

**Designing Personal Assistant Software
for Task Management
using Semantic Web Technologies and Knowledge Databases**

Purushotham Botla

Working Paper CISL# 2013-11

May 2013

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E62-422
Massachusetts Institute of Technology
Cambridge, MA 02142

Designing Personal Assistant Software for Task Management using Semantic Web Technologies and Knowledge Databases

By
Purushotham Botla
B.E., Electronics
Mumbai University, 1995

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT
AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUNE 2013

©2013 Purushotham Botla.
All rights reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature of Author: _____
System Design and Management
May 20, 2013

Certified by: _____
Stuart Madnick
John Norris Maguire Professor of Information Technologies, MIT Sloan School of Management & Professor of Engineering Systems, MIT School of Engineering
Thesis Supervisor

Accepted by: _____
Patrick Hale
Director, System Design and Management Program

This page left intentionally blank

Designing Personal Assistant Software for Task Management using Semantic Web Technologies and Knowledge Databases

By
Purushotham Botla

Submitted to the System Design and Management Program on May 20, 2013 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Engineering and Management

Abstract

Adoption of social network sites and use of smart phones with number of sensors in them has digitized user's activities in real-time. Smart phone applications such as calendar, email, and notes contain lot of user information and provide a view into user's activities, while sensors such as GPS sensor can be used to passively find information about the user. In addition to this user and device data, these devices have access to the Internet that can be leveraged to build powerful applications.

Personal assistant software (smart agent) can be used as an interface to the digital world to make the consumption of this information timely and efficient for the user's specific tasks. Goal of the thesis is to design personal assistant software that understands the semantics of the task, is able to decompose the task into multiple tasks within the context of the user and plan these tasks for the user. It will be designed using semantic web technologies and knowledge databases to understand the relations between the tasks. Agent will be integrated with online web-services to harvest the data available on-line with the data available on the device and help the user to manage his or her tasks.

Two use cases are covered in this thesis document to explore automation capabilities and planning capabilities of the agent. Design of the agent using the two use cases helped in the design of sub-modules within the agent system, and also highlighted the requirements on external data and knowledge sources.

Acknowledgements

I would like to thank my advisor Prof. Madnick for providing me the opportunity to work on this thesis and for providing valuable advice and guidance along the way. I am also thankful to Allen Moulton for reviewing my work and providing valuable feedback and direction.

Many classes I took at MIT helped shape my understanding and design of the personal assistant software covered in this thesis. Some of these classes include *Common Sense reasoning in applications* taught by Henry Lieberman, *Linked Data ventures* class taught by Sir Tim Berners Lee and *Evolution towards Web 3.0 and Emergence of Management 3.0* taught by Prof. Madnick. These classes reinforced my interest in semantic web and personal assistants.

I have enjoyed my time at MIT, and had the privilege of working with some of the brightest minds in the world. Flexibility offered by SDM program helped me venture into learning interesting classes at MIT, to expand my horizons and to gain a better understanding of management as well as technology. Thanks to SDM program director Patrick Hale for structuring the SDM program to cater to the needs of students from diverse backgrounds and still be able to tailor it to individual needs. My SDM cohort as well as students and faculty across MIT have made this an unforgettable and cherished experience.

I would like to thank my wife Shivaranjani for encouraging me to venture into SDM program at MIT and providing strong support through out my SDM program. Without her support this would not have been possible. I will always be indebted to her. Lastly, I am thankful to my two sons Kunal and Vatsal for their co-operation and understanding during this time.

