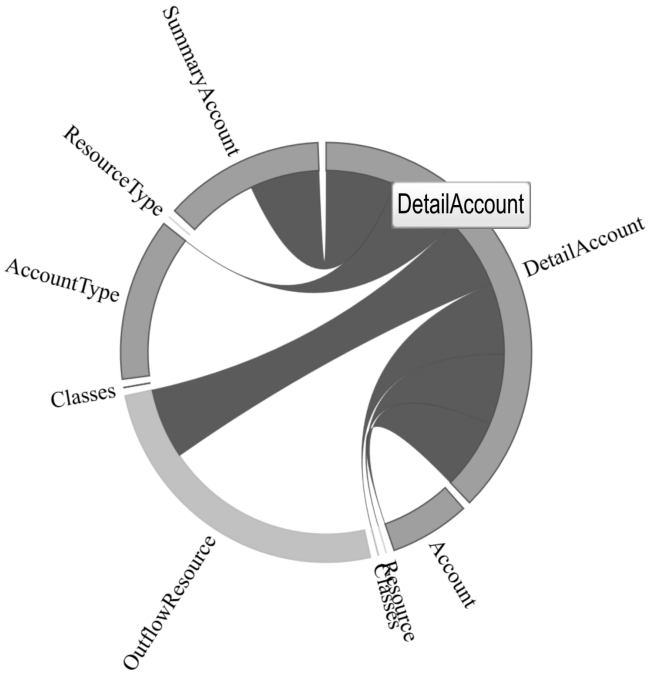


Figure 5.17: Mouse-Over Class

If the mouse is placed over a connection, it not only will show only the connections of one class, it will also provide a pop-up containing the property name and the connecting class. The mouse-over ability makes it easier to identify the relations of a single class.

Class Properties By clicking on a class, the details of the class are shown using a Cluster Dendrogram as shown in the example in Figure 5.18.

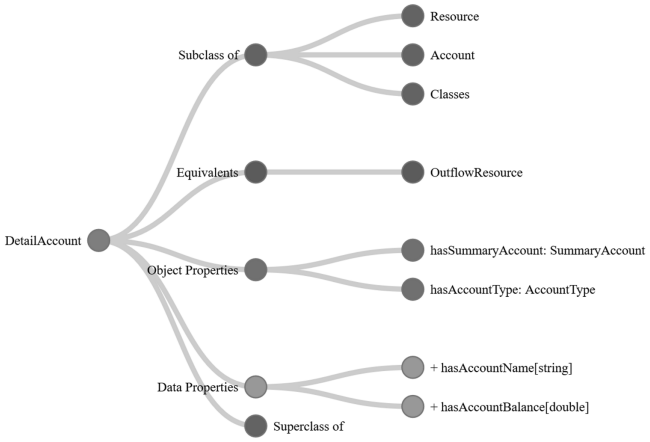


Figure 5.18: Class Properties

This diagram shows the following:

- Subclasses
- Equivalents
- Object Properties
- Data Properties

By placing the mouse over any of the detail items listed, more information is shown in a pop-up. For example, a mouse over a property will show any axioms defined for the property and the class of the property range.

5.4 Service Component Architecture

The visualization of SCA is from an enterprise perspective beginning with domains and having the ability to drill down into greater levels of detail. The OTTER test data domain is used as the example to show the dynamically created diagrams.

5.4.1 SCA Index

The index to all the domains providing services is visualized using a Cord diagram as shown in Figure 5.19.

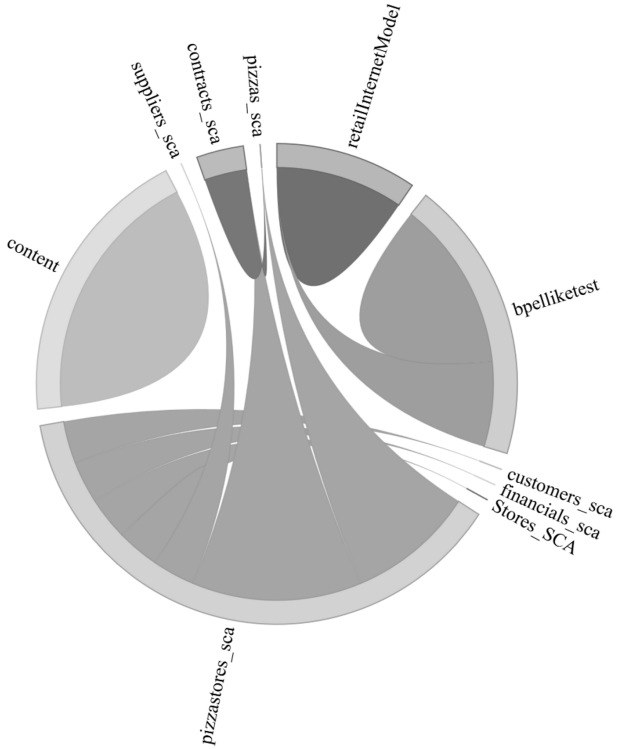


Figure 5.19: SCA Index

Each domain is given is a different color that is maintained through out all of the diagrams. The links shown are derived from reference-to-service links and component assemblies.

By doing a mouse-over a domain, its name and content are shown in a pop-up. Also, as shown in Figure 5.20 only the links to the selected domain are shown.

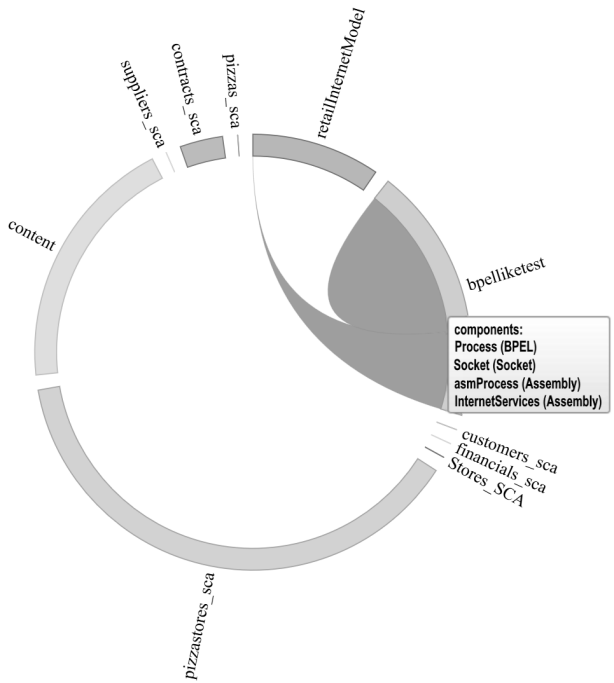


Figure 5.20: SCA Index Mouse Over

A mouse-over a link will bring up a pop-up that indicates the number of links between the domains.

5.4.2 Domain Component Links

Clicking on a domain in the SCA Index will show the component links within the selected domain as shown in Figure 5.21 using a Cord diagram.

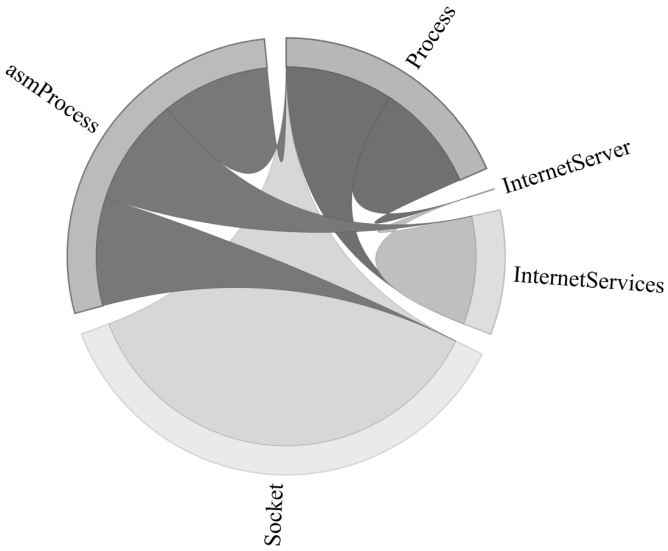


Figure 5.21: SCA Domain Components

All links are shown for the domain including links to components in other domains.

5.4.3 Components

The four types of components are visualized by generating a digraph of the component and its references linked to services. A mouse-over a component in a component links diagram will show a pop-up containing its name and type. It will also only show the links to and from the component.

Application When the component is an application, the pop-up will appear as shown in Figure 5.22.

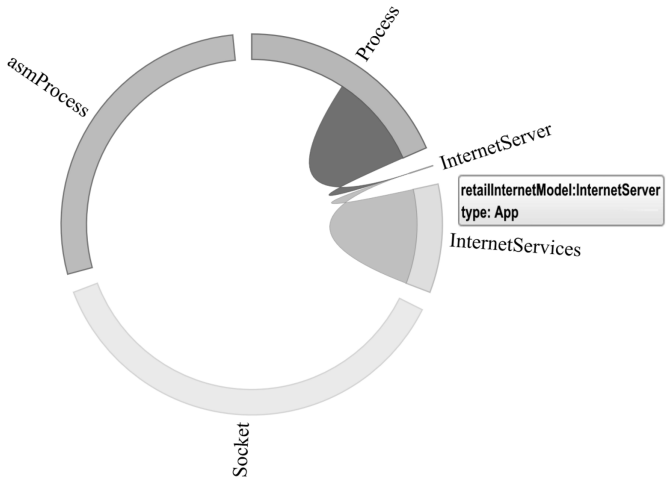


Figure 5.22: SCA Application Mouse-Over

If the component is selected, a diagram is produced to show the references and services of the application as shown in Figure 5.23.

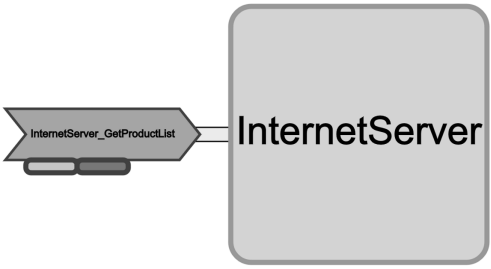


Figure 5.23: SCA Application

In this example, the test component "Internet Server" only has one service, "InternetServer_GetProductList".

Socket When the component is a socket, the pop-up will appear as shown in Figure 5.24.

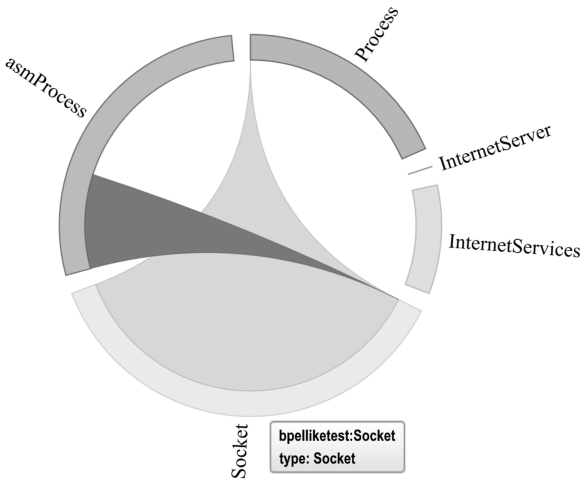


Figure 5.24: SCA Socket Mouse-Over

When this component is selected, a diagram is produced to show the references and services of the socket as shown in Figure 5.25.

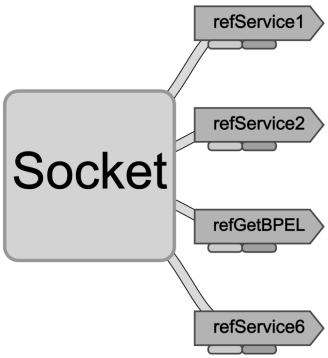


Figure 5.25: SCA Socket

In this test example, the socket references four services.

Assembly When the component is an assembly, the pop-up will appear as shown in Figure 5.26.

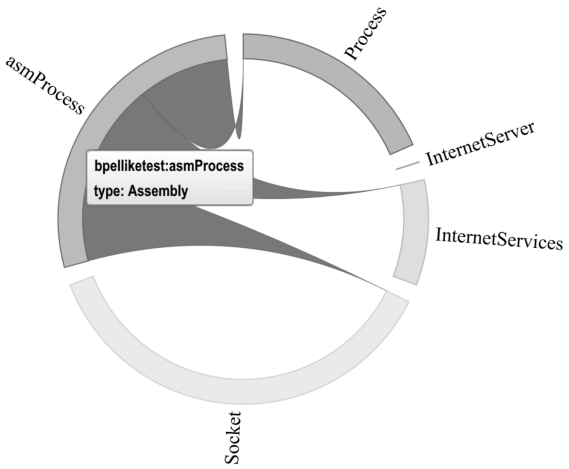


Figure 5.26: SCA Assembly Mouse-Over

If the component is then selected, the diagram in Figure 5.27 is shown.

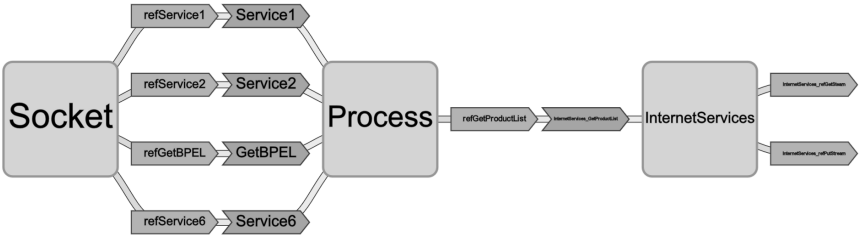


Figure 5.27: SCA Assembly

This example shows how the socket is linked to a process that provides four services and the process is linked to an application that provides one service.

BPEL When the component is a BPEL process, the pop-up will appear as shown in Figure 5.28.

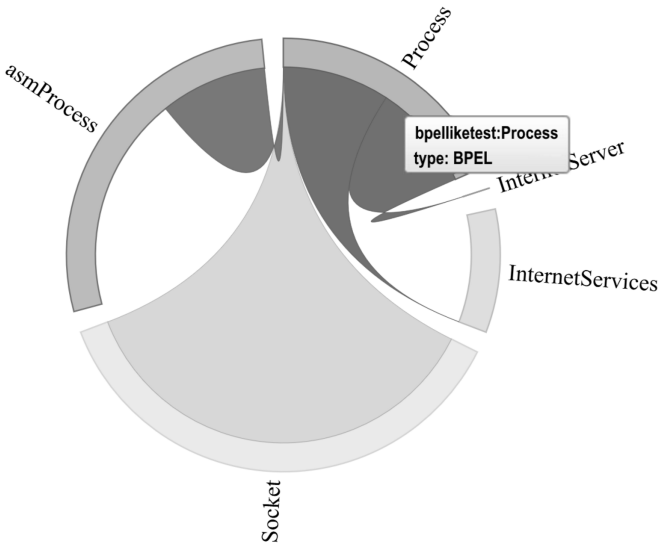


Figure 5.28: SCA BPEL Mouse-Over

Selecting the component will result in showing a diagram as in Figure 5.29.

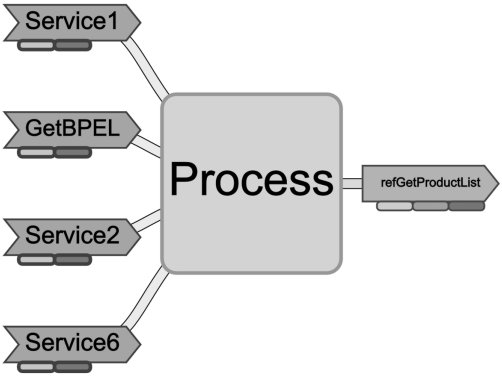


Figure 5.29: SCA BPEL

On clicking on the component in this diagram, there are two options for showing the process content. One diagram shows the logic flow as presented in Figure 5.30 and the other provides an outline of the process as shown in Figure 5.31.

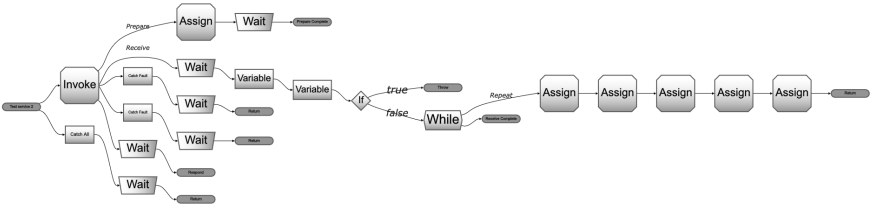


Figure 5.30: SCA Process Flow

The process flow diagram shows the logic flow of the activities within the process. A mouse-over of an activity brings up a pop-up listing the specific parameters of the activity.

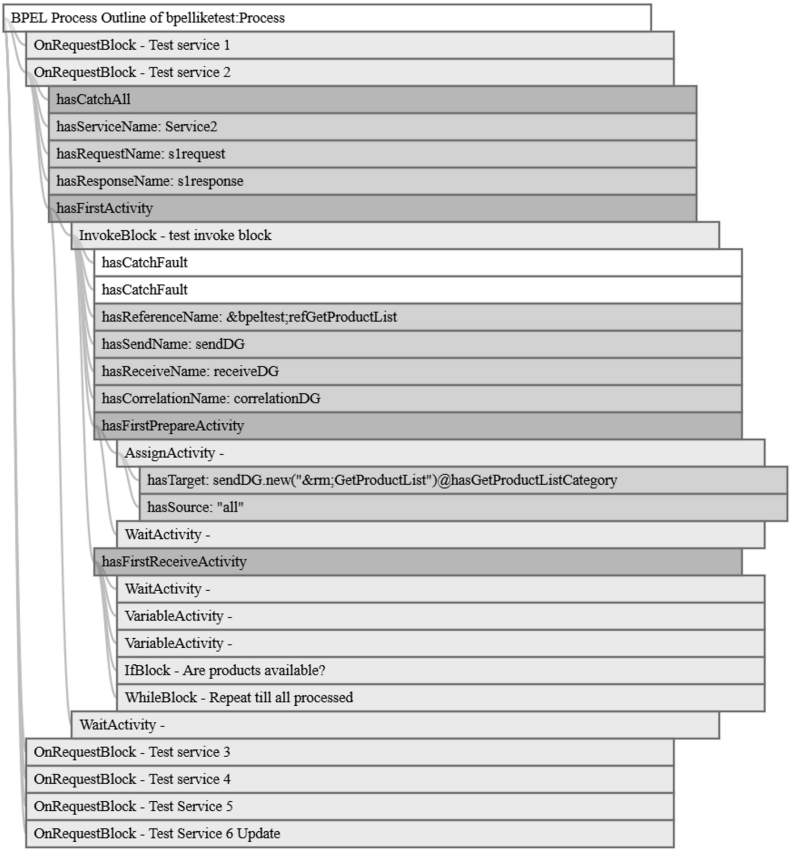


Figure 5.31: SCA Process Outline

The outline of the processes is produced using a D3 tree layout. Clicking on an element in the tree will result in the outline opening up the set of elements that make it up. In this example, only a few elements are shown open to reveal their content. The content may be selected that goes to the lowest most element.

5.4.4 Data Graphs

Data graphs, KBSgraphs, are the structure of the information that passes between a reference and a service. In SCA, this structure can be seen by clicking on the send, receive, or correlate buttons of a reference or by clicking on the request or response of a service.

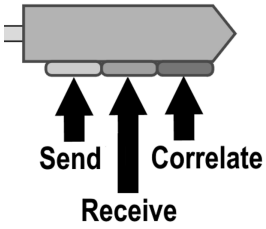


Figure 5.32: Component Reference

As shown in Figure 5.32 there are three potential data graphs that can be shown for a reference. The "send" is always required and is linked to the "request" of a service. The "receive" is not required for a send only reference, but when present it is linked to the "response" of a service. The "correlate" is not required and is only used to distinguish multiple references to the same service.

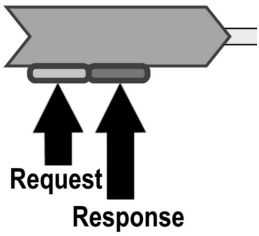


Figure 5.33: Component Service

Clicking on the "request" of a service, will display the structure of the data graph. The "response" is not required for request only

services, but when present it will display the structure of its data graph.

The structure of a data graph may be viewed from the perspective of the metadata or from the query used to define the data graph.

In Figure 5.34 is an example of the data graph following the format of a KBSgraph shown using a dendrogram.

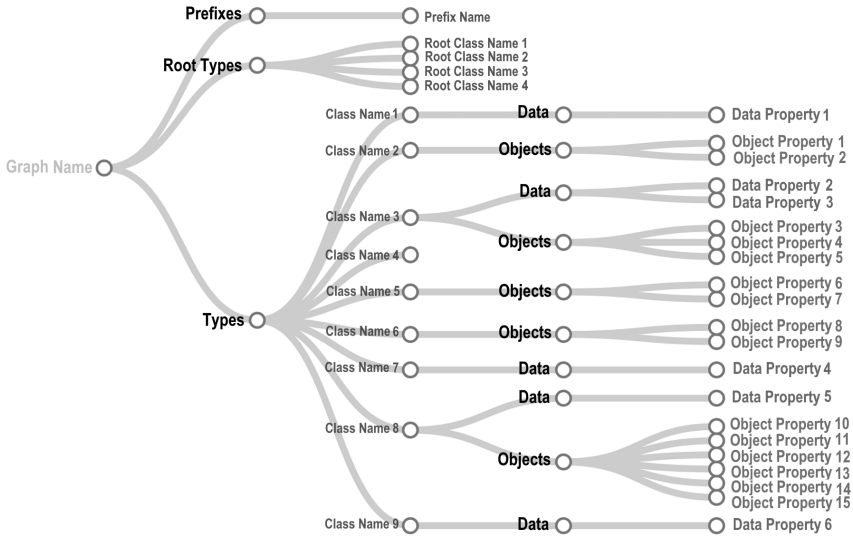


Figure 5.34: SCA Metadata

Metadata The following outline shows the structure of a selected KBSgraph:

- **Prefixes** - Defines the prefixes for names of URLs
- **Root Types** - Contains the list of class types in the KBSgraph root
- **Types** - Lists all of the class types in the KBSgraph

- **Data** - Lists the names of the data properties
- **Object** - Lists the names of the object properties

In this hypothetical example, there is one prefix name, one root type class name, and eight class types. Root types are defined under types. There are also six data properties and fifteen object properties shown. Class names 3 and 8 have both data and object properties, Class name 4 has no properties, and the others have either one or the other.