TABLE Of CONTENTS

1 Introduction	1
1.1 Motivation for this Thesis.....	1
1.1.1 Information overload on the web.....	1
1.1.2 Silos of information.....	1
1.1.3 Using multiple applications to get one user task done.....	2
1.1.4 Need for a cleaner and efficient interface to the digital world, that wraps the complexity of interaction with digital world from the user.....	2
1.2 Enabling Technologies.....	2
1.3 Objectives.....	3
1.4 Organization of the thesis.....	4
1.4.1 Chapter 1: Introduction.....	4
1.4.2 Chapter 2: Current Market trends and solutions.....	4
1.4.3 Chapter 3: Research Methodologies.....	4
1.4.4 Chapter 4: Overview of semantic web technologies.....	4
1.4.5 Chapter 5: Designing the smart agent.....	5
1.4.6 Chapter 6: Assumptions, Constraints and Limitations of the system and ways to address them.....	5
1.4.7 Chapter 7: Conclusion and Future work.....	6
2 Personal Assistant Software in the Market.....	7
2.1 Goals of Personal Assistant Software	7
2.2 Different types of Personal Assistant Software	7
2.2.1 Voice recognition as input entry medium.....	7
2.2.2 Voice recognition based task automation or information retrieval.....	7
2.2.3 Planning.....	7
2.3 Technology Enablers	8
2.3.1 Smart phones – CPU, memory, storage and gesture interface.....	8
2.3.2 Voice recognition.....	8
2.3.3 Network connectivity	8
2.3.4 Bandwidth.....	8
2.3.5 Web services.....	8
2.3.6 Sharing of data thru web services and Linked Data.....	8
2.3.7 Personal Information integrated with information on the web	9
2.3.8 Task and Domain Models for specialized tasks	9
2.3.9 Cloud computing.....	9
2.4 CALO.....	9
2.4.1 Use cases Supported by CALO.....	10
2.5 SIRI from Apple.....	10
2.5.1 User contexts supported by SIRI.....	11
2.5.1.1 Location	11
2.5.1.2 Temporal Context.....	11
2.5.1.3 Social Context.....	11
2.5.1.4 Context between other tasks	11
2.5.2 Use cases supported by SIRI.....	12
2.5.2.1 Invoking iOS applications based on user request	12
2.5.2.2 Calling external services to serve user requests.....	12
2.5.3 SIRI System Modules.....	13

2.5.3.1	Interaction between models within SIRI	16
2.5.3.2	Salient Features and limitations of SIRI	17
2.6	ReQall.....	18
2.7	Google Glass.....	20
3	Research and Design methodology	21
3.1	Research on Sources of Data	21
3.1.1	<i>Semantic Web Ontologies.....</i>	<i>21</i>
3.1.2	<i>Semantic Web Data Sources</i>	<i>22</i>
3.1.3	<i>Web services</i>	<i>22</i>
3.1.4	<i>Identifying and Defining Scope.....</i>	<i>22</i>
3.2	Reviewing existing literature	23
3.3	Reviewing Existing Solutions	23
4	Overview of Technologies Reviewed and Used	24
4.1	Semantic Web Technologies.....	24
4.1.1	<i>Introduction to Semantic Web.....</i>	<i>24</i>
4.1.2	<i>What is Semantic Web?.....</i>	<i>25</i>
4.1.3	<i>Who is contributing to Semantic Web?.....</i>	<i>26</i>
4.2	What is covered in this chapter?.....	26
4.3	RDF	27
4.3.1	<i>RDF Graph Model</i>	<i>27</i>
4.3.2	<i>RDF Examples</i>	<i>28</i>
4.3.3	<i>Features of RDF.....</i>	<i>30</i>
4.4	RDFS	30
4.5	SPARQL	32
4.5.1	<i>Anatomy of a SPARQL Query.....</i>	<i>32</i>
4.5.2	<i>SPARQL Syntax by example</i>	<i>33</i>
4.5.3	<i>SPARQL Query against BestBuy RDF product dataset.....</i>	<i>34</i>
4.5.4	<i>Types of SPARQL Queries.....</i>	<i>35</i>
4.5.5	<i>Limitations in current SPARQL 1.1 version, covered in new SPARQL 2</i>	<i>36</i>
4.6	Ontologies	37
4.6.1	<i>Web Ontology Language (OWL)</i>	<i>37</i>
4.6.2	<i>OWL 2 Features</i>	<i>37</i>
4.6.3	<i>Modeling Knowledge</i>	<i>38</i>
4.6.4	<i>OWL by example</i>	<i>39</i>
4.7	Evaluating Ontologies for Smart Agent.....	42
4.7.1	GoodRelations.....	42
4.7.1.1	<i>How does it help spread of Linked Data?</i>	<i>45</i>
4.7.1.2	<i>Example RDF/XML for a CD 'Feats Don't fail me now' on bestbuy.com.....</i>	<i>46</i>
4.7.1.3	<i>Leveraging GoodRelations ontology.....</i>	<i>47</i>
4.7.2	DBpedia	48
4.7.3	Wordnet	49
4.8	Knowledge Databases	53
4.8.1	OpenCyc	54
4.8.1.1	<i>Why use logic to represent knowledge?</i>	<i>54</i>
4.8.1.2	<i>What kind of knowledge is encoded in OpenCyc.....</i>	<i>55</i>
4.8.1.3	<i>Reasons for choosing OpenCyc</i>	<i>56</i>
4.8.1.4	<i>Components of Cyc</i>	<i>57</i>
4.9	Online Web Services	59
4.9.1	<i>Yelp: Local Business Database API</i>	<i>59</i>