A mouse-over a data property will show a pop-up containing the name of the data type. A mouse-over an object property will show the class name of the range of the property.

Parsed Class Expression The OWL class expression provided to define the structure of the data graph may also be shown in a parsed format as shown in Figure 5.35.

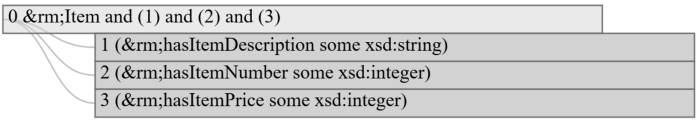


Figure 5.35: SCA Parsed Query

The expression in this format is shown using a tree structure by separating the expression into parts and sub-parts based on the use of parenthesis in the expression.

5.4.5 Composite Socket

The composite socket utilizes a zoom-able Sunburst to show all the content information as shown in Figure 5.36. The rings of the Sunburst from the center out are:

- **Ring 1** Services and References

- **Ring 2** Composite References
- **Ring 3** Domain References
- **Ring 4** Reference Send, Receive, and Correlation

Clicking on any ring will cause the Sunburst to zoom in on only the ring clicked and those outside it. Clicking the center of the Sunburst will bring the zoomed ring back out again.

References and Services In the initial view, all services and references are shown in Figure 5.36. In this example, there are no services defined for this composite socket.

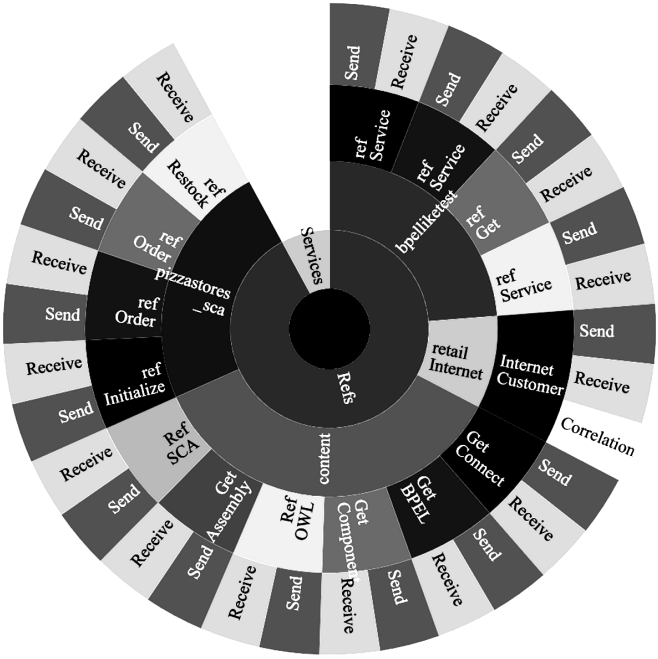


Figure 5.36: Composite Socket

Composite References Zooming in on services or references will then only show the one selected. In the example shown in Figure 5.37, references have been selected and the services section is no longer in the view.

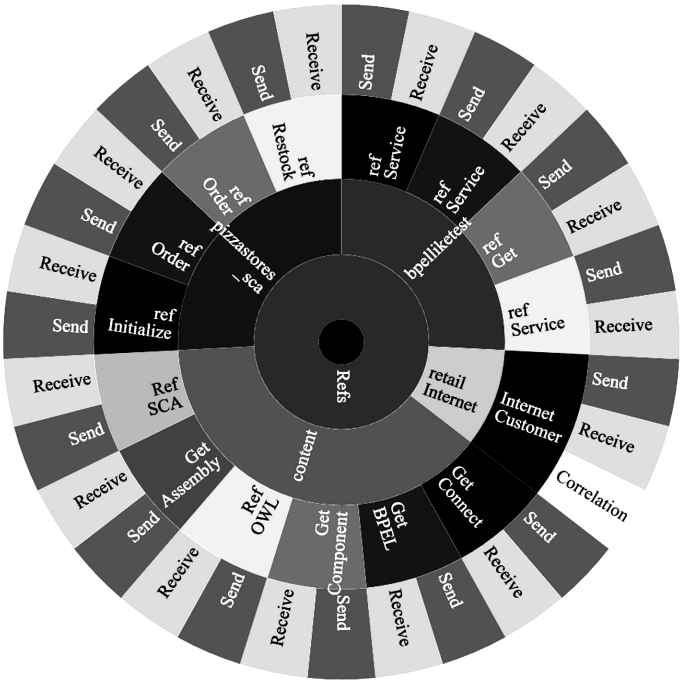


Figure 5.37: Composite Socket References

Domain References Zooming in on the "content" domain in references in this example produces the diagram shown in Figure 5.38. It contains only the references available within the selected domain.

Figure 5.38: Domain References

Reference Zooming in on a specific reference will show the data graphs defined for the reference as shown in Figure 5.39. In this example the reference selected is "Get Assembly" that has a send and a receive data graph.

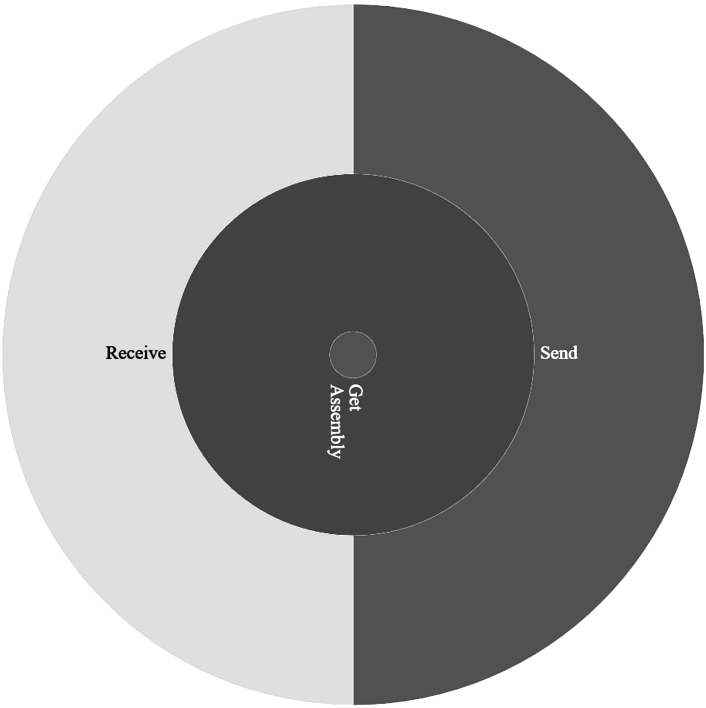


Figure 5.39: Domain Reference

Send, Receive, Correlation Clicking on the "receive" in this example brings up the data graph diagram as shown in Figure 5.40.

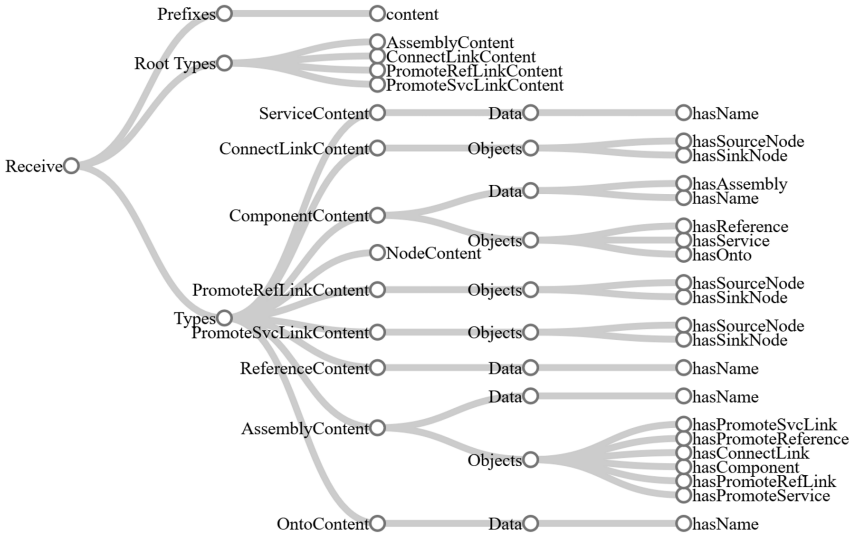


Figure 5.40: Reference Receive

Usage The composite socket visualization is useful in locating the exact requirements for requesting a service and interpreting the response. Also, when services are present, it will show the structure of the service request and the expected response.

5.5 3D Graphics

It seems that in order to provide a view of everything as real, it should be provided in three dimensions. This section shows one experimental view of SCA assembly components. The experiment utilizes the X3dom support for JavaScript using WebGL.

5.5.1 X3dom

X3dom is a JavaScript library of components providing an implementation of the X3D evolving standard. It is provided as open source. The following is an excerpt from the x3dom website describing the goal of the product.

> *The goal here is to have a live X3D scene in your HTML DOM, which allows you to manipulate the 3D content by only adding, removing, or changing DOM elements. No specific plugin or plugin interface (like the X3D-specific SAI) is needed. It also supports most of the HTML events (like onclick) on 3D objects. The whole integration model is still evolving and open for discussion.*

5.5.2 Assembly Components

Generally, assembly components are more complex and offer a greater potential to get value from a three dimensional view. The following are three included in the testing. Although the text can be difficult to read when it it is too far away from the viewer, the implementation provides for clicking on any component or flow to get a pop-up containing the name.

All the assembly components use a sphere to represent a component and rectangles to represent references and services. The Digraph module is used to calculate the x, y, and z coordinates needed for the view. Another component, Conduit, is used to compute the coordinates of the flow pipes connecting the components.

The X3dom library provides the means to view a scene from any angle and zoom in and out or rotate.

Content 3D View The diagram in Figure 5.41 is a view of the Content assembly. The scene is turned slightly so the components on the left appear closer than the components on the right.

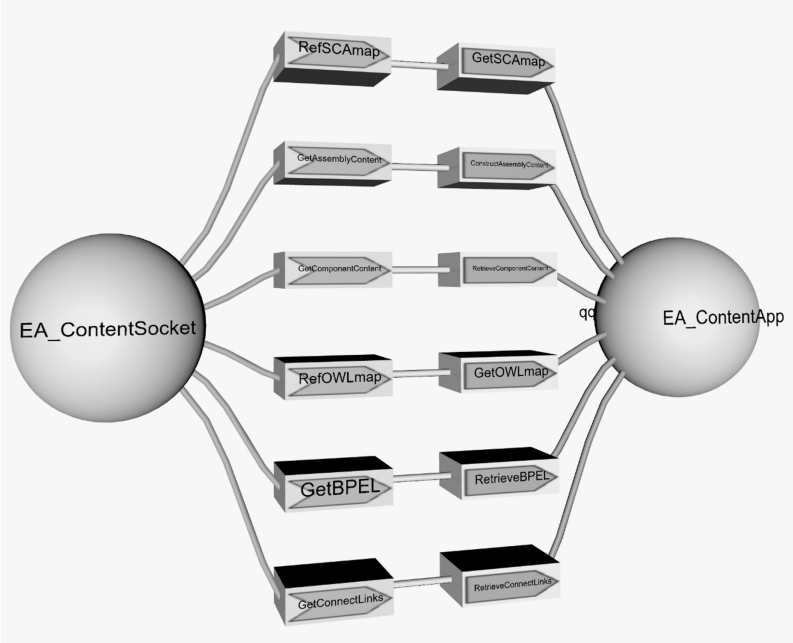


Figure 5.41: Content 3D View

Process 3D View In Figure 5.42 the scene is viewed more from the right and above for the process testing assembly.

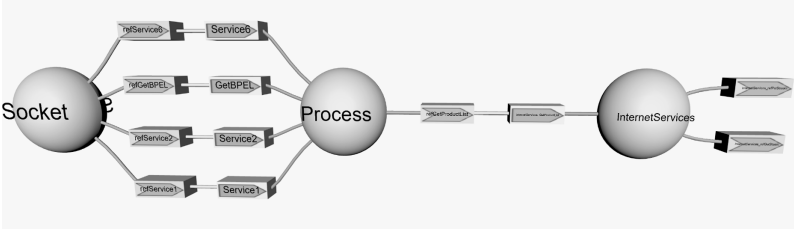


Figure 5.42: Process 3D View

Pizza Store 3D View In Figure 5.43 the scene is being viewed from the bottom so the components all appear to be laying on a flat surface.

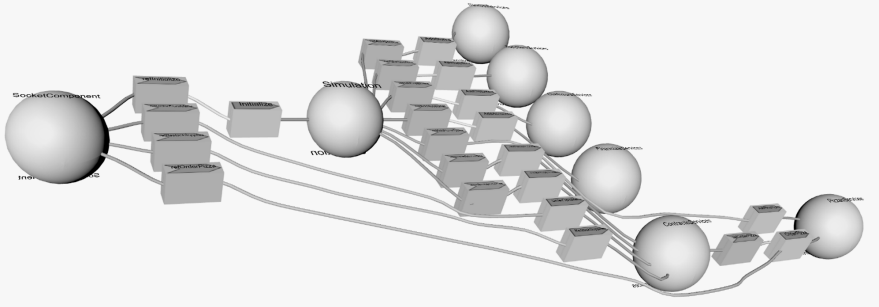


Figure 5.43: Pizza Store 3D View

Future of 3D The real future of 3D for KBS applications is not so much in the technology. It is in the creative application of 3D to allow for a better understanding of what is inside our accumulation of knowledge. This is particularly the case for metadata and process definitions.

5.6 Creative Opportunity

This chapter has shown some current visuals of the framework that are being applied to help see the inside of KBS applications. These visuals take a top-down approach by viewing the big picture and then drilling down into more detail.

The concept is good, yet there is enormous opportunity for creativity.

5.6.1 Previous Ideas

From an Enterprise Architect's view, the analogy of all the parts working together as a city seems very appropriate.

IT City Enterprise Architects Masters of the Unseen City is a book in which business domains are viewed as buildings where each floor of the building is a sub-domain. One of the graphics dynamically generated from the repository is shown in Figure 5.44.

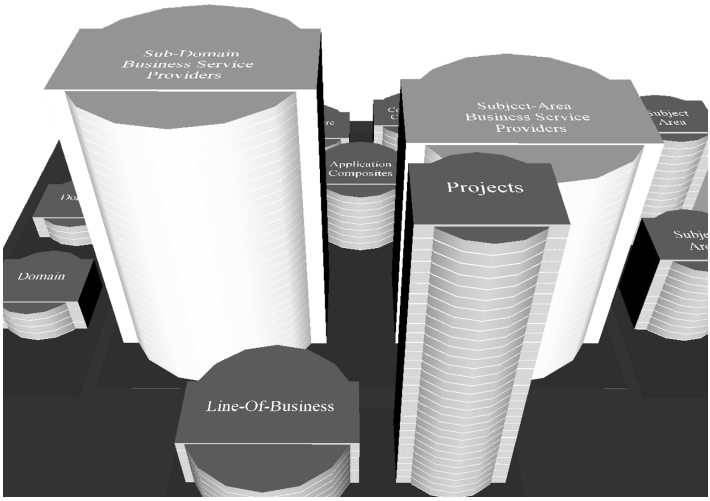


Figure 5.44: IT City

The city is produced in 3D with each building having the name of the domain on the top of the building. The lobby of each building could be entered and a directory of its floors could be used to go to the floor for more detail about a sub-domain.

IT City with Music An extension to the city analogy combined music with the simulated activity within the sub-domains. Activity was shown by appearing to turn the lights on for a floor of a building as shown in Figure 5.45. The volume of activity was mapped to the notes in the selected music.

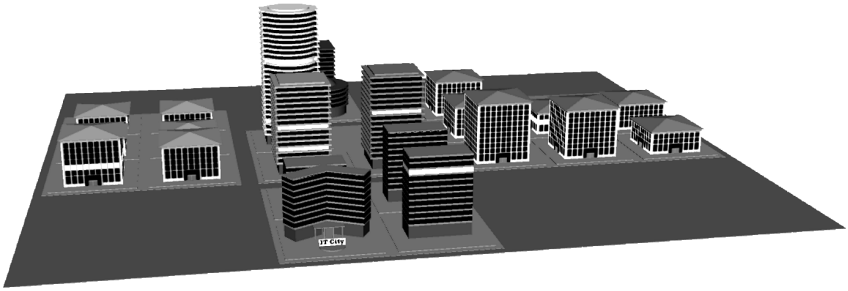


Figure 5.45: IT City with Music

The notes were sorted by the play time in the music and then divided into as many groups as sub-domains having activity. If there were more sub-domains than notes, the groups would have overlapping notes. In the end, every note was mapped to a floor in the city.

When the music played, the floor would appear to have its lights turned on while the note played. For fast music, the city would be very active. For slower music, the activity was easier to follow.

IT City With Plumbing In another city view, all the reference and service links were shown as underground plumbing. In Figure 5.46 is a view of the city from the side where the underneath of the building can be seen.

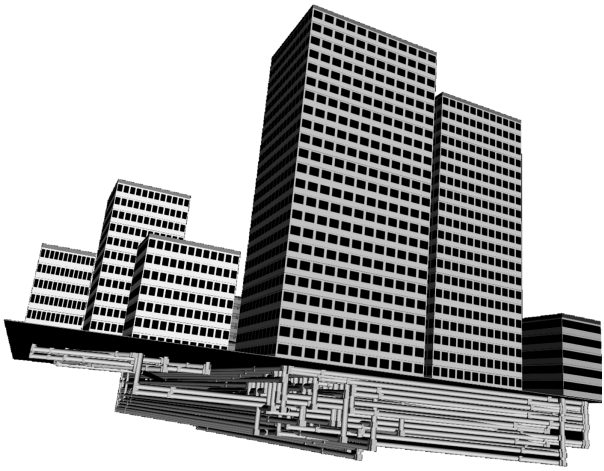


Figure 5.46: IT City with Plumbing

In Figure 5.47 the 3D view is turned to show more of the plumbing beneath the city.

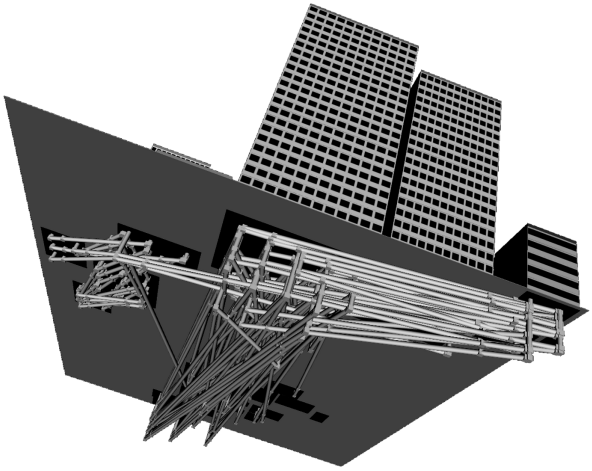


Figure 5.47: IT City from Underneath

In this view, any component could be selected to get more detail whether it was a building or a pipe.

5.6.2 Future Ideas

What should the view of a KBS application be? Should it follow the city approach and bring the plumbing up top in the form of flying cars moving back and forth as activity occurs? Should music play a role in the view? Should everything be viewed in virtual reality letting our avatar roam about?

Possibly, there will be many different views depending upon the range of domains included in a KBS application. It is also possible there may be many different analogies selectable by the viewer.

The nature of how to view a KBS application has more possibilities than restrictions. This calls for unlimited creativity.

Chapter 6

REA Pizza Stores Simulation

6.1 Putting It All Together

The pizza store simulation demonstrates a transaction-based operation using executable ontologies. The demo uses the Resource Event Agent (REA) academic disciple ontology to create a knowledge-based system for multiple pizza stores that use multiple providers of supplies.

The REA approach to tracking business operations is perfectly suited to pizza stores. Each store sells pizzas to customers in exchange for payment, following the concept of "exchange" described in REA. To complete orders for pizzas, a store must convert raw materials to produce the pizza, following the concept of "convert" described in REA. The store's purchase of raw ingredients from suppliers further applies the concept of "exchange".

The pattern applied in the simulation is presented in Pavel Hruby's book on model-driven design using REA.[1] The models - and even the code needed - to provide the exchange between a pizza store and a customer is included in the book. Many other examples of patterns are

[1] Pavel Hruby, *Model-Driven Design using Business Patterns*, Springer, 1998

also presented in the book, including patterns describing the "convert" pattern of REA.

The simulation does not attempt to cover all of the operations needed to run a pizza store. The purpose of the simulation is to demonstrate the functions of the framework, rather than to show how to run an actual food franchise.

Architecture The architecture of the simulation consists of the domains needed to run pizza stores, and the assembled process steps needed to make and sell pizzas, using ingredients purchased from suppliers.

Domains Combining the business domains needed to run pizza stores with the academic discipline of the REA pattern results in the architecture as shown in Figure 6.1.

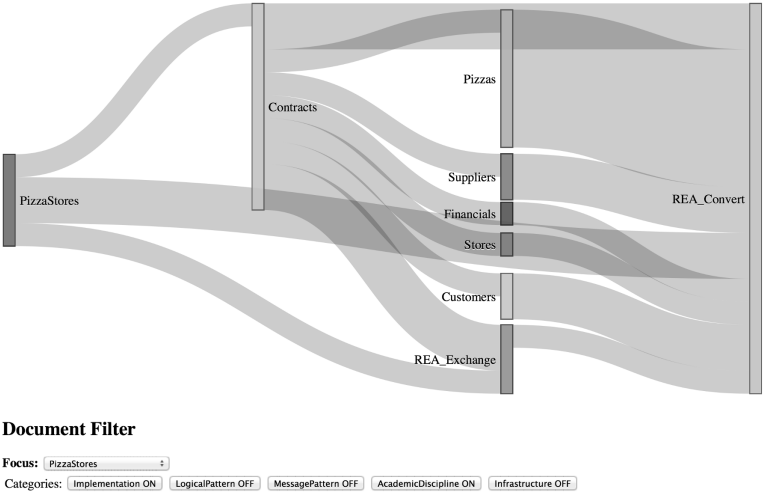


Figure 6.1: Pizza Domain Architecture

The chart in Figure 6.1 was dynamically produced by selecting the pizza stores domain as the focus, and by only including implementa-

tion ontologies and academic discipline ontologies. Although each of the domains contain an implementation, logical pattern, and message pattern ontology, only the implementation ontology is shown to reduce the visual complexity.

What is obvious by looking at this figure is that contracts play a major role in the operation of the simulation. It also shows the usage of the REA convert ontology in all of the other ontologies. This is due to the REA convert ontology containing the basic classes defining agents and resources.

Assembly The assembly component diagrammed in Figure 6.2 shows all of the SCA components used in the simulation.

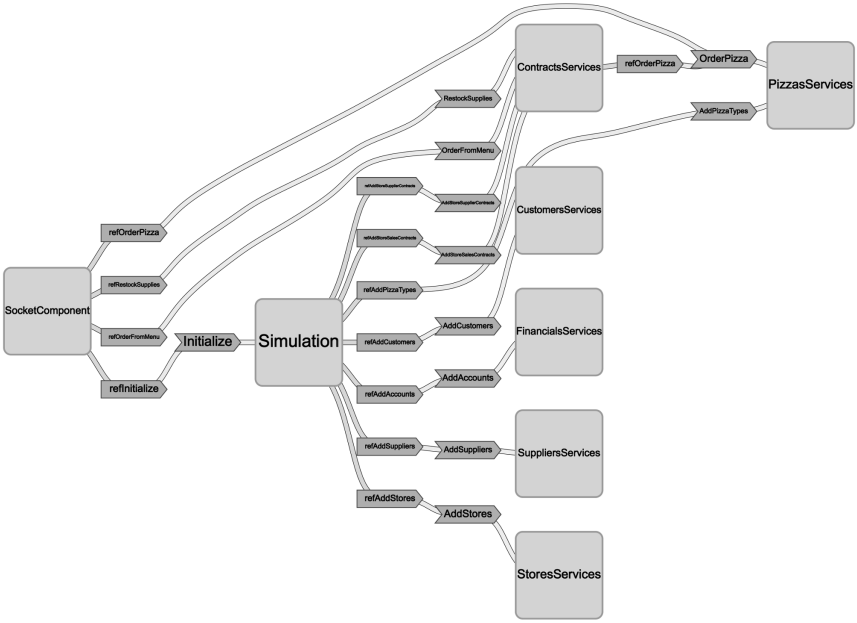


Figure 6.2: Pizza Component Assembly

All message activity is initiated from the “Socket Component”.

From a web browser, the test data loading can be initiated, pizzas may be ordered, and ingredients may be refilled from suppliers.

6.1.1 REA Model for Selling Pizzas

There are two REA models applied in the simulation: the exchange pattern and the convert pattern. The exchange pattern is used in two ways in the simulation. Firstly, for buying ingredients needed by a pizza store to make pizzas, and secondly, to allow customers to purchase pizzas. The convert pattern is used to make the specific pizza ordered by a customer from the ingredients that were purchased.

The REA model for exchange is based upon contracts between a provider agent and a receiver agent. This is applied in the simulation with contracts between each store and each supplier, and with contracts between each store and its customers. The contracts define the exchange rates.

Selling Pizzas - Exchange Selling pizza involves an exchange of money for goods between a customer and a pizza store. The store is the agent that provides the resource of pizza to the customer agent. The money received from the customer for the pizza then goes into the account resource for tracking sales. The pizza resource is decremented and the sales account is incremented.

The exchange rate in the simulation is the price for each type of pizza. The menu lists the pizzas and their exchange rates, and this information is used by a customer to place an order for pizzas.

Providing Supplies - Exchange To sell pizzas, a store must maintain the needed ingredients to make pizzas. The supplier agent provides the ingredient resources. The store agent then pays for the ingredients from an expense account resource. The ingredient resources are incremented and the expense account resource is decremented.

The exchange rates set the cost of purchasing a specific quantity of an ingredient. In the simulation, only multiples of the quantity

defined may be purchased from a supplier. In other words, the store must purchase a whole can of pizza sauce rather than part of a can.

Making Pizzas - Convert In the convert, the ingredients are combined to make a pizza. The ingredients are the resources converted to create the resource of pizzas. The ingredients are decremented and the pizzas are incremented.

For each type of pizza there is a recipe that defines the specific quantity of each ingredient that goes into one pizza. The simulation does not offer the customer the option to change the recipe by adding or removing toppings.

Variable Cost Only The variable costs of the ingredients is all the simulation includes. The fixed costs of real estate for each store, the equipment required, and personnel needed to run the store are not included in the simulation. If they were, however, they could all be based upon the REA models.

Running the Simulation Once the store simulation page is loaded, there are only two steps needed to set it in motion:

- **Load Test Data** - All the data representing all of the individuals defined in the ontologies may be loaded by pressing the "Load Data" button.

  Loading of the test data may take several minutes. However, it is only necessary to load the data once.

- **Run Simulation** - Pressing this button starts the process of randomly generating orders of pizzas for the stores. Each time the simulation is run, the results will be different due to the random selection of stores and pizza types.

  Starting the simulation will result in displays that are updated as the sale of pizzas continues and ingredients require restocking.

Each cycle of the simulation is for one day of making and selling pizzas. The base simulation is set to five days, but may be adjusted for longer or shorter periods.

Visuals The simulation produces multiple visuals during the process. These include showing which component links are active, the revenue and cost by store, and the transaction activity.

Heartbeat The heartbeat is a 3D view of all the components and links in the framework. This view is updated approximately every second when the location of the components are recalibrated. The recalibration causes all the components and links in the diagram to briefly compress and expand. Consequently, it appears like a heartbeat.

An example of the heartbeat is shown in Figure 6.3.

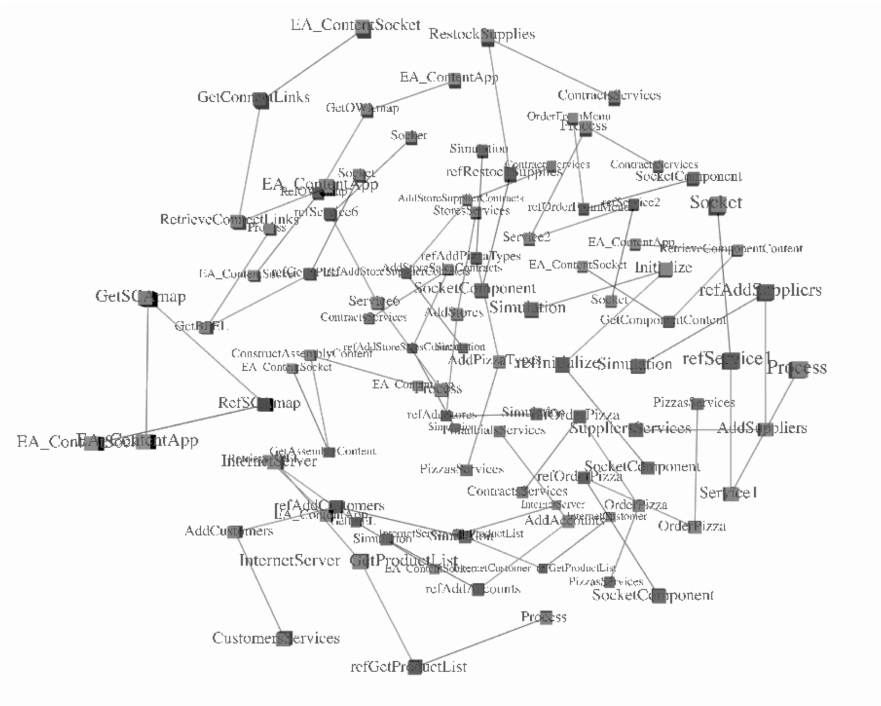


Figure 6.3: 3D Heartbeat

The 3D view may be rotated and zoomed in and out to examine specific areas of activity.

Another example of the heartbeat is shown in Figure 6.4.

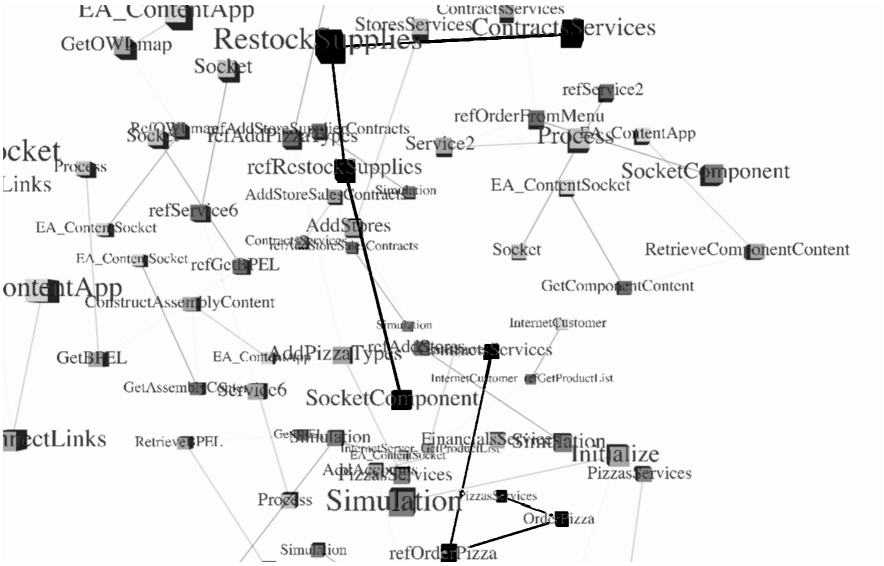


Figure 6.4: 3D Heartbeat Active

In this example, components and their links that are active are highlighted. The highlight flashes on and fades quickly.

Store Revenue and Expense As pizza sales occur over the time of the simulation, the revenue and expenses are incrementally shown, as in the example in Figure 6.5. Both the revenue and expenses show the totals for a store over the entire period of the simulation.

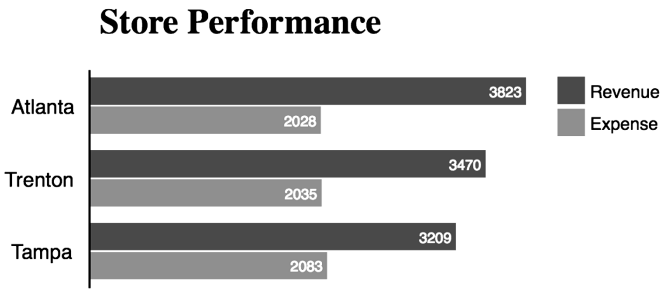


Figure 6.5: Store Revenue and Expense

Moving the mouse pointer over a revenue bar will show the contribution for each type of pizza sold, as shown in Figure 6.6.

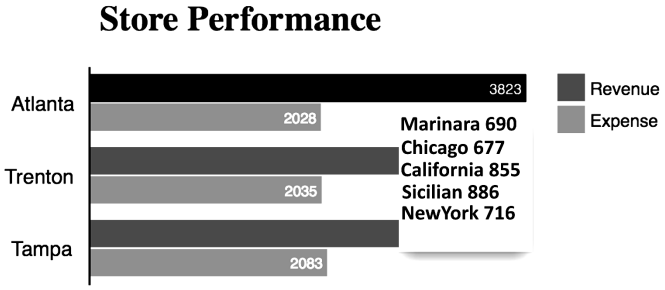


Figure 6.6: Store Revenue and Expense / Revenue Content

Moving the mouse pointer over an expense bar will show the total cost of the ingredients used in preparing pizzas, as shown in Figure 6.7.

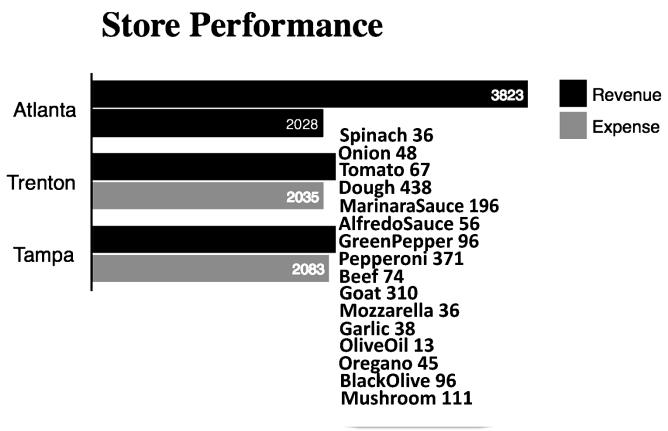


Figure 6.7: Store Revenue and Expense / Expense Content

The correlation between revenue and expense is circumstantial. As ingredients are consumed in their conversion to pizzas, the store's stock is reduced. The expense bar represents the cost of restocking ingredients that have fallen below their minimum requirement. Over the time of the simulation, the revenue and expense will have a relative correlation.

Supplier by Store Suppliers are monitored by store as shown in Figure 6.8. As the simulation progresses, the bar showing the cost to each store changes incrementally.

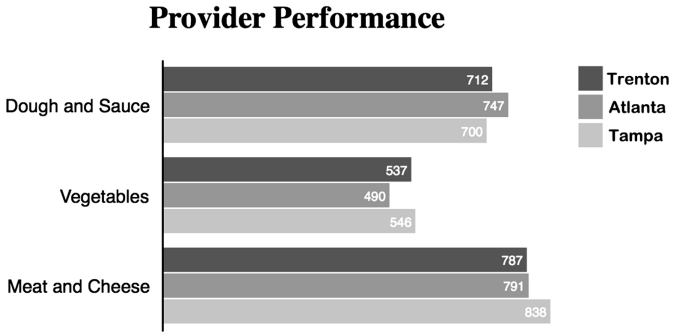


Figure 6.8: Provider Expense by Store

Moving the mouse pointer over a store bar will show the total of each ingredient restocked, as shown in Figure 6.9.

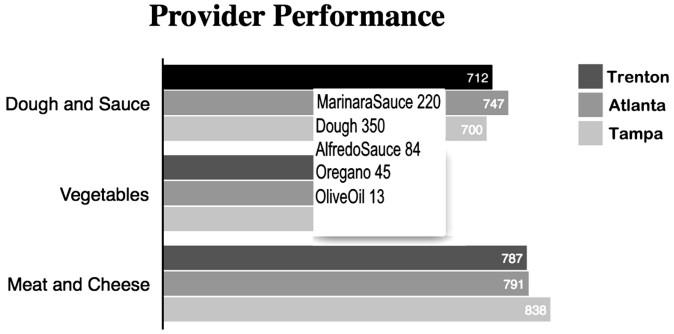


Figure 6.9: Provider Expense by Store / Content

Order from Menu During each day of the simulation, pizzas are ordered from the menu and made for customers. Figure 6.10 shows an example of an order sent through the socket. Figure 6.11 shows the response to the orders sent.

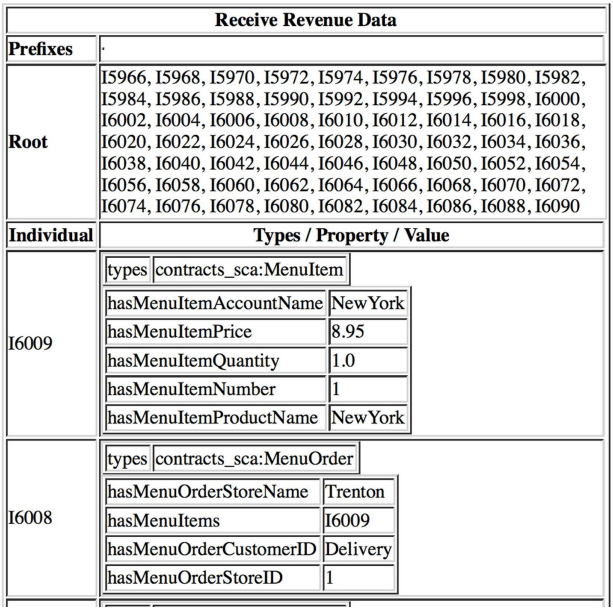
Pizzas are ordered randomly from the menu by type, store, and

customer. In Figure 6.10, only two of the orders are shown, although the request includes sixty nine unique orders (order0 - order68).

Send Order From Menu Data		
Prefixes	bpelliketest	http://www.SelfServiceIT.com/otter/lib/ea/2012/7/BPELLikeTest.owl#
	retailInternetModel	http://www.SelfServiceIT.com/otter/lib/2012/4/ea/RetailInternetModel.owl#
	content	http://www.SelfServiceIT.com/otter/lib/2012/6/ea/$EA_SCAliteContent.owl#
	pizzastores_sca	http://otter.eatoolkit.com/ontologies/2014/11/PizzaStores_SCA.owl#
Root	order0, order1, order2, order3, order4, order5, order6, order7, order8, order9, order10, order11, order12, order13, order14, order15, order16, order17, order18, order19, order20, order21, order22, order23, order24, order25, order26, order27, order28, order29, order30, order31, order32, order33, order34, order35, order36, order37, order38, order39, order40, order41, order42, order43, order44, order45, order46, order47, order48, order49, order50, order51, order52, order53, order54, order55, order56, order57, order58, order59, order60, order61, order62, order63, order64, order65, order66, order67, order68	
Individual	**Types / Property / Value**	
order0	types	contracts_sca:MenuOrder
	hasMenuOrderCustomerID	Delivery
	hasMenuOrderStoreID	3
	hasMenuOrderProductName	California
order1	types	contracts_sca:MenuOrder
	hasMenuOrderCustomerID	Delivery
	hasMenuOrderStoreID	3
	hasMenuOrderProductName	Sicilian

Figure 6.10: Order From Menu

The revenue received, as shown in Figure 6.11, includes the type of pizza, the store, and the price at which the pizza was sold. In this example, only two of the sales are shown of the sixty-three sales that occurred in the simulation.



Receive Revenue Data	
Prefixes	·
Root	I5966, I5968, I5970, I5972, I5974, I5976, I5978, I5980, I5982, I5984, I5986, I5988, I5990, I5992, I5994, I5996, I5998, I6000, I6002, I6004, I6006, I6008, I6010, I6012, I6014, I6016, I6018, I6020, I6022, I6024, I6026, I6028, I6030, I6032, I6034, I6036, I6038, I6040, I6042, I6044, I6046, I6048, I6050, I6052, I6054, I6056, I6058, I6060, I6062, I6064, I6066, I6068, I6070, I6072, I6074, I6076, I6078, I6080, I6082, I6084, I6086, I6088, I6090
Individual	**Types / Property / Value**
I6009	types: contracts_sca:MenuItem hasMenuItemAccountName: NewYork hasMenuItemPrice: 8.95 hasMenuItemQuantity: 1.0 hasMenuItemNumber: 1 hasMenuItemProductName: NewYork
I6008	types: contracts_sca:MenuOrder hasMenuOrderStoreName: Trenton hasMenuItems: I6009 hasMenuOrderCustomerID: Delivery hasMenuOrderStoreID: 1

Figure 6.11: Revenue

Restock Stores The restock of store ingredients is a single request for all stores. Each store examines its ingredients and orders those that have fallen below a specified quantity. As shown in Figure 6.12, a restock request has no message content. This is what is shown as "Nothing" in the root individuals of the request KBSgraph.

Send Restock Request Data		
Prefixes	bpelliketest	http://www.SelfServiceIT.com /otter/lib/ea/2012/7 /BPELlikeTest.owl#
	retailInternetModel	http://www.SelfServiceIT.com /otter/lib/2012 /4/ea/RetailInternetModel.owl#
	content	http://www.SelfServiceIT.com /otter/lib/2012 /6/ea/$EA_SCAliteContent.owl#
	pizzastores_sca	http://otter.eatoolkit.com /ontologies/2014/11 /PizzaStores_SCA.owl#
Root	Nothing	
Individual	**Types / Property / Value**	

Figure 6.12: Restock Request

An example response to a restock request is shown in Figure 6.13.

Receive Supply Orders Data	
Prefixes	·
Root	I6092, I6093, I6094, I6095, I6096
Individual	**Types / Property / Value**
I6096	types: contracts_sca:Order hasOrderPrice: 27.0 hasOrderStoreID: 1 hasOrderAccountID: 02 hasOrderQuantity: 296.0 hasOrderIngredient: Mushroom hasOrderStoreName: Trenton hasOrderAccountName: Vegetables
I6095	types: contracts_sca:Order hasOrderPrice: 25.3 hasOrderStoreID: 1 hasOrderAccountID: 02 hasOrderQuantity: 140.0 hasOrderIngredient: BlackOlive hasOrderStoreName: Trenton hasOrderAccountName: Vegetables

Figure 6.13: Ingredient Expense

The response includes the store, the provider, the ingredient, and the cost of restocking.

6.2 REA Ontologies

The REA structure is provided in two separate ontologies developed within the OTTER project:

- **Exchange** - Provides the structure for contracts.
- **Convert** - Provides the structure for producing.

Basically, REA exchange is the methodology for an organization to follow when working with another organization. REA convert supports the process of an organization producing a product or service by using the organization's internal resources. In other words, exchange is an external process and convert is an internal process.

6.2.1 Exchange - Structure for Contracts

The REA exchange ontology is summarized in Figure 6.14. It shows that a contract is between a provider agent and a receiver agent. The contract also may have multiple commitments, and each commitment may have multiple exchange rates.

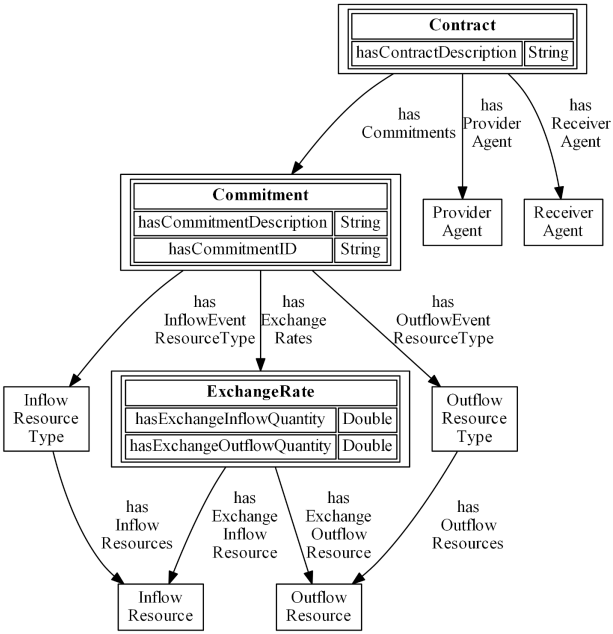


Figure 6.14: REA Exchange

The exchange rates define the quantity of each exchange of resources. The inflow resource is incremented by the inflow quantity, and the outflow is decremented by the outflow quantity.

6.2.2 Convert - Structure for Producing

The REA convert ontology summarized in Figure 6.15 shows the convert as having resource constituents and an agent that performs the conversion.

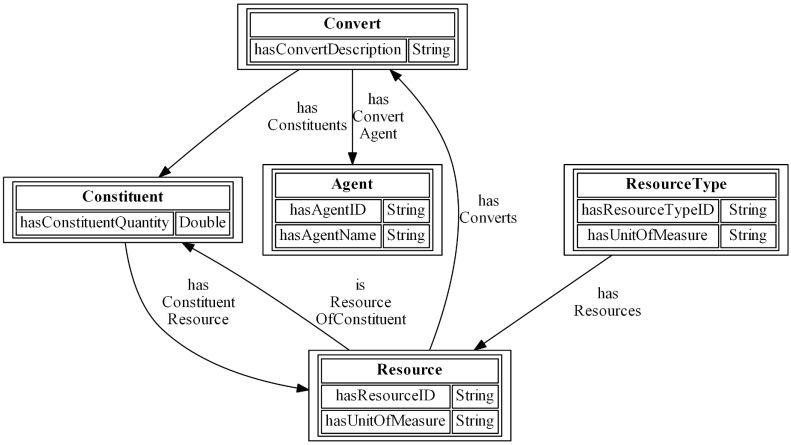


Figure 6.15: REA Convert

The constituents are the things that go into the production of a product or service. This may include the raw materials, equipment used, and human resource time.

6.3 Pizza Stores Business Domains

The pizza store implementation includes six business domains. Each of the domains includes an implementation ontology, a logical ontology, a message ontology, and a component that provides services.

The Contracts business domain utilizes the REA exchange ontology as its logical ontology. This is because there are no extensions needed to define contracts for the pizza stores. The other domains utilize the REA convert ontology and do provide extensions for agents and resources.

6.3.1 Domain Ontologies

The domains included in the pizza store ontology are shown in their parts in Figure 6.16. These parts include the implementation, the

logical pattern, and the message pattern of each of the domains.

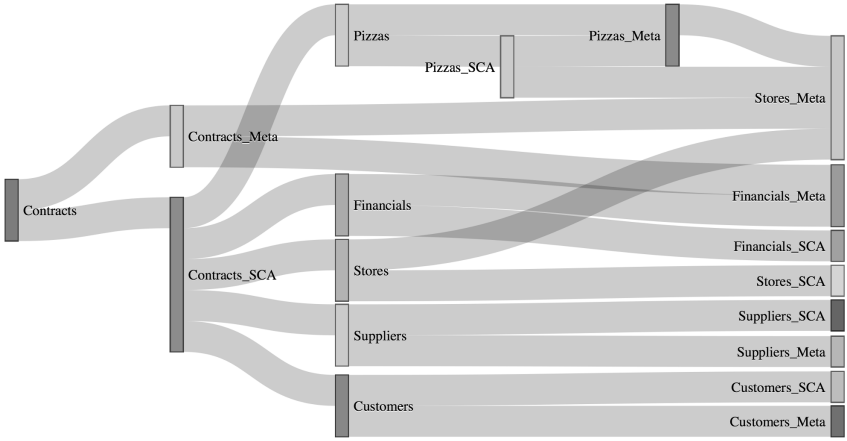


Figure 6.16: Domain Ontologies

All of the REA exchange processes are controlled by the contracts defined in the Contract's message ontology, "Contracts_SCA".

6.3.2 Components

There are nine components shown in Figure 6.17, along with their links to one another.

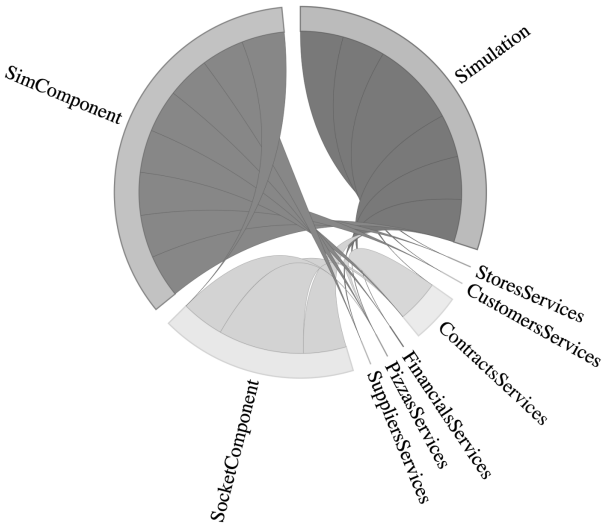


Figure 6.17: Component Links

The "SimComponent" is the assembly of the other eight components. This assembly is shown in Figure 6.2. The assembly begins with the socket component, which gives access to the services of the other components.

Socket Component The socket component is the gateway to the simulation. As shown in Figure 6.18, access is given to place orders, restock ingredients, and load the test data.

Figure 6.18: Pizza Stores Socket

The SCA model of the socket component as shown in Figure 6.19 has four references:

- **refInitialize** - This reference is linked to the simulation services to load the test data.
- **refOrderFromMenu** - Linked to the contract service that orders pizza from a menu.
- **refOrderPizza** - This is a testing link to verify that the convert process for making pizzas is working correctly.
- **refRestockSupplies** - Linked to the contract services that identify which ingredient supplies have fallen below minimum stock requirements, and then uses supplier contracts to restock them.

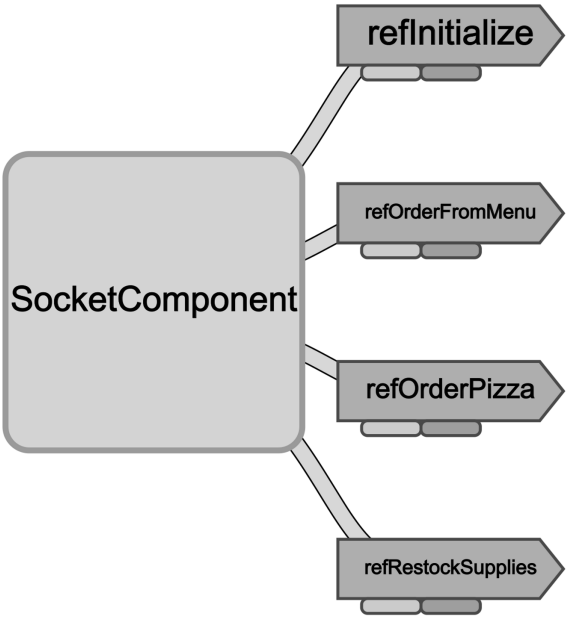


Figure 6.19: Socket Component

Simulation The simulation component is primarily used to load the test data using references, as shown in Figure 6.20.

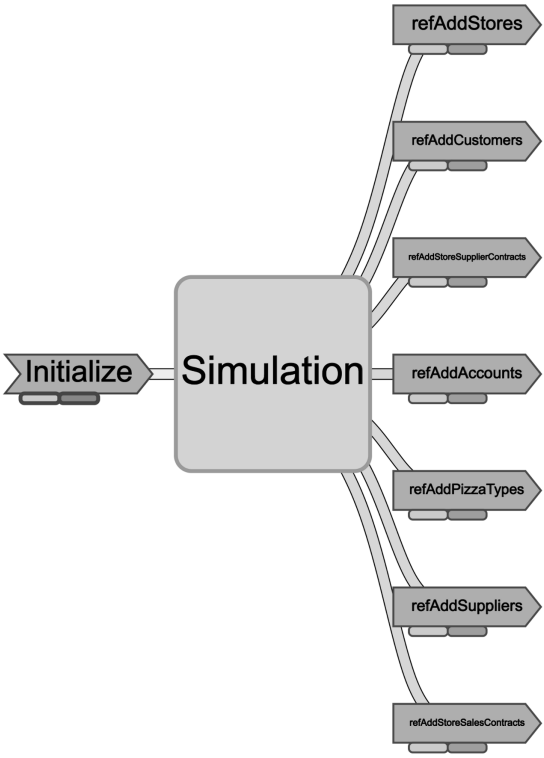


Figure 6.20: Simulation

The simulation component has only one service requested to load the test data. It then calls the appropriate business domain services to load the actual test data using the following references:

- **refAddStores** - This reference is linked to the stores service to add stores and locations in the stores for stock ingredients.

- **refAddCustomers** - Linked to the customer service to add customer types.
- **refAddAccounts** - This links to the financial services to add accounts.
- **refAddStoreSupplierContracts** - Linked to the contract service to add contracts between the stores and the suppliers.
- **refAddPizzaTypes** - This reference is linked to the pizza service to add a pizza type along with the associated recipes.
- **refAddSuppliers** - This reference is linked to the supplier service to add suppliers.
- **refAddStoreSalesContracts** - This reference is linked to the contract service to define the price for selling each pizza type.

Contract Services The contract services component is shown in Figure 6.21 as having four services.

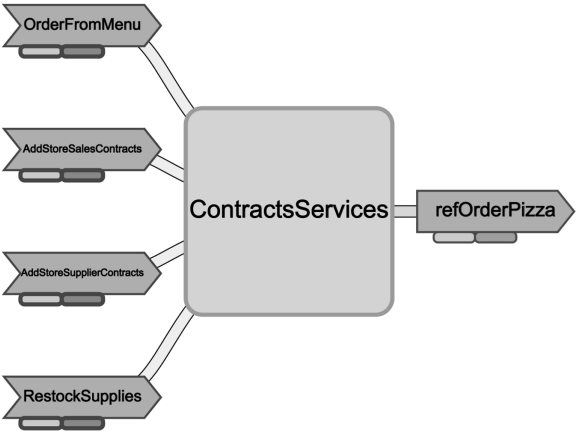


Figure 6.21: Contract Services

- **OrderFromMenu** - This service simulates the ordering of pizza types from stores.
- **AddStoreSalesContracts** - Adds sales contracts that are essentially the menu for ordering pizzas.
- **AddStoreSupplierContracts** - Adds store/supplier contracts for a store to purchase pizza ingredients.
- **RestockSupplies** - This service checks all store ingredients for those falling below minimum stock requirements and orders from the suppliers according to the store/supplier contract.

Customer Services As shown in Figure 6.22, the only service provided by customer services is to add a type of customer.

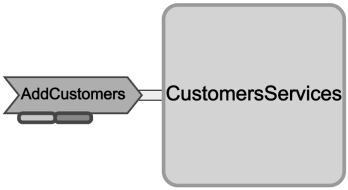


Figure 6.22: Customer Services

Financial Services Figure 6.23 shows that financial services supports the addition of new accounts.

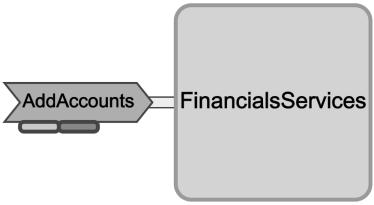


Figure 6.23: Financial Services

Supplier Services Supplier services has one service to add new suppliers, as shown in Figure 6.24

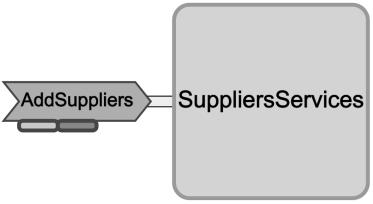


Figure 6.24: Supplier Services

Store Services Store services, as shown in Figure 6.25, shows the service to add a new store. When a new store is added, this service also defines the locations for all the ingredients to be used by the store in making pizzas.

Figure 6.25: Store Services

Pizzas Services The pizzas services component has two services, as shown in Figure 6.26.

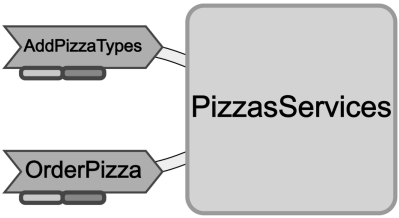


Figure 6.26: Pizza Services

- **AddPizzaTypes** - This service adds pizza types, along with their recipe of ingredients.
- **OrderPizza** - Using the REA convert process, a pizza is made from the ingredients in a recipe.

6.4 Individual Content

The content of the simulation covers all that is needed to perform the selling of pizzas and the purchase and conversion of ingredients.

Customer Two types of customers are included in the simulation:

- **Pickup** - Customers who pick up their orders.
- **Delivery** - Customers whose orders are delivered by the store.

Having two types of customers allows for different pricing. Only the "Delivery" type is used in the simulation.

Stores The simulation includes three stores named:

- **Trenton** - Store 01
- **Atlanta** - Store 02
- **Tampa** - Store 03

Each store offers all pizza types and maintains the complete selection of supplies needed to prepare the pizzas.

Supply Location In each store, the simulation includes a location for supplies for each pizza ingredient. To manage the supplies kept in each location, the following data is maintained:

- **Ingredient** - A link to the ingredient kept at this location.
- **Quantity** - Quantity on hand at this location.
- **Quantity Max** -The maximum quantity that can be stored at this location.
- **Quantity Min** - When the quantity at this location falls below this minimum, the ingredients need to be refilled.
- **Quantity Less Min** - This quantity is maintained as (Quantity minus Quantity Min).

The "Quantity Less Min" is used to query for any locations that have a zero or less than zero value. The OWL class expression supports this type of selection.

Ingredients Ingredients are identified in four groups: Basic, Cheese, Meat, and Vegetable, as follows:

Type	Ingredient	Unit	Min	Max
Basic	Marinara Sauce	Ounce	32	320
Basic	Dough	Ball	18	180
Basic	Alfredo Sauce	Ounce	32	320
Basic	Oregano	Ounce	4	40
Basic	Olive Oil	Ounce	4	40
Basic	Garlic	Ounce	4	40
Cheese	Mozzarella	Ounce	16	160
Cheese	Goat	Ounce	32	320
Meat	Beef	Ounce	16	160
Meat	Pepperoni	Ounce	9	90
Vegetable	Green Pepper	Ounce	16	160
Vegetable	Mushroom	Ounce	16	160
Vegetable	Black Olive	Ounce	6	60
Vegetable	Spinach	Ounce	16	160
Vegetable	Tomato	Ounce	16	160
Vegetable	Onion	Ounce	3	30

The minimum and maximum values for each ingredient are set the same for all stores.

Suppliers The simulation includes three suppliers named:

- **Meat and Cheese** - Supplier 01
- **Vegetables** - Supplier 02
- **Dough and Sauce** - Supplier 03

Each supplier offers specific supplies to each store.

Pizzas There are four types of pizzas prepared in the simulation. The following lists the pizzas and their contents. The measure is based upon the unit of measure of the ingredient.

Italian - Marinara

Ingredient	Measure
Dough	1.000
Marinara Sauce	8.000
Tomato	6.500
Oregano	0.035
Garlic	0.035
Olive Oil	0.035
Pepperoni	3.390
Beef	6.500
Mushroom	6.500
Onion	2.500
Black Olive	4.000
Green Pepper	4.000

Italian - Sicilian

Ingredient	Measure
Dough	1.000
Marinara Sauce	8.000
Mozzarella	0.7400
Pepperoni	3.390
Beef	6.500
Mushroom	6.500
Onion	2.500
Black Olive	4.000
Green Pepper	4.000

American - Chicago

Ingredient	Measure
Dough	1.000
Marinara Sauce	8.00
Mozzarella	0.740
Beef	6.500
Tomato	6.500
Mushroom	6.500
Onion	2.500
Black Olive	4.000
Green Pepper	4.000

American - New York

Ingredient	Measure
Dough	1.000
Marinara Sauce	8.000
Mozzarella	0.740
Beef	6.500
Mushroom	6.500

American - California

Ingredient	Measure
Dough	1.000
Alfredo Sauce	8.000
Spinach	2.500
Goat	0.74

Financial Accounts Financial accounts are set up for each store to track expenses and revenue. The balance of an expense account is deducted as payments are made to suppliers and the revenue accounts are incremented on the sale of pizzas.

Expense Accounts Expense accounts for supplies are set up for each store. Each of these accounts is summarized into a single account for each store under "Ingredient Supplies" and initialized with a

starting balance. As the simulation progresses, the balance is reduced as orders for new supplies are posted.

ID	Name	Summary	Balance
Account01	Basic	Ingredient Supplies	1000.00
Account02	Cheese	Ingredient Supplies	1000.00
Account03	Meat	Ingredient Supplies	1000.00
Account04	Vegetable	Ingredient Supplies	1000.00

Revenue Accounts Revenue is recorded for each pizza sale by store. The accounts are named after the pizza types, the summary account for each store is "Pizza Sales", and the initial balance is set to zero.

ID	Name	Summary	Balance
Sales01	Marinara	Pizza Sales	0.00
Sales02	Sicilian	Pizza Sales	0.00
Sales03	Chicago	Pizza Sales	0.00
Sales04	New York	Pizza Sales	0.00
Sales05	California	Pizza Sales	0.00

Contracts Contracts exist between a provider and a consumer. Within each contract there are commitments that define the exchange rates. The stores have a contract with customers to deliver pizzas at a given price. Suppliers have a contract to fill supply orders for the stores for the units of ingredients at a contracted price.

These contract exchanges are presented in the following.

Store - Supplier Contracts The supplier contracts establish the commitments between a store and a supplier. In the contract, each store is the receiver and each supplier is the provider. Since there are three stores and three suppliers, there are nine supplier contracts. In the following, the contracts are summarized by a supplier showing commitment and the exchanges for each ingredient.

Following the REA pattern, a commitment is for a category of ingredients, and it includes exchange rates for multiple ingredients.

The "Inflow" is the account to be increased by the exchange and the "Outflow" is the account to be decreased. The "Units" are the measure of the inflow, and the "Cost" is the expense of the outflow.

The store can only purchase supplies in multiples of the units specified. There may be more than one exchange rate for an ingredient such that purchasing larger units may potentially result in a lower prices.

Supplier: Meat and Cheese				
Commitment	**Inflow**	**Units**	**Outflow**	**Cost**
Supply Cheese	Mozzarella	16	Cheese	3.00
Supply Cheese	Goat	32	Cheese	13.00
Supply Meat	Pepperoni	9	Meat	9.50
Supply Meat	Beef	32	Meat	2.00

Supplier: Vegetables				
Commitment	**Inflow**	**Units**	**Outflow**	**Cost**
Supply Vegetables	Tomato	32	Vegetable	2.00
Supply Vegetables	Garlic	16	Vegetable	18.75
Supply Vegetables	Mushroom	16	Vegetable	1.50
Supply Vegetables	Onion	3	Vegetable	0.45
Supply Vegetables	Black Olive	16	Vegetable	1.10
Supply Vegetables	Green Pepper	16	Vegetable	2.45
Supply Vegetables	Spinach	16	Vegetable	3.00

Supplier: Dough and Sauce				
Commitment	**Inflow**	**Units**	**Outflow**	**Cost**
Supply Basics	Dough	18	Basic	17.50
Supply Basics	Marinara Sauce	32	Basic	4.00
Supply Basics	Alfredo Sauce	32	Basic	4.00
Supply Basics	Oregano	4	Basic	4.50
Supply Basics	Olive Oil	4	Basic	1.28
Supply Basics	Garlic	4	Basic	9.37

Store - Customer Contract Each store sells pizzas at the same price. The "Outflow" is one pizza made by applying the REA convert pattern, and the "Inflow" of the same name is the financial account for recording the transaction.

Contract: Pizza Sales Contract for Store				
Commitment	**Inflow**	**Price**	**Outflow**	**Units**
Provide Pizzas	Marinara	10.95	Marinara	1
Provide Pizzas	Sicilian	9.95	Sicilian	1
Provide Pizzas	Chicago	9.95	Chicago	1
Provide Pizzas	New York	8.95	New York	1
Provide Pizzas	California	12.95	California	1

6.5 Lessons Learned

As with any project, there are some things that work well and some things that do not. This section describes some of the most important lessons learned about the concepts and the technologies applied in the prototype.

6.5.1 Key Concepts

The key concepts of the framework have been proven to work and to work well. The following describes the most important of these concepts.

Graphs The structure of graphs fits well into the real world view of information processing. Although structures such as tables, flat files, and XML are useful, they are only unique forms of graphs. The pizza store simulation only uses graphs and proves that graphs can be effective in defining a real world automation problem.

Conceptually, information is either kept in a repository or used to communicate between two processes. Graphs are shown to work for both. This has proven to provide consistency and simplicity.

OWL Class Expression Using the OWL class expression to define a graph is very effective. First, it provides the structure of a graph by defining the classes and properties to be included in a graph. Second, the graph may be populated with individuals within the classes and properties given from a repository or by services using SDO.

Graphs defined in this manner are used for querying the repository, and for defining the messages processed by the framework. In querying the repository, a unique feature is the ability of recursion. Recursion is useful for retrieving a chain of relationships. It can also result in unexpected large responses when a chain of relationships is not desired.

The OWL class expression has proved to be very flexible. In the simulation, a single query is used to retrieve from the repository all the store ingredient locations that have fallen below their minimum required quantity on hand. Other single queries are used to retrieve the contract exchange data needed to restock the stores, and to update the location quantities and financial accounts.

Defining messages as OWL class expressions resulted in letting the requirements of OWL dictate the validity of the message structure. Messages as graphs have proven to fit the real world view of information flow.

Reasoner Validation Maintaining a repository where all of the content meets the rigorous requirements of OWL is accomplished by the application of a reasoner. In the simulation, all changes applied to the repository are required to pass the reasoner validation before the repository is changed.

This validation proved beneficial during the testing of the simulation. Any process errors would be caught by the reasoner and rejected. This lets the knowledge base of ontologies set the bar for the quality of a repository's content.

Service Data Objects The SDO standard, which is designed to access and modify a repository of information, proved to be an excellent

fit for graphs derived from OWL class expressions. In the infrastructure prototype, this form of graph is defined as a KBSgraph.

The framework prototype proves that SDO graphs can be defined using the OWL class expression. SDO includes the concept of root elements where elements have data content and relationships with other elements. The prototype framework successfully maps elements to OWL individuals, data content to data relationships, and element relationships to object relationships.

In the framework prototype, an empty graph may be created, or a graph that is fully populated with the elements defined may be created. The simulation proves the need for both capabilities as it adds test data to the repository by populating empty graphs and retrieves account information in populated graphs for updating.

All the capabilities to create, read, update, and delete elements is supported by SDO. These are standard functions of SDO, and they are implemented in the framework and proven in the simulation.

Once a graph has been processed by SDO, it can then become a request to update the repository. Proven in the simulation is that it will reject any change to an element that may have changed since it was retrieved, or if the graph fails the reasoner validation.

Service Component Architecture The infrastructure prototype proves that SCA can be defined in ontologies and used to reference and execute services. In the simulation, all the parts that make up SCA for the execution of the simulation are defined in multiple ontologies utilizing a common ontology that defines the structure of SCA.

REA REA is an excellent model for monitoring the performance of an organization. In the simulation, the exchange and convert models were used for all processing functions. Pizzas are ordered and produced using the convert model and then the payment from a customer is accomplished using the exchange model. The exchange model is also used to restock the pizza ingredients for the stores.

Layered Knowledge The simulation shows how layering ontologies using the OWL import capability provides a means of organizing business knowledge. In the simulation, all the business domains are organized into implementation, logical patterns, and message patterns.

By separating a business domain in this manner, the simulation proves that restricting ontologies from having access to other ontologies provides added security. In the simulation, only those ontologies requiring access or update capability imported the implementation or logical pattern of another business domain. Most of the ontologies in the simulation only have access to the message pattern ontology which describes the services and their message content.

Composite Sockets Composite sockets add another level of security by only providing access to defined services at the time of logging into the system. In the framework prototype, the identification of a person determines what access they are given.

Because all of the messages are in the form of KBSgraphs, access through a socket follows a single format. This has proven beneficial in simplifying the development of browser-based access.

Messages In the simulation, messages are used for sending requests and receiving responses. The messages are serialized into JSON for external communications, and transferred internally as KBSgraphs.

The messages sent and received by the simulation through the socket proves that serialization works. Messages passed between internal functions in the simulation assembly component proves that KBSgraph transfers also work.

Because messages have a source KBSgraph and a target KBSgraph, the simulation proved that not only would the contents from the source be moved to the target, but that the initial target KBSgraph is transformed by merging the structures of the source and target into a new KBSgraph. This was done in the simulation when retrieving a response from a service and not requesting all the information available.

Business Process Although the simulation does not contain a business process using BPEL, this capability was included in other testing. The significance of this demonstration is that an ontology can be used to define a stepwise process language that is executable.

Real-time Visualization The visuals provided by the prototype framework and the simulation prove that it is possible to see inside the software. The prototype visual tools give insight to the structure of the ontologies and SCA. Also, the simulation has visuals showing the results of processing actions.

Real-time visualization is proven, but this only represents an initial step. That step is important since it shows that the engineering diagrams that describe a KBS application should actually come directly from a repository that is the KBS application. This is very different today with diagrams being produced separately from the actual KBS systems.

6.5.2 Technology

The technology provided by Protégé is adequate for a prototype, but an industrial strength version of the framework needs to be developed if it is to be used in a productive manner.

The following are some of the major technology issues that must be solved before going forward.

Multitasking Reasoner One of the greatest benefits of the framework is that all changes made to the repository are first validated logically before they are applied. It is also the primary bottleneck in the overall performance. In the prototype, all requests are single-threaded through the reasoner to prevent overlapping changes.

There should be single-threaded reasoner operations when the same elements are involved in an update, but only when the same elements are involved. Otherwise, the reasoner should be able to multi-task.

Also, the reasoner could distinguish between updates to individuals rather than updates to ontologies. This would allow the reasoner to reuse the validation structures from the ontologies and only validating changes made to individuals.

Graph Repository A high-performance graph repository needs to be used to deal with large volumes of data with many element relationships. It should work for small data as well as big data and support the basic set operations of union and intercept.

Dynamic Updates In the infrastructure prototype, the server must be restarted if any ontology or service implementation is modified. This is currently required since the server starts up and predefines all service links.

Any changes to ontologies, including changes to individuals, should be validated and applied dynamically. The server should recognize any changes in SCA individuals, as well as reload any changed implementation code while running.

Visual Editors Only the Protégé editor was used in developing the prototype and the simulation ontologies. This approach was tedious and prone to errors.

There is a need for visual editors to define SCA structures and KB-Sgraph OWL class expressions. These were not necessary to test the prototype and the simulation, but would be necessary for a production use of the framework.

As mentioned earlier, this is an area where creative ideas can make development more like playing a game. Success in this area would probably determine the overall success of the infrastructure usage.

Part III

Transition to the Future KBS

Chapter 7

Next Steps

The next steps are to shift the development of KBS applications primarily to domain experts. This would be a major change to the project methodology as applied in today's development of systems. Domain experts would take on full responsibility for the creation of a system, and in turn would drive the execution of the framework.

To accomplish this shift, the next steps must provide both a delivery system and useful content. The recommendation is to have a delivery system based upon an open collaborative sharing of knowledge among professionals with domain expertise. Most of these collaborations could be made available for no charge, although contributors of implementation components should have the option whether to provide them for free or at a price.

This section presents a more detailed plan for the next steps.

7.1 Introduce The Knowledge Domain Component

Over the last few decades, the IT industry has sought to find some form of component in software development that could be reused by multiple projects. The reasoning is quite simple. Reuse of components should reduce development cost since they only need to be developed

once. Also, the quality of the components should improve with regular updates and bring better overall quality to the delivered system.

7.1.1 Historical Components

As the complexity and size of applications have increased, different components and their methods of communications have been defined.

Modules Modules are blocks of code that may be used by multiple programs by calling upon the module's API (Application Program Interface). This type of component is very useful for capturing unique processes.

Device drivers are modules that provide access to printers, disk devices, networks, etc. These drivers follow a standard API, so the program producing the input and output does not need to know the technical specifics of the device in order to use its functionality.

Object-Oriented Object-Oriented programming was a paradigm shift from procedural programming techniques. It introduced the concept of defining information within object classes which have attributes and methods. Sub-classes can be defined that inherit the methods of their super-class. This approach provided a means of defining not only technical objects, but business objects as well.

Object classes contain attributes in data types and / or attributes of relations with other object classes. They may also contain methods. The methods are implemented in program code that takes action on the containing object. Program code that has access to an object can call upon the methods exposed to the program code. The methods are the API of the object's class.

This structure of objects provides many advantages in designing systems. Most importantly, it isolates the functionality of an object to the attributes and methods of the object. And, due to inheritance, objects can be defined to perform differently from the same method call.

Object components have some reuse in the business arena, but the greatest usage has been found in implementing reusable infrastructure code. For example, libraries of class definitions are available in support of all of the basic functions needed to run a website. These include the components for message flow, security, protocols, visual components, and access to databases.

SCA SCA components are program code that can be requested to perform multiple services that may return a response. If another component's services are needed by a service, it can access the service from a reference. Components can be combined into an assembly component by using external control statements to connect references and services, and by promoting services or references within the assembly component.

Database Databases provide a means to isolate information into controllable units. In a relational database, the units are tables. In an XML database, the units are XML documents.

In application design, where services might directly relate to the components identified in the logical database model of a system, the database structure and the SCA structure are closely mapped together.

7.1.2 The Knowledge-Domain Component

A new component, the KDC, supported by the framework and defined in the OTTER project, is a domain taking a form at the next highest level. It encompasses modular, object-oriented, SCA, and database logical data modeling. As with each of the component types, reuse capability of KDC reduces cost and improves quality.

A KDC represents a single knowledge domain but may be dependent upon many other knowledge domains and therefore may represent a composite of knowledge. Since OWL provides the ability to import external ontologies, a single KDC may import multiple KDCs.

Consequently, although a KDC may be small in its actual content, it may import multiple knowledge domains and in total be very large, since KDCs may be entire applications assembled from multiple domain components. This type of component competes with traditional applications, but does not have the high cost of integration when all components are domains.

KDC Content A KDC is a single OWL ontology that may contain one or more of three parts. Each part is defined in the OWL language:

- **Meta** - The metadata describing classes, object properties, and data properties
- **Data** - Individuals within the meta classes
- **Action** - Service Component Architecture

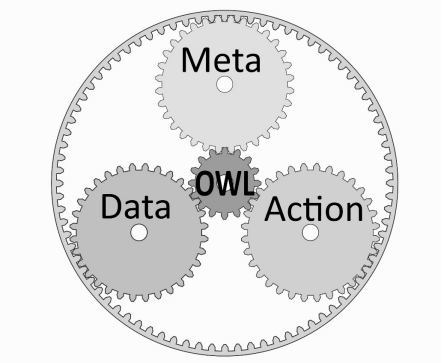


Figure 7.1: Knowledge-Domain Component

As shown in Figure 7.1, the Meta provides the full definition of all the data stored and the content of messages. Any data must satisfy all the axiomatic requirements given and exist with the defined classes having only the defined properties. This validation is accomplished

by the use of computerized reasoners that build logic trees and check for inconsistencies in the Meta, the Data, and the Action.

In OWL, the Data is defined as individuals. In a KDC, all individuals are stored as OWL-named individuals for access and update. Each individual may be a member of one or more classes and have many data properties or object properties, with object properties being the relationships between individuals. Individuals are persistent storage that is kept in data repository facilities that can handle efficiency for large volumes of information.

Action is based upon the Service Component Architecture standard, SCA. The SCA component structure and messages are defined in OWL. Processes are defined using an ontology based upon the Business Process Execution Language, BPEL. Services are defined in a programming language such as Java.

In summary, a KDC is a form of metaphysics in its content. The "meta" is descriptive in OWL, the "physics" are prescriptive in SCA, and the state of the metaphysical domain universe is defined by the data.

KDC Sub-Ontologies The three parts that make a KDC may be developed as separate ontologies, as done in the pizza store simulation. The parts are brought together by importing sub-ontologies into the data ontology following the approach as shown in the following:

Meta The meta is a specific area of knowledge that may have multiple sub-domains. These areas are often referred to as subject areas, each having subject-area domain experts. For example, if the area of knowledge is software development, it may have sub-domains for project management, design, testing, quality assurance, and implementation.
Action Action ontologies describe the messages and components used within a domain. The messages may be thought of as the interface to the domain. In design, an ontology developer could expose only the domain entities needed for particular messages. This could be an independent ontology or it could be based upon meta ontologies within the domain.
Data The data ontology is build upon the meta and action ontologies. It includes the configuration needed to access the individuals within the ontology and the software definitions for the services. Implementation ontologies in the framework are only accessible by those with security access. A service that reads and updates individuals would need to have access, while a player may only be limited to accessing services using the action ontologies.

7.2 Improved System Development Life Cycle

The System Development Life Cycle (SDLC) has followed the same approach applied in other engineering-based development. Essentially, the SDLC defines the processes and documents required to maintain communications between domain experts and the engineers providing the application.

7.2.1 Traditional Methods

Organizations usually face the "build or buy" question when a new application is needed. They can build it from the ground up, or it can be purchased. Purchasing has multiple alternatives. It can be installed at the organization's location or in the cloud. It can be used as a software or as a service. And, it may need modifications before it fits the organization's requirements.

The SDLC In the traditional method, those knowledgeable of the purpose of the final product, the domain experts, communicate with the engineers. The engineers then produce the engineering documents required to show their understanding of the final product, and to give a technical view that both the engineers and the domain experts can understand.

These documents become the basis for developing a project plan that shows the tasks to be performed by the developers. The "Waterfall" method follows the steps of defining requirements, designing the system, developing the system components, testing the system, and implementing the system into production. For some projects, a form of an "Agile" method may be applied. In "Agile", the development process is iterative through each of the "Waterfall" steps where the requirements are adjusted, as understandings of the need may change due to new understanding.

No matter which project methodology is followed, it is up to the project manager to be aware of the project's scope as defined in the requirements. If a change is made to the requirements, then the project

scope may change, possibly resulting in schedule and cost changes. The term "scope creep" is used to describe the circumstance when the changes result in higher costs or delays in the schedule. It is also the prime reason for major project failures.

Purchased Applications A common approach by most organizations is to purchase the applications they need. Organizations generally only apply the traditional SDLC methodology when a unique solution is required.

The selection of applications follows a common methodology by an organization issuing a RFP (Request For Proposal) from application providers. The RFP is basically the SDLC requirements document. The vendors can then respond with pricing, which may include extensions that must be applied to the application. The response to the RFP is called a vendor bid. The organization selects from the vendors based upon their bids. For government organizations, the selection is usually made on lowest price.

Application Integration Bringing a new application into an organization, whether its a build or buy, is usually not a standalone issue. The application generally needs to be integrated with other applications. This is often the most challenging of all the activities of implementing a new application.

Conversion from Old to New Conversion to a new application is the process of transferring stored information from one system to another. The source of the information may be automated or manual, but it usually requires careful review and generally must be transformed in some automated manner.

7.2.2 New Knowledge-Based Method

The knowledge-based method of building and maintaining systems would not follow the traditional approach. There is no requirements

document to be used as the communications vehicle between domain experts and developers, or to be used to issue an RFP. Instead there are executable models built using iterative development cycles. These models define both the requirements and the methods of execution.

Jan Dietz[1] introduced a methodology for using ontologies as the base for defining systems requirements. He describes a methodology containing four models. Listed below are the models and how they are implemented in the framework:

- **Construction Model** - This model is developed in the framework using the business model ontology.
- **Process Model** - In the framework this model is developed using the SCALite ontology.
- **Action Model** - Actions are implemented as services in program code or processes defined in BPEL.
- **State Model** - State is maintained in the framework by the individuals within the data ontologies.

Following these models, the knowledge-based method supported by the framework contains three steps that may be performed iteratively:

- **Step 1:** Select or create the domains needed
- **Step 2:** Select or create the domain Implementations
- **Step 3:** Select or create socket access

In each step, the domain experts and the enterprise architects collaborate. For personal systems, and possibly even small business systems, the domain expert and the enterprise architect may be the same.

[1] Jan I.E. Dietz, *Enterprise Ontology*, 1998, Springer

Domains Needed An organization would utilize existing domains to assemble the knowledge that meets their needs. In those areas where knowledge has not been placed into the structure determined by the framework, new domains must be created.

The creation of a domain requires domain experts and enterprise architects. The domain experts are responsible to make sure the domain ontologies correctly define the business and the professionals needed by the business. Enterprise architects would focus on reuse and additional message flow definitions by preparing the SCA components.

Implementations Selected Since the framework separates the definition of an application from its implementation, there could be multiple alternatives for implementing the services defined in SCA components. The alternatives may include algorithms designed for specific operating environments or processing volume. The selection would be made in collaboration between the domain experts and enterprise architects.

Socket Access Selected Sockets are linked from external applications. The application may be framework-based, mobile, or browser. For each of these, solutions may already exist in the repository.

When it is necessary to create a socket access, there should be utilities available to develop what is needed. For mobile-based and browser-based, these utilities would provide the ability for domain experts to define human interfaces.

Incorporating New Knowledge Traditionally once a system is developed, it falls into the maintenance stage. With knowledge-based systems, there is no difference between development and maintenance. Both development and maintenance are efforts needed to record knowledge in a machine-readable form. Development is accomplished by iteratively enhancing the knowledge captured, and maintenance continues this effort.

7.3 Encourage Domain Expert Sharing

The key to success lies in sharing knowledge by providing an open environment for domain experts. This open environment, built upon a social media structure, should have free usage with limits and royalties for some component creators.

7.3.1 Social Media Structure

Knowledge building should be driven through social media sharing and collaboration. Through peer review, the experts should determine the structure and the content of knowledge.

Library of Ontologies For reuse, ontologies must be cataloged so they can be found. The catalog would be similar to cataloging books in a library. Each ontology could be found in only one location in the library. Ontologies needing knowledge from other ontologies in the library would incorporate them by using the OWL "imports" statement.

Having only one location for an ontology is important. When there is a concern that the ontology could go in multiple locations, this is a strong indication that the ontology needs to be divided into smaller ontologies with different classifications.

KBS systems are constructed to serve us. Consequently, ontologies should be organized around how they will serve us in our academia, professions and occupations. Taking this view, we only need to examine the existing knowledge classifications we use to describe the areas in which we need service.

Existing taxonomies can be used to provide the high-level indexing for the library for the layers of infrastructure, academic disciplines, occupations and professions, industry, and enterprise. These layers are shown in figure 7.2. Each of these index levels could then be sub-divided utilizing the existing taxonomy codes as described in the following.

Figure 7.2: Enterprise Layers

Infrastructure Infrastructure classification contains the ontologies supported by the framework. These ontologies are the basis for execution and are directly aligned with the functionality of the framework.

Academic Disciplines The ontologies in this layer may be classified by using the Classification of Instructional Programs (CIP 2010) maintained by the Federal government and universities.

The CIP is a taxonomic coding scheme that contains titles and descriptions of primarily postsecondary instructional programs. It was developed to facilitate NCES(National Center for Educational Statistics)[2] collection and reporting of postsecondary degree completions by major field of study using standard classifications that capture the majority of reportable program activity. The CIP: 2000 edition is the third revision of the CIP. It was originally published in 1980 and was revised in 1985 and 1990. The CIP is the accepted federal government statistical standard on instructional program classifications and the 2000 edition has been adopted as the standard field of study taxonomy by Statistics Canada.[3]

Occupations and Professions This layer is built upon the ontologies of academic disciplines and other occupations. These ontologies would not only include the services needed to be used by a professional, but also the services provided by the professional to the KBS system.

This classification may be best provided by the Standard Occupational Classification and the National Career Clusters^{TM}Framework.

[2]http://nces.ed.gov/

[3]https://nces.ed.gov/ipeds/cipcode/Default.aspx?y=55

Both of these have been cross-walked from the CIP 2010 classifications.

The 2010 Standard Occupational Classification (SOC) system is used by Federal statistical agencies to classify workers into occupational categories for the purpose of collecting, calculating, or disseminating data. All workers are classified into one of 840 detailed occupations according to their occupational definition.[4]

The National Career Clusters™Framework provides a vital structure for organizing and delivering quality Career and Technical Education Programs of Study through learning and comprehensive programs of study. In total, there are 16 Career Clusters in the National Career Clusters Framework, representing more than 79 Career Pathways to help students navigate their way to greater success in college and career.

As an organizing tool for curriculum design and instruction, Career Clusters provide the essential knowledge and skills for the 16 Career Clusters and their Career Pathways. It also functions as a useful guide in developing programs of study bridging secondary and postsecondary curriculum and for creating individual student plans of study for a complete range of career options. As such, it helps students discover their interests and their passions, and empowers them to choose the educational pathway that can lead to success in high school, college and career.

- Agriculture, Food & Natural Resources
- Architecture & Construction
- Arts, A/V Technology & Communications
- Business Management & Administration
- Education & Training
- Finance

[4] http://www.bls.gov/soc/

- Government & Public Administration
- Health Science
- Hospitality & Tourism
- Human Services
- Information Technology
- Law, Public Safety, Corrections & Security
- Manufacturing
- Marketing
- Science, Technology, Engineering & Mathematics
- Transportation, Distribution & Logistics

5

Industries Industry domains would be defined for specific types of industries by including the occupational or other industry ontologies needed. Occupational ontologies would be included and extended to meet the specialized needs of the specified industry. For example, the profession of accountant would be extended as needed for the industry type defined.

Industry classification can be accomplished using the North American Industry Classification System (NAICS) which has been crosswalked to the Career Clusters.

NAICS is the standard used by Federal statistical agencies in classifying business establishments for the purpose of collecting, analyzing, and publishing statistical data related to the U.S. business economy.[6]

[5] http://www.careertech.org/career-clusters
[6] http://www.census.gov/eos/www/naics/

Enterprise The enterprise layer includes the business domains that make up the enterprise. Each enterprise domain is unique and applies to only one enterprise. In the case of an enterprise responsible for multiple enterprises, each of the enterprise ontologies might be imported along with the industry ontologies needed to manage multiple enterprises.

Layer Dependency The layers organized using the classification codes presented in this section are shown in 7.3 at the highest summary level of each classification group. Read from left to right, the first layer is the academic disciplines layer using CIP 2010, the second is the occupations and professions layer using the SOC and the career clusters classifications, with the last the industries layer using the NAICS codes.

Each of the high-level classifications of each layer is shown with multiple connections. This visually shows the significant potential for reuse of ontologies registered within each of the classifications.

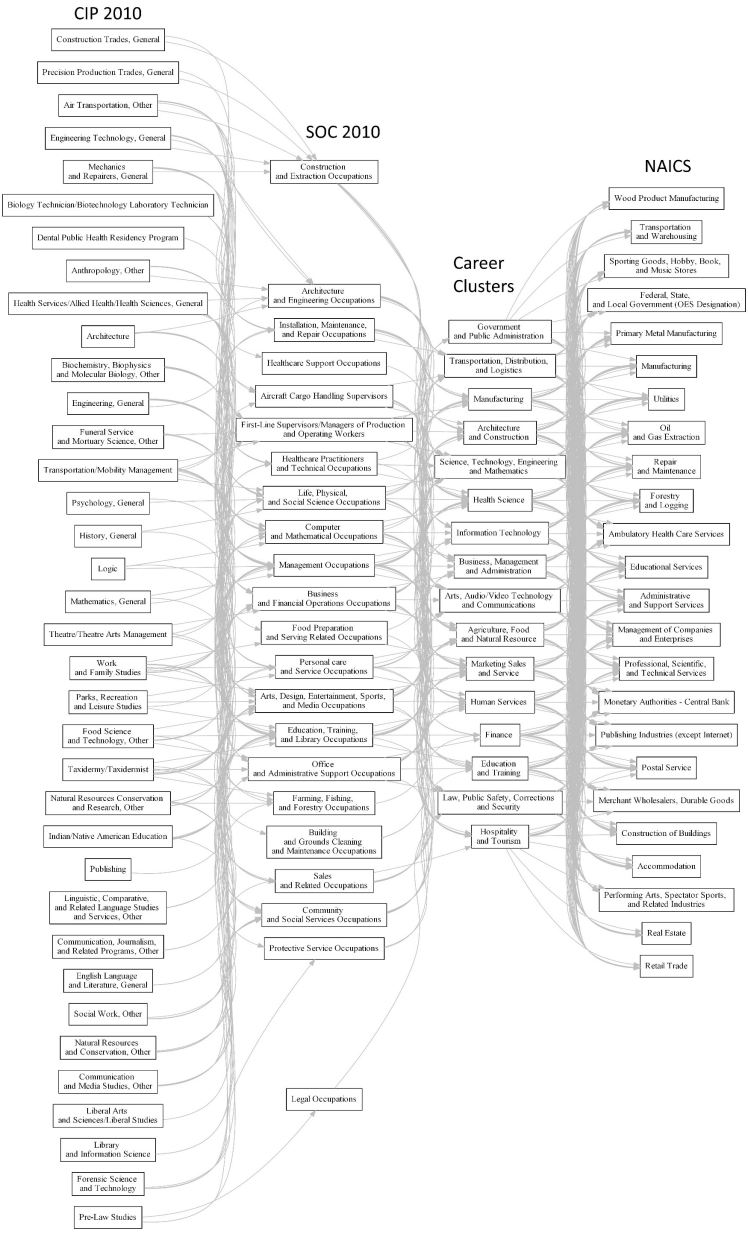


Figure 7.3: Code Layers

Domain Parts Domain components are made up of multiple parts: ontologies, implementation, and socket access.

Ontologies Ontologies are OWL documents that are prepared by following the requirements of the KBS framework. These documents represent an area of knowledge, and are given an axiomatic content to prove the validity of individuals to exist within the domain.

These documents also fully define the SCA of the domain, including all message content.

Implementation There may be multiple forms of implementation, using different programming languages or meeting specific performance criteria.

Implementations provide algorithms to process the messages described in the ontologies. The implementation may also access and update the repository of information as defined by the ontologies.

Socket Access A domain may include specific program code to access sockets defined in the SCA of ontologies. The code may be designed to be used by another KBS application, or for human access using a mobile or browser-based application.

7.3.2 Free Usage (with limits)

Usage would require membership and acceptance of within a service level. The entry level for access should be free for collaboration and sharing of domains, although usage of a created KBS application could have limits.

Knowledge Developers Collaboration and sharing are the foundation of building a repository of knowledge. Consequently, this access and usage should have few restrictions. More than likely, there would be a free service level, but to get additional features some form of membership upgrade might be required. Upgrades would be billable.

KBS Application Usage A quantity of storage and processing time would be available at an entry level for free. Going over this level would require an upgrade in membership.

7.3.3 Provide Royalties on Purchases

Some components may require a purchase to view and use. This protects the work produced by business groups, and implementations provided by professional developers.

Business Group Models Business groups typically require membership to gain access to their repositories of information. These groups include business domains such as accounting, financial, supply-chain, etc.

These business groups would determine their own membership and access levels to their domains.

Business Component Implementations Business component implementations may be free or require a purchase. Purchasing might provide more features than a free version of the same implementation.

7.3.4 KBS for Collaboration and Sharing

The social media system should be built as a KBS application utilizing the framework.

Features The KBS application should have the following features:

- Accounts for domain experts
- Domain expert profiles
- A library of knowledge domains
- The creation and updates of domains
- Expert groups for creating and updating domains

- The creation and updating of KBS applications
- Execution of KBS applications

These features could be used by an individual for personal applications, or scaled to small business or large business usage.

Application Access Access to the application would provide domain experts with the ability to take advantage of the features of the application.

Presentation and Edit All aspects of domains should be viewable and editable. Full advantage should be taken of easy-to-use displays and visual updating.

Game-Oriented Usage Game-oriented design would be preferable for all viewing and editing. Particularly the use of 3D to allow for working in a virtual reality scene as though entering into the interworkings of the domains.

Web and App Access Access should be available from a web browser or from a mobile app. The opportunities are endless due to the speed and capacity of devices that may be used to access the application.

7.4 Next Development Steps for Framework

The next technological step for the prototyped infrastructure is to provide it in an industrialized form. This is a significant project. It involves building an environment capable of extreme volumes of storage and messaging. It must also provide a repository implementation with the ability to verify all changes against the imported ontologies. It will require human-friendly socket access as well.

7.4.1 Framework Industrialization

This section describes some of the major initiatives needed to industrialize the framework.

High Performance Repository All storage of information is done in graph form. The prototype uses storage that requires everything to be kept in memory. The industrial version must keep all information in external storage with only the most active data maintained in memory.

Multi-threaded Reasoner The reasoner must verify all ontologies and maintain maps of axiomatic relationships. Using these maps, verification processes could be executed using multiple threads. However, if there is axiomatic dependency between changes, the changes should be single threaded.

Implement in the Cloud Implementation should operate in the cloud. This will provide the ability to quickly react to any need for more computing power or storage resources.

Access to Other Repositories The industrial form of the framework should assume that multiple instantiations of the websites utilizing the framework will exist. Consequently, access to networked repositories should function the same as local repositories.

This should be accomplished by adding the "Plug" template to the SCAlite ontology. As shown in Figure 7.4 where an external composite socket's references and services are exposed for access and implementation.

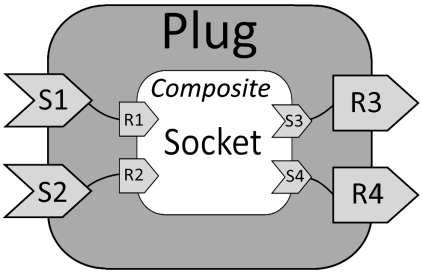


Figure 7.4: Plug Template

Using the "Plug" template, KBS applications using the Otter framework may be networked as shown in Figure 7.5.

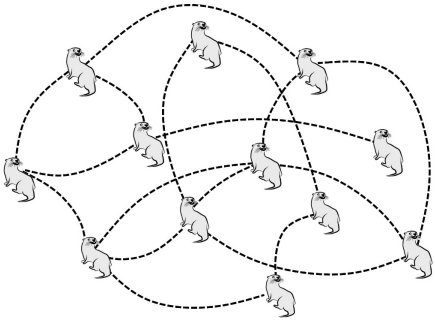


Figure 7.5: Networked

Networking KBS applications provides a means for sharing information in an ecosystem for business and science and even the Internet of Things.

Events To extend the capability of the framework to handle events, even complex events, the service as defined in the SCAlite infrastructure ontology could be extended. A request-only service could

be described as an event. In doing so, the framework would monitor the existence of any individuals satisfying the service's request KBSgraph. In other words, when the KBSgraph is not empty, the service would be initiated to process the event.

Website Design In the REA model, an agent can be viewed as an individual or as an external program needing to access the KBS application. Their access provides the means to provide the events that change the resources through services controlled by the OTTER framework.

Although the framework supports a standard website delivery, it does not include functionality to support designing a website of a KBS application. Design would include the creation and storage of HTML, Style Sheets, and JavaScript.

Since there are already multiple website design tools, this capability might be provided by supporting existing tools with extensions to access SCA using socket components.

7.4.2 KBS Delivery System

The social network system described previously would need to be defined following the requirements for using the framework. This would include developing ontologies, developing the SCA, and developing the socket access.

Develop Ontologies The ontologies would describe the capturing of knowledge, as supported by the framework and used by domain experts in a social media structure. Essentially, these are the domains of ontologies for developing ontologies.

Develop SCA Using the SCA ontology, the processing components of the domain would be described and the services implemented. This would provide the application processing for the website.

Develop Socket Access The prototype provides socket access for viewing components and accessing services. This must be enhanced to include the additional social network capability.

Enhance Security with REA Security can be enhanced by applying the REA model to restrict an agent to access only the information contracted. Not only would an individual be restricted by the services available within their composite socket, they would also be limited by contract to the specific content of a domain.

7.5 Organic Growth

A knowledge base of domains in the form required to execute within the framework will take time to grow. Therefore, an organic approach to building knowledge is the best alternative. This will provide an opportunity to verify the performance of the framework, and gain valuable feedback from domain experts in the early stages.

The following is a potential categorical list for domain development in the order of occurrence:

- **Big Data** - Provide access to some of the most used big data.
- **Research Projects** - Offer ontologies for running research projects. This would include the analytical domains and reporting domains.
- **Academic Disciplines** - Some of the most basic disciplines, such as accounting, would be defined by domain experts.
- **Personal Accounting Management** - This would offer individuals a means of managing their accounts for budgeting and planning.
- **Small Business** - This would begin providing functionality for specific small businesses.

- **Large Business** - As domain experts provide sophisticated domains for complex and large businesses, large businesses could begin utilizing the framework.
- **Internet of Things** - The existing ontologies for the Internet of things could be updated to operate within the framework as infrastructure. Business domains having specific information available would be developed by building upon this infrastructure.

Growth in a social network setting is difficult to predict. The assumption is that after an initial introduction of an infrastructure that includes minimal reusable domains, the growth would be slow. However, given the potential of crowd sourcing, growth could move rapidly in an unexpected direction.

Chapter 8

Summary

The standard approach of an Enterprise Architecture evaluation - the current state of knowledge-based systems is presented, followed by a description of the future state, and then a demonstration is made which leads the way to a transition - has been applied in this book. The demonstration was presented in detail as a prototype. Then, with the proof of the prototype, the transition from current to future action items was described.

8.1 Current State

The current state was presented as:

- Thousands of people attend conferences each year to share ideas on how knowledge can be captured and shared. This is a strong indication of the growing demand for the development of knowledge-based systems.
- The current standards that are developed give a provable mathematical base to the information that is captured.
- There are multiple large repositories and applications that freely share information, and this is continuing to be expanded.

- Knowledge sharing is done within specific application domains.
- Sharing across application domains remains a limitation to sharing knowledge.

The current state is a heads-up for all those that only focus on reaching a single destination. Join the crowd, look up, and recognize that KBS is changing our society. It is the best approach we have for building the very large complex systems upon which our society depends. However, there is a big problem with today's implementations. Our society wants fully integrated information, yet it is difficult to integrate the knowledge from multiple domain applications.

8.2 Future State

The future state shows that society has been slowly moving upwards on what promises to be the starting incline of a wild roller coaster ride. Once the top of the first elevation is reached by having a framework for KBS applications and a taxonomy for the library of knowledge, the fast-paced ride can begin.

The high speed of the coaster will come from the new development paradigm for KBS as a result of a common framework. It will reduce development time for KBS applications. The quality of knowledge will control development and execution. There will be common, easy-to-use tools, and knowledge will be available to everyone. Anyone with a great idea will have the means to test that idea. Researchers will be able to share information across domains more quickly and easily.

One of the biggest changes will be in Enterprise Architecture. Executable ontologies will replace models by allowing visual inspection of all software computing resources. Governance will be turned over to the logic applied by the KBS framework.

Society will suddenly gain all new possibilities for computing resources to guide and help us in everyday life.

8.3 OTTER Prototype

The OTTER project prototype was described to include the simplicity of using the OWL class expression to form graphs, having infrastructure ontologies, and visualizing the KBS application. To prove key concepts that the prototype framework can be used in real situations, a simulation was developed and run. By utilizing the value of the REA models, multiple pizza stores serving customers and purchasing ingredients from suppliers was demonstrated.

Simplicity of KBSgraph The simplicity of the framework is found in the class expression. This single statement is used to define both queries and messages. Those that learn how to use the class expression enhance their ability in all areas. They move closer to understanding the knowledge upon which the queries and messages rely.

The KBSgraph cornerstone, upon which all aspects of the framework are aligned, is more than a format for information. It is a simplified view that reduces the complexity of accessing and processing information within the fullness of knowledge.

Prototype Ontologies The ontologies of the OTTER framework include business, services, and process. "Services" is the primary ontology for framework implementation. The service ontology is used by the business ontology to define the business components as either providers or consumers of services. The process ontology defines a set of commands that are supported by the framework and operate within a service environment.

Visualizing KBS Applications Visualizations peek into the reality of KBS applications. They are not blueprints of something to be constructed. They are snapshots of what has already been constructed and is currently operating within the framework.

A challenge exists to find a way that humans can see the knowledge captured in the domains. Since OWL and the framework ontologies

are the forms which the knowledge takes, the visualization must come from those sources.

The OTTER project demonstrated visualizations for domains, classes, graphs, services, sockets, processes, and more. Yet the question is still open: what should the overall view of a KBS application be? Possibly there will be many different views, depending upon the range of domains included in a KBS application. It is also possible there may be many different analogies selectable by the viewer.

The nature of how to view a KBS application has more possibilities than restrictions. This calls for unlimited creativity.

Simulation The pizza store simulation demonstrates a transaction-based operation using executable ontologies. The demo uses the Resource Event Agent (REA) academic disciple ontology to create a knowledge-based system for multiple pizza stores that use multiple providers of supplies.

The key concepts proven in the simulation are:

- **Graphs** - The structure of graphs fits well into the real-world view of information processing.
- **OWL Class Expression** - Using the OWL class expression to define a graph is very effective.
- **Reasoner Validation** - Maintaining a repository where all of the content meets the rigorous requirements of OWL is accomplished by the application of a reasoner.
- **Service Data Objects** - The SDO standard, which is designed to access and modify a repository of information, proved to be an excellent fit for graphs derived from OWL class expressions.
- **Service Component Architecture** - The infrastructure prototype proves that SCA can be defined in ontologies and used to reference and execute services.

- **REA** - REA is an excellent model for monitoring and controlling the performance of an organization.
- **Layered Knowledge** - The simulation shows how layering ontologies using the OWL import capability provides a means of organizing business knowledge.
- **Composite Sockets** - Composite sockets add another level of security by only providing access to defined services during the time a player is logged into the system.
- **Messages** - In the simulation, messages defined by class expressions are used for sending requests and receiving responses.
- **Business Process** - The significance of this demonstration is that an ontology can be used to define a stepwise process language that is executable.
- **Real-time Visualization** - The visuals provided by the prototype framework and the simulation to prove that it is possible to see inside the software. The prototype visual tools give insight to the structure of the ontologies and SCA. Also, the simulation has visuals showing the results of processing actions.

The simulation using the prototype framework is a success.

8.4 Next Steps

The next steps are to shift the development of KBS applications primarily to domain experts. This would be a major change to the project methodology as applied in today's development of systems. Domain experts would take on full responsibility for the creation of a system, and in turn would drive the execution of the framework.

The summary of next steps is:

- **Introduce The Knowledge Domain Component** - The KDC, supported by the framework and defined in the OTTER project, is a domain taking a form at the next highest component level. It encompasses modular, object-oriented, SCA, and database logical data modeling and through reuse reduces cost and improves quality.

- **Improved System Development Life Cycle** - The knowledge-based method of building and maintaining systems would not follow the traditional approach. Current approaches based upon business ontology development would be used.

- **Encourage Domain Expert Sharing** - The key to success lies in sharing knowledge by providing an open environment for domain experts.

  KBS systems are constructed to serve us. Consequently, ontologies should be organized around how they will serve us in our academia, professions and occupations. Taking this view, we only need to examine the existing knowledge classifications we use to describe the areas in which we need service. These taxonomies already exist for infrastructure, academic disciplines, occupations, and professions.

- **Next Development Steps for the Framework** -The next technological step for the prototyped infrastructure is to provide it in an industrialized form. This is a significant project. It involves building an environment capable of extreme volumes of storage and messaging. It must also provide a repository implementation with the ability to verify all changes against the imported ontologies. It will require human-friendly socket access as well.

- **Organic Growth** - A knowledge base of domains in the form required to execute within the framework will take time to grow. Therefore, an organic approach to building knowledge is the best alternative. This will provide an opportunity to verify the

performance of the framework, and gain valuable feedback from domain experts in the early stages.

The next steps include technical, social, and methodology changes.

8.5 Overall Summary

Two major conclusions can be reached:

- **Common Framework** - There must be a common framework to share knowledge and the OTTER project has demonstrated an excellent alternative.
- **Common Library System** - Knowledge must be shared within an organized library system that utilizes existing taxonomies of how KBS applications will serve us.

Once these goals are achieved, our potential to collect and utilize knowledge will rise to a level never before thought possible. And with that heightened potential, we will change everything. Our endless struggles to advance technology, fight disease, improve education, solve the riddles of the world we live in and of the whole universe beyond; all of these challenges and more will become so much easier to face and conquer once we effectively transform the world's knowledge into a tool which computers can understand and use, allowing them to join with us in the ecosystem of service to humanity. The OWL has shown the architect a shining vision of an incredible future filled with amazing achievements and discoveries, simply by answering the question **Why?** I trust that you can share that vision as well, now that you've seen there is logic and structure throughout the universe of Information Technology in *every* sector.

Part IV

Appendices

Appendix A

Semantic Conferences

The following list was found at `http://wikicfp.com` in April of 2015.

- Ontology Summit 2015 - Internet of Things: Toward Smart Networked Systems and Societies
- IJNLC 2015 - International Journal on Natural Language Computing
- ICKD 2015 - 4th International Conference on Knowledge Discovery
- AIAI 2015 - The 11th International Conference on Artificial Intelligence Applications and Innovations
- KESW 2015 - 5th International Conference on Knowledge Engineering and Semantic Web
- IC3K 2015 - International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management
- SEKE 2015 - The 27th International Conference on Software Engineering and Knowledge Engineering (SEKE2015)

- JIISIC-CEIS 2015 - XI Jornadas Iberoamericanas de Ingeniera de Softwara e Ingeniera del Conocimiento y 1er Congreso Ecuatoriano en Ingeniera de Software
- NLP 2015 - Fourth International Conference on Natural Language Processing
- ICKE 2015 - International Conference on Knowledge Engineering
- JSE 2015 - Fourth International Conference on Software Engineering and Applications
- ICK 2015 - International Conference on Knowledge
- VISUAL 2014 - International Workshop on Visualizations and User Interfaces for Knowledge Engineering and Linked Data Analytics
- SEAS 2014 - Third International Conference on Software Engineering and Applications
- BdKCSE 2014 - Big Data, Knowledge and Control Systems Engineering
- KESE 2014 - 10th Workshop on Knowledge Engineering and Software Engineering (KESE10) at ECAI 2014
- KSEM 2014 - The 7th International Conference on Knowledge Science, Engineering and Management
- 23rd IEEE WETICE 2014 - Track Web2Touch I-KNOW Workshops 2014 - Workshops for 14th Iternational Conference On Cognitive Computing And Data-Driven Business
- AIP 2013 - Acta Informatica Pragensia
- CIA 2013 - International Workshop on Computational Intelligence and its Applications

- KICSS 2013 - 8th International Conference on Knowledge, Information, and Creativity Support Systems
- SOFSEM 2014 - 40th International Conference on Current Trends in Theory and Practice of Computer Science
- ICCKE 2013 - 3rd International eConference on Computer and Knowledge Engineering (ICCKE 2013)
- IWDM 2013 - First IEEE International Workshop on Data Management
- KESE 2013 - The 9th Workshop on Knowledge Engineering and Software Engineering (KESE9)
- ASAI 2013 - Argentine Symposium on Artificial Intelligence
- OTM Workshops 2013 - OTM 2013 call for workshop proposals
- KSEM 2013 - International Conference on Knowledge Science, Engineering and Management
- SEKE 2013 - The 25th International Conference on Software Engineering and Knowledge Engineering
- MSEPS 2013 - International Workshop On Modeling States, Events, Processes And Scenarios
- vSI-KAMIoT-Automatika 2012 - Special issue on Knowledge Acquisition and Management in the Internet of Things
- ICT-KE 2012 - The 10th International Conference on IEEE ICT and Knowledge Engineering 2012
- IC3K 2012 - 4th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - IC3K 2012
- KESE 2012 - The 8th Workshop on Knowledge Engineering and Software Engineering (KESE2012) a.k.a KESE8

- NLP-KE 2012 - The 8th International Conference on Natural Language Processing and Knowledge Engineering
- URKE 2012 - International Conference on Uncertainty Reasoning and Knowledge Engineering
- EKAW 2012 - The 18th International Conference on Knowledge Engineering and Knowledge Management
- KEOD 2012 - International Conference on Knowledge Engineering and Ontology Development
- ICDMKE 2012 The 2012 International Conference of Data Mining and Knowledge Engineering
- Web2Touch 2012 - IEEE WETICE Track - Modeling the Collaborative Web Knowledge Conference
- KES-IIMSS 2012 - 5th KES International Conference on Intelligent Interactive Multimedia Systems and Services
- KES-IDT 2012 - 4th KES International Conference on Intelligent Decision Technologies
- AIKED 2012 - 11th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases
- COMPUTE 2012 - 1st Annual Conference of ACM Pune Professional Chapter
- ICTHC 2011 - International Conference on Advances of Information & Communication Technology in Health Care
- NLPKE 2011 - 7th International Conference on Natural Language Processing and Knowledge Engineering
- INAP 2011 - 19th Conference on Applications of Declarative Programming and Knowledge Management

- ICCKE 2011 - International Conference on Computer and Knowledge Engineering
- KSEM 2011 - The 5th International Conference on Knowledge Science, Engineering and Management
- STAKE 2011 - 3rd Semantic Technology And Knowledge Engineering Conference
- KEOD 2011 - Int'l Conf. on Knowledge Engineering and Ontology Development
- IC3K 2011 - IC3K- 3rd International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management
- KEOD 2011 - KEOD-International Conference on Knowledge Engineering and Ontology Development
- KESE7 2011 - 7th Workshop on Knowledge Engineering and Software Engineering
- ICDKE 2011 - International Conference on Data and Knowledge Engineering
- SEKE 2011 - The Twenty-Third International Conference on Software Engineering and Knowledge Engineering
- FOMI 2011 - 5TH Workshop on Formal Ontologies Meet Industry
- IKE 2011 - The 2011 International Conference on Information and Knowledge Engineering
- URKE 2011 - The International Conference on Uncertainty Reasoning and Its Application in Knowledge Engineering
- Inderscience IJKWI Special Issue 2010 - Call for Journal Articles: Inderscience Special Issue on Semantic Digital Library

- ICT&KE 2010 - 8th International Conference on ICT and Knowledge Engineering (ICT & Knowledge Engineering 2010)
- KKDEO 2010 - 1st International Workshop on Heterogeneous-Data Mining and KDD in Support of Earth Observation
- KESE 2010 - 6th Workshop on Knowledge Engineering and Software Engineering KESE6 at the 33rd German Conference on Artificial Intelligence
- NLPKE 2010 - The 6th IEEE International Conference on Natural Language Processing and Knowledge Engineering
- IC3K 2010 - IC3K - 2nd International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management
- KEOD 2010 - 2nd International Conference on Knowledge Engineering and Ontology Development
- ICDKE 2010 - International Conference on Data and Knowledge Engineering
- KSEM 2010 - The fourth International Conference on Knowledge Science, Engineering and Management (KSEM‘2010)
- IWAAPO 2010 - The International Workshop on the Applications and Advances of Problem-Orientation
- SemWiki 2010 - 5th Workshop on Semantic Wikis - Linking Data and People
- SEKE 2010 - 22nd International Conference on Software Engineering and Knowledge Engineering
- WMSCI 2010 - The SUMMER 4th International Conference on Knowledge Generation, Communication and Management

- MIS Review - Special Issue 2009 Knowledge and Software Engineering: A Seamless Interplay - A special issue of the MIS Review: An International Journal (MISR)
- iEMSs 2010 - Session on Intelligent and collaborative engineering of environmental knowledge: Software platforms, agents and semantics
- KESE 2009 - 5th Workshop on Knowledge Engineering and Software Engineering at the 32nd German Conference on Artificial Intelligence
- IEEE NLP-KE 2009 - IEEE International Conference on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE'09)
- IC3K 2009 - International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management
- SEKE 2009 - The Twenty-First International Conference on Software Engineering and Knowledge Engineering
- DEECS 2009 - 4th International Workshop on Data Engineering Issues in E-Commerce and Services
- KESE 2008 - 4th Workshop on Knowledge Engineering and Software Engineering at the 31st German Conference on Artificial Intelligence
- ReLAIS 2014 - Revista Latinoamericana de Ingeniera de Software

Appendix B

Business Process Execution

BPELlike is an OWL ontology applied by OTTER to provide business process execution. BPELlike is based upon the standard language BPEL (Business Process Execution Language). Figure B.1 is a Cord diagram showing all domain classes and relationships in the BPELlike ontology.

The BPELlike ontology contains activities linked together in the order they are to be executed. These activites are OWL individuals defined by the Activity class. The BlockActivity class defines OWL individuals containing activities linked together. An individual of class BlockActivity may include one or more sets of linked activities.

Most block activity classes inherit from both the Activity class and the BlockActivity class. Although there are some activities that only inherit from the BlockActivity class or the Activity class.

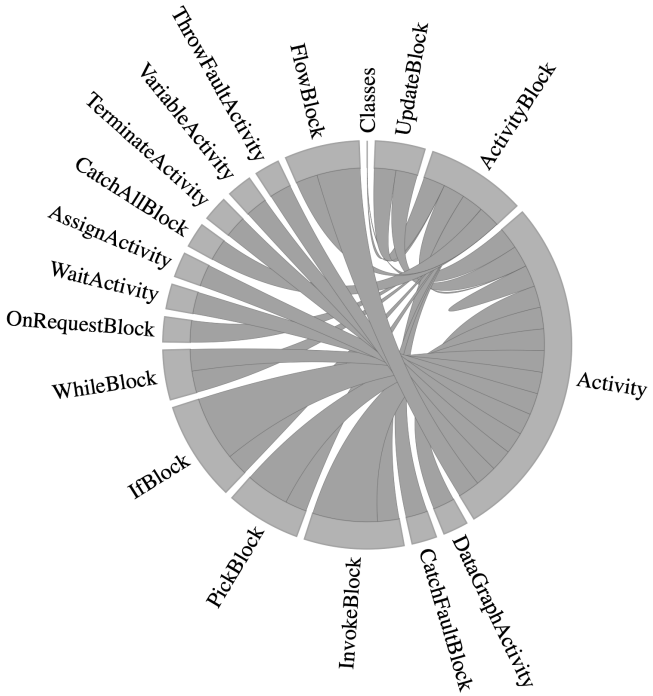


Figure B.1: Cord Diagram of All Classes in BPELlike Domain

The following is a list of all the process classes defined in BPELlike:

- OnRequestBlock - The service entry to a process.
- CatchAllBlock - Activities to catch and process all exceptions.
- CatchFaultBlock - Activities to catch and process specific exceptions.
- InvokeBlock - Activities to prepare for and to make a reference to an external service.
- UpdateBlock - Activities to access, update, and store individuals into an ontology.

- WhileBlock - Repeats activities until a specific condition is met.
- PickBlock - Starts multiple sub-processes simultaniously and completes when the first sub-process completes.
- FlowBlock - Starts multiple activities to execute in parallel.
- IfBlock - Selects the direction of process flow based upon a condition.
- VariableActivity - Creates a variable.
- DataGraphActivity - Creates a datagraph.
- AssignActivity - Provides for assigning values to variables and datagraphs.
- WaitActivity - Causes the process to wait a specified amount of time.
- ThrowFaultActivity - Throws a specific fault.
- Terminate - Ends the process.

Each of these process activities are described in separate sections.

B.1 Class Structure

The two primary classes in BPELlike are Activity and ActivityBlock. The process classes inherit from either or both of the primary classes.

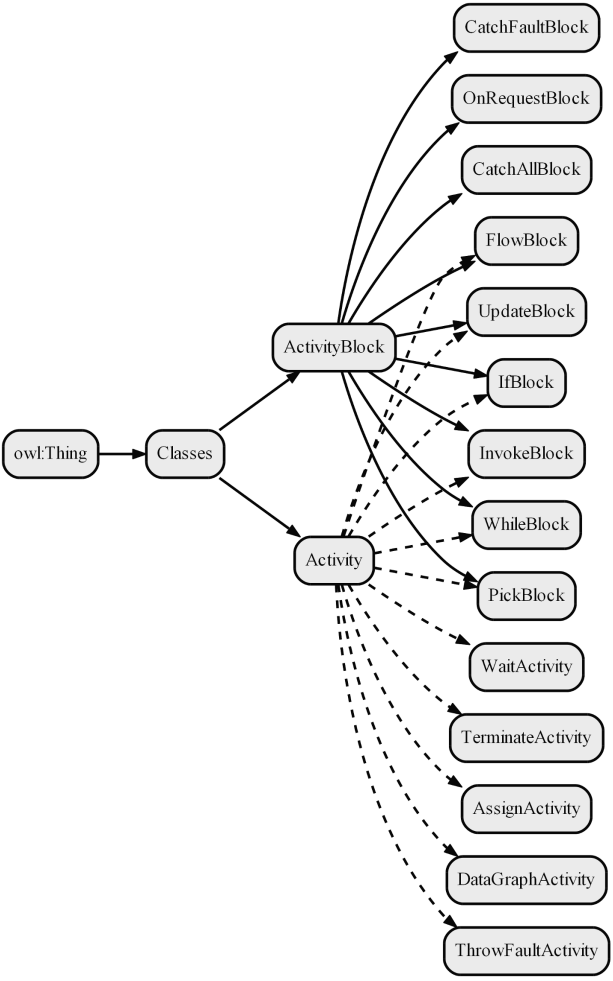


Figure B.2: Activity Class Structure

Figure B.2 shows the inheritance structure. The following describes the properties of each of these classes and their function in process execution.

B.1.1 Activity Class

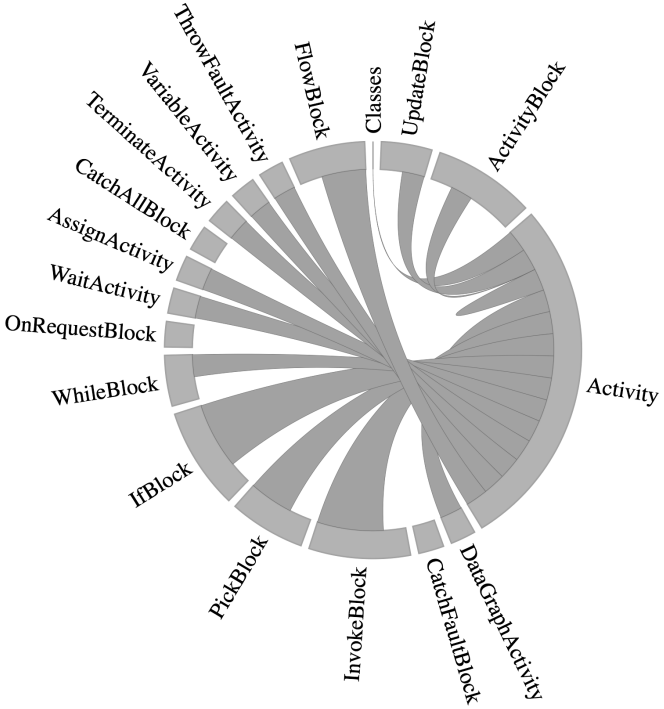


Figure B.3: Cord Diagram of Activity Class

The Activity class is a super class for most of the process activities. This is shown in the cord diagram in Figure B.3. It provides object properties common to all process activites.

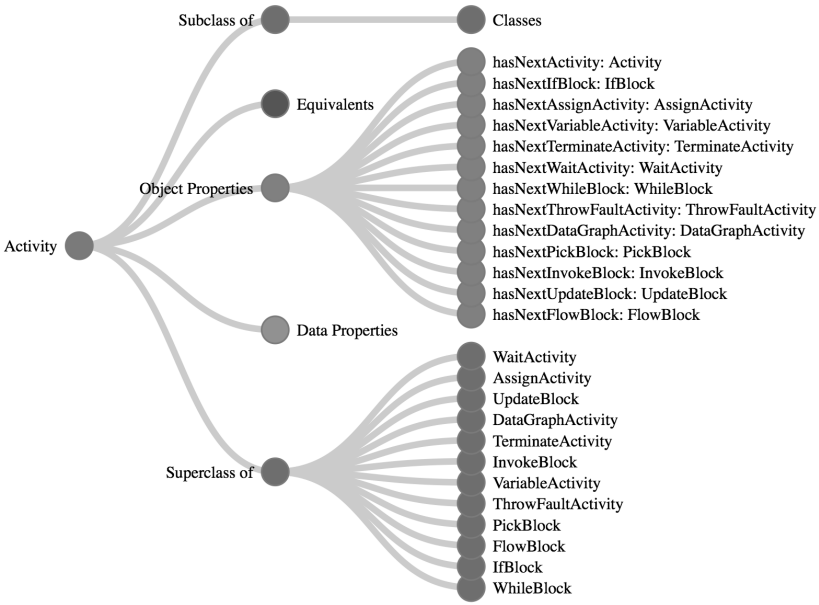


Figure B.4: Class Activity

As shown in Figure B.4 there are twelve object properties. Eleven of these properties are sub-properties of the property "hasNextActivity". This property is functional so there is only one activity that follows an activity.

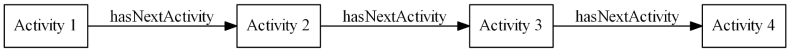


Figure B.5: Activity Process Chain

By linking the activites in this manner, as shown in Figure B.5 the activites are excuted one activity at a time until an activity does not have a next activity.

B.1.2 ActivityBlock Class

The ActivityBlock class contains one or more object properties with a range of individuals of class Activity. These individuals are then linked together as one or more sub-processes.

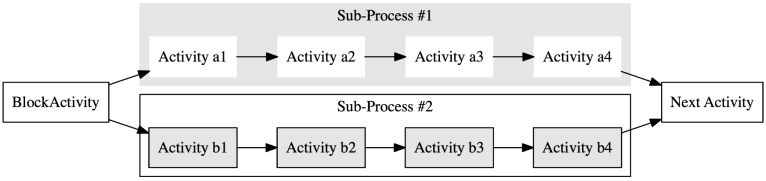


Figure B.6: Sub-Process Chains

Figure B.6 is an example of a ActivityBlock having two sub-processes. How the sub-processes are executed is determined by the class of the ActivityBlock.

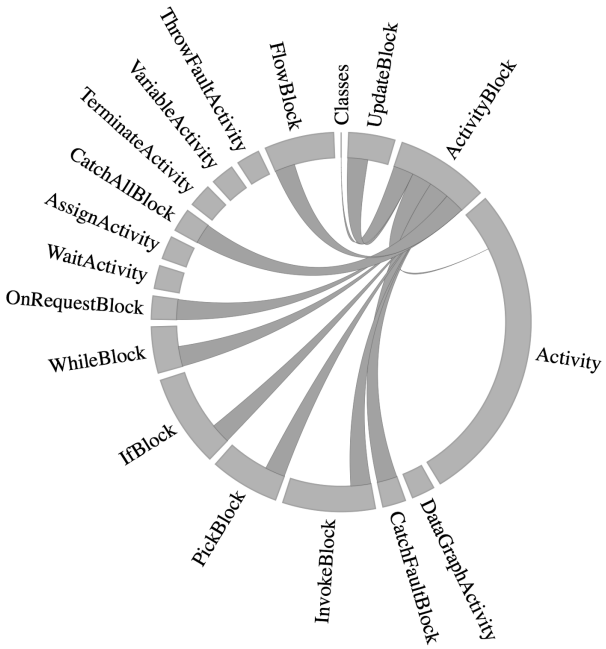


Figure B.7: Cord Diagram of ActivityBlock Class

Figure B.7 shows the classes having object properties.

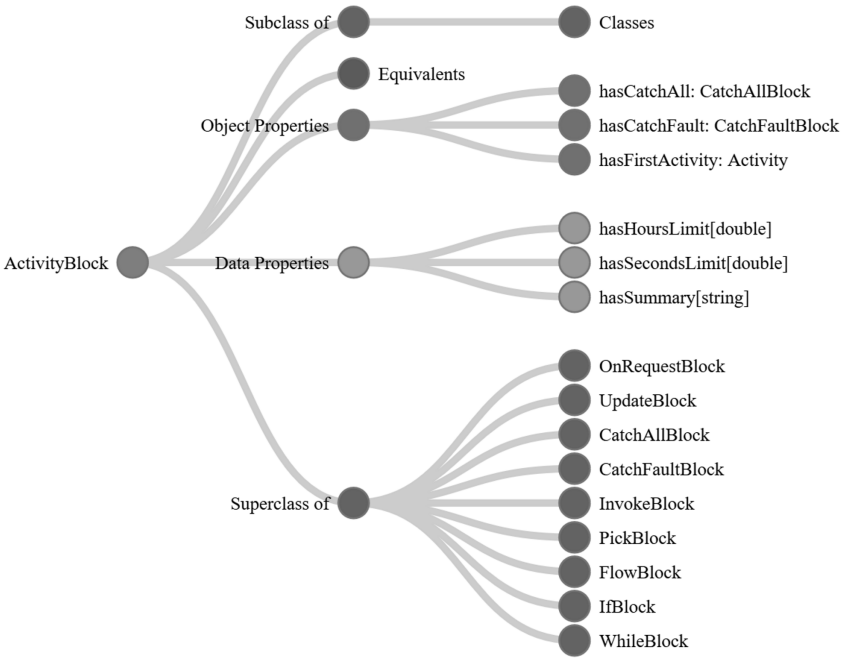


Figure B.8: ActivityBlock Class Properties

Figure B.8 shows the object properties and data properties of the ActivityBlock class.

Object Properties

- **hasCatchAll:** Links a CatchAllBlock
- **hasCatchFault:** Links a CatchFaultBlock
- **hasFirstActivity:** Links an Activity

Data Properties

- **hasHoursLimit:** Limits the time in hours that a BlockActivity may execute

- **hasSecondsLimit:** Limits the time in seconds that a BlockActivity may execute
- **hasSummary:** A general description of the BlockActivity

B.2 Process Classes

B.2.1 OnRequestBlock

A OnRequestBlock is the service entry to a process.

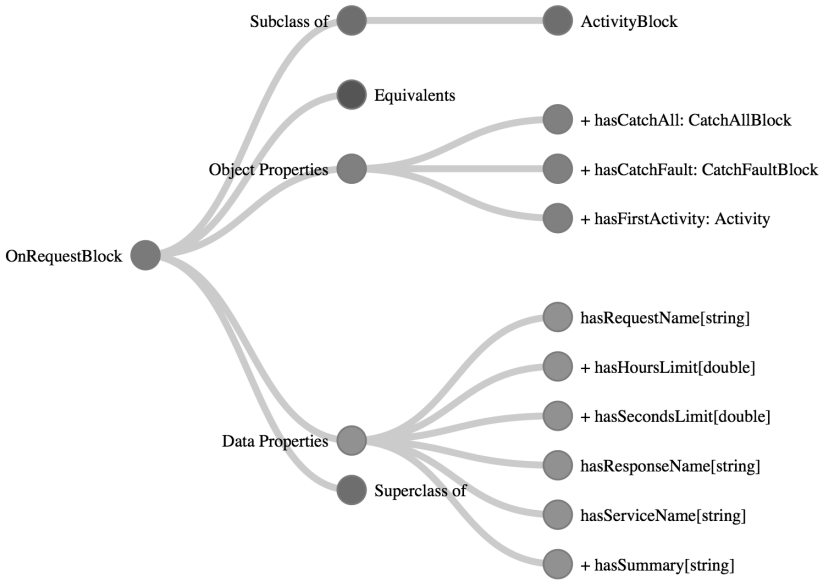


Figure B.9: OnRequestBlock Properties

Figure B.9 shows the object properties and data properties. All of the object properties are inherited from the BlockActivity class and

many of the data propertes are inherited from the BlockActivity class. The data properties with a "+" sign are inherited.

The following are the unique data properties of the OnRequest-Block class:

Data Properties

- **hasServiceName:** This is name of the service.
- **hasRequestName:** The name used by the process for the request datagraph.
- **hasResponseName:** The name used by the process for the response datagraph.

External service references are made to the name of the service. The request datagraph provides information that may be referenced using the request datagraph name. The results of the process are returned in the response datagraph.

B.2.2 CatchAllBlock

Activities to catch and process all exceptions. Only a BlockActivity can catch all faults occuring in the BlockActivity's sub-processes.

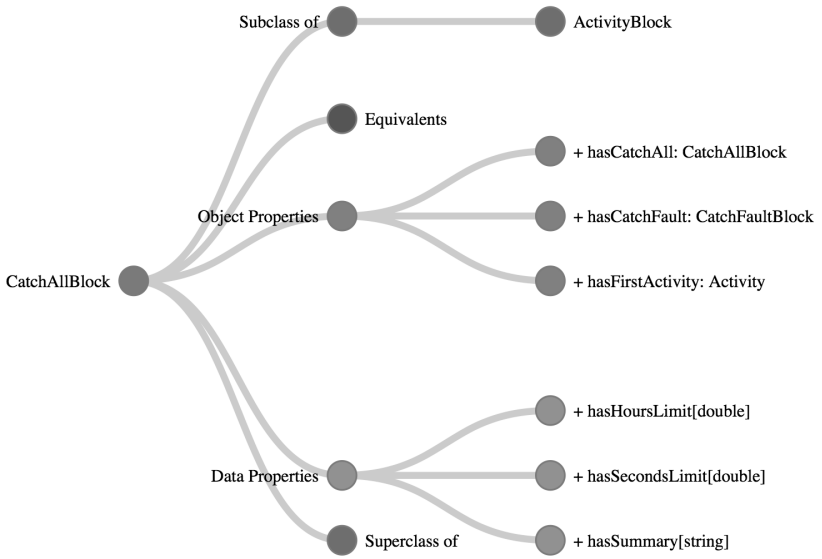


Figure B.10: CatchAllBlock Properties

Figure B.10 shows that all the CatchAllBlock properties are inherited from the BlockActivity class.

B.2.3 CatchFaultBlock

Activities to catch and process specific exceptions. As shown in

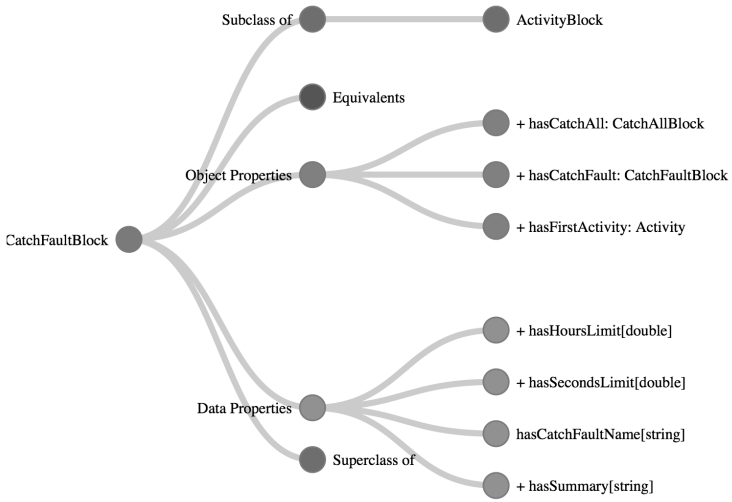


Figure B.11: CatchAll

Figure B.11 the data property "hasCatchFaultName" provides the name of the specific exception. This property is also the only property not inherited from the BlockActivity class.

B.2.4 InvokeBlock

Activities to prepare for and to make a reference to an external service. Figure B.13 shows the properties of the InvokeBlock class. The following describes the properties extending the BlockActivity class.

Object Properties

- **hasFirstPrepareActivity:** The prepare activity is executed first to prepare the request datagraph.
- **hasFirstReceiveActivity:** The activity executed after the receive datagraph is returned from the invoked service.

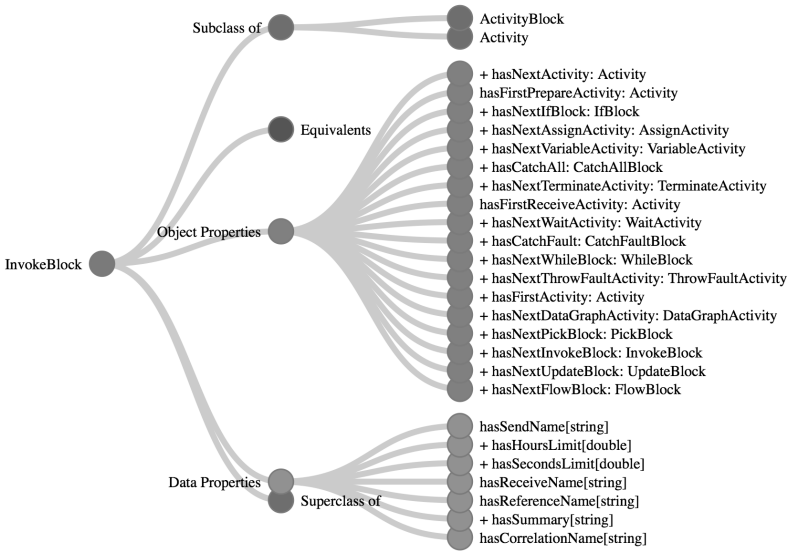


Figure B.12: InvokeBlock Properties

Data Properties

- **hasSendName:** Name of send datagraph.
- **hasReceiveName:** Name of the receive datagraph.
- **hasCorrelationName:** Name of the correlation datagraph.
- **hasReferenceName:** Name of the service reference.

When the InvokeBlock is executed the following steps occur:

1. The send, receive, and correlation datagraphs are created from the metadata provided from the reference.
2. The prepare process is executed.
3. The reference is initated and the invoke waits for a response.
4. Once the response is received, the receive activity is executed.

B.2.5 UpdateBlock

Activities to access, update, and store individuals into an ontology domain. Figure B.13 shows the data and object properties.

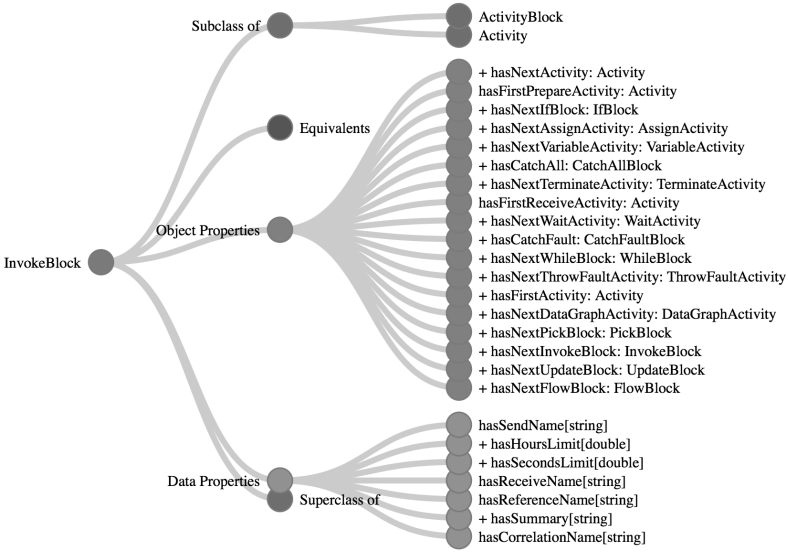


Figure B.13: InvokeBlock Properties

The following are the unique data properties:

Data Properties

- **hasUpdateQuery:** The query used to create the datagraph.
- **hasUpdateDatagraphName:** The name of the datagraph.

The process steps of the update activity:

1. The datagraph is created using the query.
2. The first activity is executed.

3. The datagraph change list is applied to the domain individuals.

4. If the update fails due to parallel changes to the individuals, the process returns to step 1.

B.2.6 WhileBlock

Repeats activities until a specific condition is met.

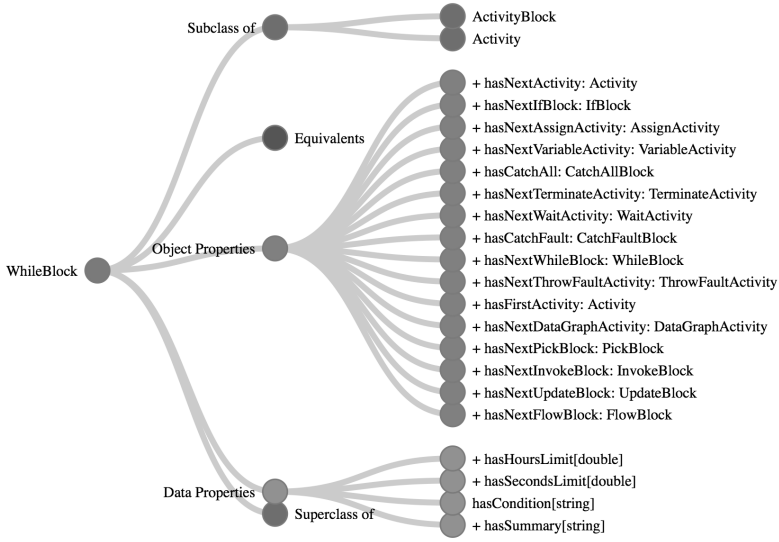


Figure B.14: WhileBlock Properties

The WhileBlock class has one property that defines the condition. This is show in Figure B.14 as the data property of "hasCondition". The format of the condition string is defined in Appendix C.

The condition is tested when the WhileBlock is executed. If the condition is true, the sub-process is executed and the condition is tested again. Execution of the sub-process will continue until the condition is false.

B.2.7 PickBlock

PickBlock starts multiple sub-processes simultaneously and completes when the first sub-process completes.

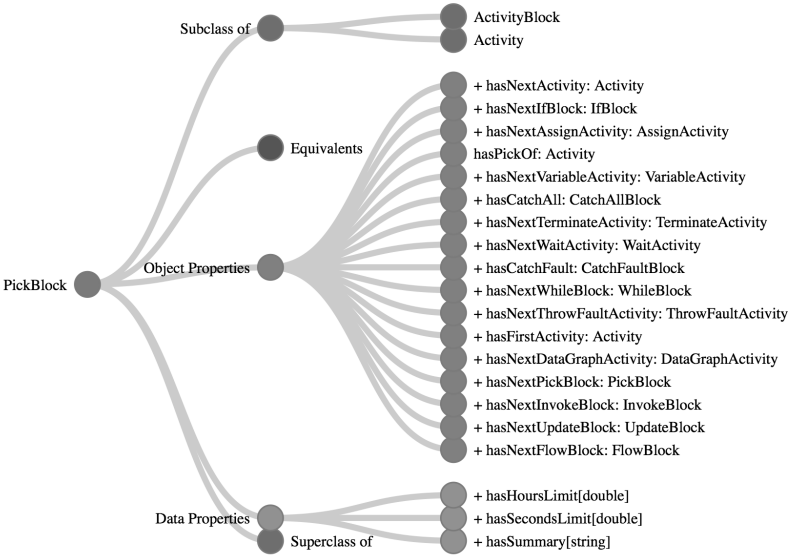


Figure B.15: PickBlock Properties

The object property "hasPickOf" as shown in Figure B.15 has a range of one or more sub-processes.

When the PickBlock is executed, the sub-processes are executed. The PickBlock ends when one of the sub-processes completes.

Sub-processes are not executed in parallel. Sub-process are queued for execution. A sub-process is removed from the queue and executed until it enters a wait state. When the sub-process enters a wait state it is placed onto the wait queue.

When a sub-process is placed on the wait queue, execution continues with the next sub-process in the execution queue. If all sub-

processes are in the wait queue, the PickBlock process waits until one of the sub-processes is ready for execution.

B.2.8 FlowBlock

FlowBlock starts multiple activities to execute in parallel.

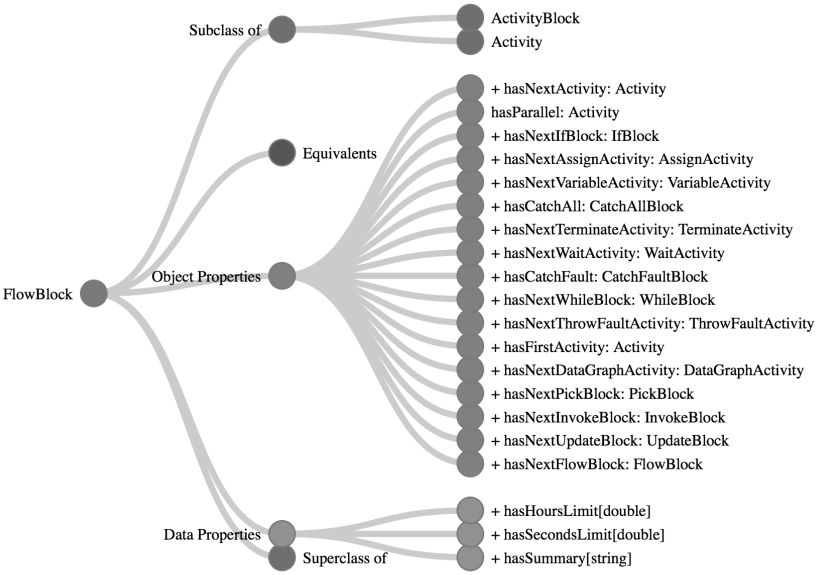


Figure B.16: FlowBlock Properties

The property "hasParallel" can have one or more sub-processes as shown in Figure B.16.

All of the sub-processes must complete before the FlowBlock is complete.

B.2.9 IfBlock

Selects the direction of process flow based upon a condition.

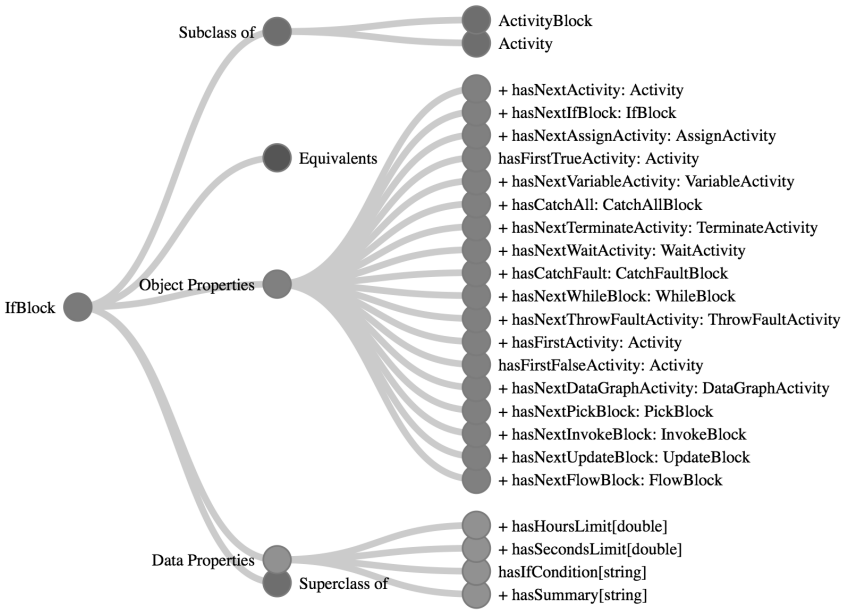


Figure B.17: IfBlock Properties

As shown in Figure B.17 the data property "hasIfCondition" is a condition string as defined in Appendix C. The object property "hasFirstTrueActivity" is a reference to a sub-process to execute if the condition is true. If the condition is false, and the object property "hasFirstFalseActivity" is defined, the false sub-process is executed.

B.2.10 VariableActivity

VariableActivity creates a variable within the scope of the current BlockActivity. Figure B.18 shows the unique properties of the VariableActivity.

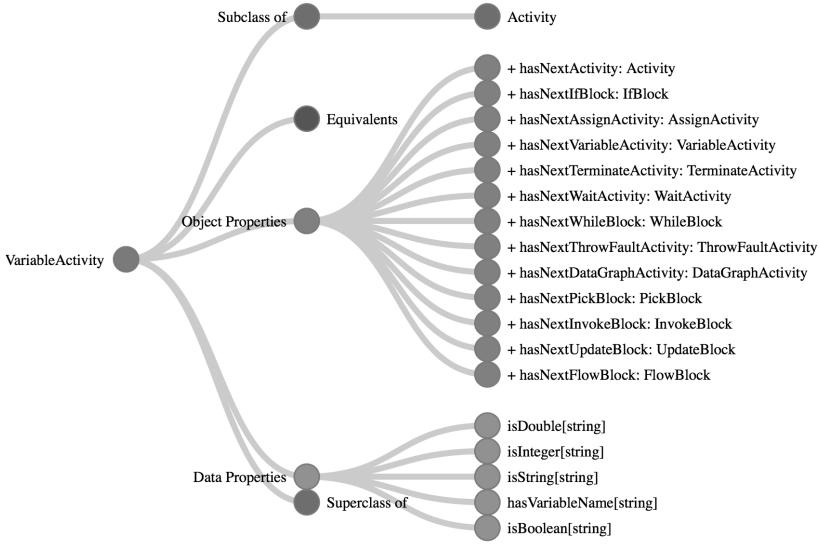


Figure B.18: VariableActivity Properties

Data Properties

- **hasVariableName:** The name used to reference the variable.
- **isBoolean:** Defines the variable as type boolean and sets the initial value as given in the string.
- **isString:** Defines the variable as type string and sets the initial value as given in the string.
- **isInteger:** Defines the variable as type integer and sets the initial value as given in the string.
- **isDouble:** Defines the variable as type double and sets the initial value as given in the string.

The string to set the value of the variable is defined in Appendix C as "*source*".

A sub-process can reference any variable by name that is defined in any of it's super BlockActivity processes.

B.2.11 DataGraphActivity

DataGraphActivity creates a datagraph. The object properties shown

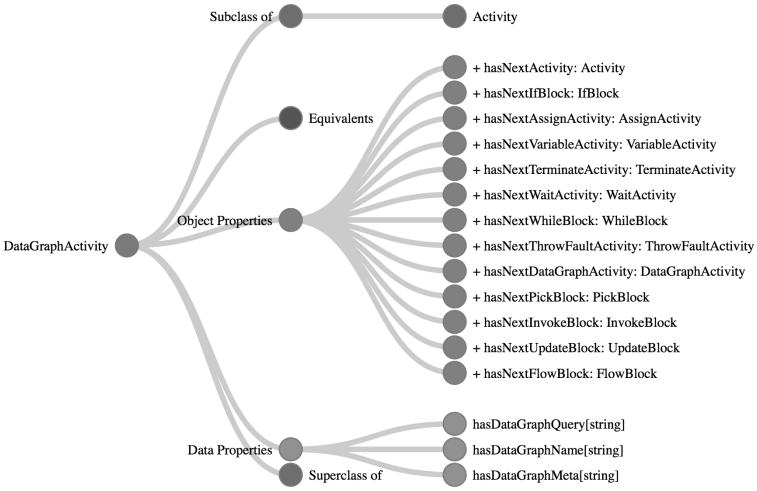


Figure B.19: DataGraphActivity Properties

in Figure B.19 are all inherited from the Activity class. The DataGraphActivity data properties are unique.

Data Properties

- **hasDataGraphName:** The name used to reference the datagraph.
- **hasDataGraphQuery:** Defines the query to use to define the metadata of the datagraph and insert the individuals of the defined sets from the repository.

- **hasDataGraphMeta:** Defines the query to use to define only the metadata of the datagraph.

A DataGraphActivity name is used for refence by other activities. The query and the meta properties are mutually exclusive. Providing meta will create the datagraph without any individuals. With query, the datagraph is created and populated with individuals from the repository.

B.2.12 AssignActivity

AssignActivity provides for assigning values to variables and datagraphs.

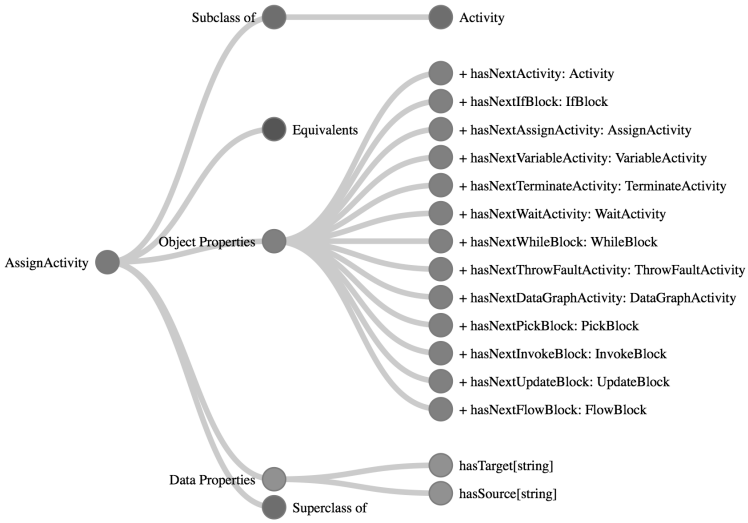


Figure B.20: Assign Activity Properties

The data property "hasTarget" defines what value is to be assigned and the data property "hasSource" defines what value to assign as shown in Figure B.20.

The string values for target and source are both described in Appendix C.

B.2.13 WaitActivity

The WaitActivity causes the process to wait the specified amount of time. The data properties of "hasHoursWait" and "hasSecondsWait"

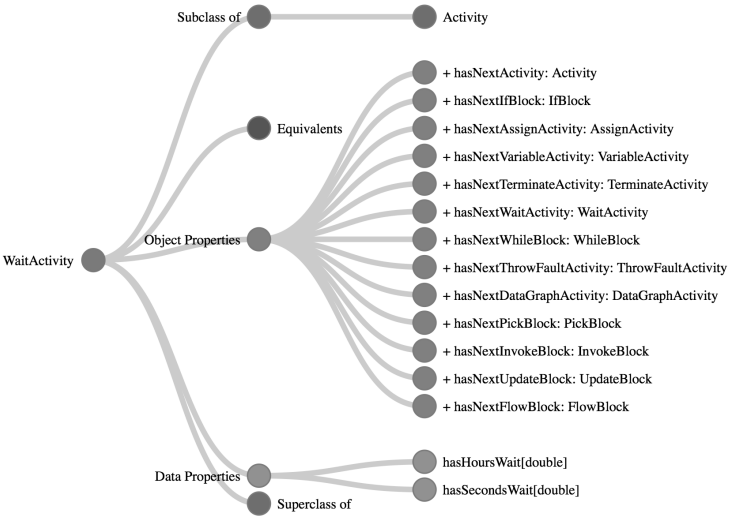


Figure B.21: Wait Activity Properties

provide the time to wait before the process is restarted. Both properties have a value of type double.

B.2.14 ThrowFaultActivity

The ThrowFaultActivity throws a specific fault.

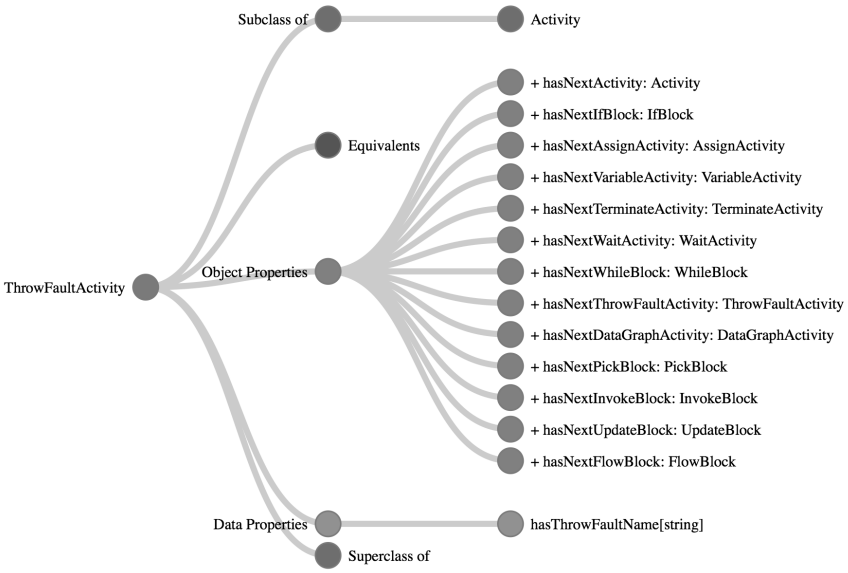


Figure B.22: ThrowFaultActivity Properties

The unique data property of "hasThrowFaultName" as shown in Figure B.22 gives the name of the fault to be thrown. The fault may be caught by any of the super BlockActivity processes by fault name or by a catch all faults.

Throwing a fault terminates the current process.

B.2.15 Terminate

Terminate ends the current process.

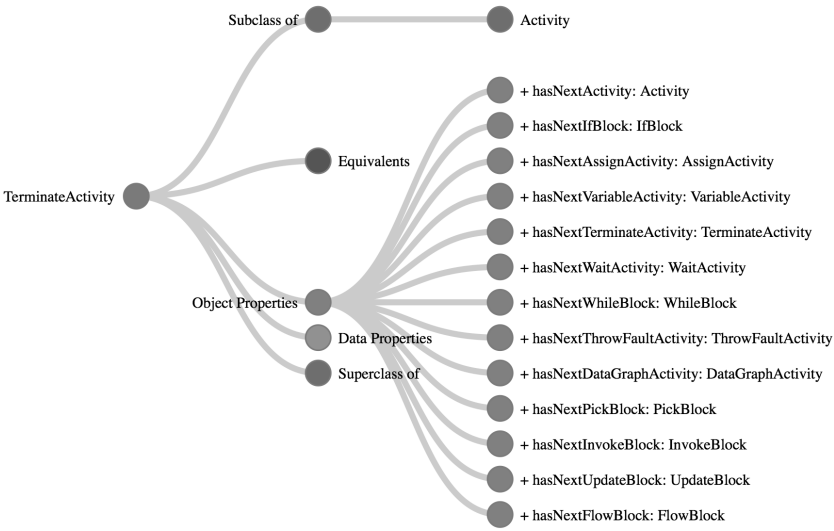


Figure B.23: TerminateActivity Properties

Figure B.23 shows that the TerminateActivity has no unique properties.

Appendix C

BPEL Reference Syntax

The following lists the syntax for references in the Assign, Variable, If, and While processes activities. Names in italics are references and brackets are used to show optional items. Alternatives are listed for each reference.

target

- **variable:** Name of a variable
- **datagraph[*index*]*target-path*:** Datagraph by dataobject at index in list of root objects and target-path
- **datagraph.new(root class)*target-path*:** Create a new root object and the path

source

- **literal:** A specific value
- **variable:** Name of a variable
- **datagraph[*index*]*source-path*:** A datagraph dataobject at index in the list of root objects and the path

- **datagraph.sourceID:** The identity of the creator of the datagraph

target-path

- ***data-attribute*:** Data property
- ***target-object-path data-attribute*:** Path to a data property

source-path

- ***data-attribute*:** Data property value
- ***object-path data-attribute*:** Path to a data property value

target-object-path

- ***target-object-property*{*target-object property*}:** Path to an object property value

data-attribute

- **@data property name:** Data property of a data object

object-path

- ***/object property name*{[*index*]}{*object-path*}:** Path to a data object

target-object-property

- **/object property name:** Object property of an object
- **/object property name.new:** Create a new object as defined by property

literal

- ***integer*:** Literal integer value
- ***string*:** Character string in quotes
- ***double*:** Double value
- ***boolean*:** Value of true or false

index

- ***integer*:** Literal integer value
- ***integer value*:** Integer value from an object data attribute
- **integer variable:** Value of an integer variable

condition

- **true:** Literal value for true
- **false:** Literal value for false
- **(*condition*):** Parenthesis to offset condition
- ***condition* & *condition*:** And two conditions
- ***condition* | *condition*:** Or two conditions
- ***value relation value*:** Test relation between two values
- **!*condition*:** Not condition

relation

- **=** : Equal
- **!=** : Not equal
- **>** : Greater than
- **<** : Less than
- **>=** : Greater than or equal
- **<=** : Less than or equal

About the Author

Tom Tinsley is a visionary with decades of experience in software development as a developer, as a manager, and as an Enterprise Architect. He is also the author of the books, *Enterprise Architects: Masters of the Unseen City* and *Deadlines and Duct Tape*. After years of managing Information Technology as a Vice President in Banking and as a Director in State Government, Tom turned to strategy and mentoring as an Enterprise Architect. He has led Enterprise Architecture in the retail industry and the media industry to achieve a common vision for the use of computing technology to meet business goals.

His personal passion has been to bridge the gap between business management and Information Technology. Tom has published many internal documents and given multiple speeches and presentations to further the awareness of new opportunities and the application of best practices. He recognizes the need for business management to play a greater role in the actions of Information Technology. He sees this as a win-win where business management can focus on servicing their customers and Information Technology can focus on providing reliable, high-performance, automated services.