4.9.2	<i>National Weather Service web service to weather information</i>	60
4.9.3	<i>Google Web services</i>	60
4.10	Mobile Devices	61
5	Designing Smart Agent	65
5.1	Semantic Web as a building block	65
5.2	Designing Smart Agent	66
5.2.1	<i>Requirement Analysis</i>	66
5.3	General Architectural and Design Principles	68
5.3.1	<i>Modularity</i>	68
5.3.2	<i>Extensibility</i>	68
5.3.3	<i>Complexity</i>	69
5.3.4	<i>Falling back to the user in case of ambiguity or unknowns</i>	69
5.3.5	<i>Adopting Open Standards and Data Reuse</i>	69
5.3.6	<i>Integrating with external data sources and Services</i>	69
5.3.7	<i>Early Detection</i>	69
5.4	Abstract Models in the agent	70
5.4.1	Task templates	70
5.4.2	Web Service Integration	71
5.4.3	Task Model	73
5.5	Modules within the system	75
5.5.1	<i>Five Layers for connecting to external systems</i>	76
5.5.2	<i>Agent modules orchestrating interactions with five integration layers</i>	79
5.6	Use cases	80
5.6.1	<i>Use case 1: Agent is requested to get weather report for my parent's hometown</i>	80
5.7	Sequence Diagram & Data Flow during execution of task	82
5.7.1	<i>Task Input</i>	83
5.7.2	<i>Task Execution</i>	84
5.7.3	<i>Use case 2: Planning the task – Drive to Airport</i>	85
5.8	Sequence Diagrams for use case 2: Drive to airport	87
5.8.1	<i>Parsing the input</i>	87
5.8.2	<i>Identify the task domain model</i>	88
5.8.3	<i>Create a new task object</i>	89
5.8.4	<i>Collecting additional information from the user</i>	89
5.8.5	<i>Get sub-tasks for the activity from knowledge base</i>	90
5.8.6	<i>Planning the activity</i>	91
5.8.7	<i>Monitoring the plan</i>	91
5.8.8	<i>Plan for the activities at the airport</i>	91
5.8.9	<i>Summary of this use case</i>	91
6	Assumptions, Constraints and Limitations of the system and ways to address them	93
6.1	Personal assistant can be built for specific domains, but integrating this with the knowledge of the domain is the challenge.	93
6.2	Voice interface for the agent	93
6.3	User interactions with the agent	94
6.4	Stronger NLP to capture user input unambiguously	94
6.5	Performance of semantic web endpoints	94
6.6	Data accuracy and completeness	94
6.7	Proliferation of web services	95

6.8	Task Models.....	95
6.9	Planning	95
6.10	Sharing task decomposition with other users.....	96
6.11	Ability to add new facts and task decomposition to database.....	96
6.12	Learning from user input and actions	96
6.13	Personal information on the mobile phone.....	96
6.14	Integrating with social networks, and delegating tasks to other users in the network	96
7	Conclusion and Future work.....	97
7.1	Personal Assistants are the future	97
7.2	Supporting Tasks	97
7.3	Evolution of semantic web ecosystem	98
7.4	Building the agent.....	98
7.5	Knowledge Databases	98
7.6	Planning Algorithms	99
7.7	Reasoning.....	99
8	Bibliography.....	100

TABLE of FIGURES

Figure 2-1: SIRI System Modules	13
Figure 2-2: SIRI input and output interfaces	15
Figure 2-3: Interactions between models within SIRI	16
Figure 4-1: WWW - hyperlinked web pages	24
Figure 4-2: Linked Open Data Cloud Diagram	26
Figure 4-3: RDF Graph Model	27
Figure 4-4: RDF Example 1	28
Figure 4-5: RDF Example 2	29
Figure 4-6: Classes and Resources as Sets and Elements	31
Figure 4-7: GoodRelations UML Diagram	44
Figure 4-8: GoodRelations example RDF data in XML form	46
Figure 4-9: Wordnet hyponym, antonym and meronym relationship example	50
Figure 4-10: Wordnet synsets and antonyms example	51
Figure 4-11: RDF representation of words in Wordnet	52
Figure 4-12: ABox and TBox Systems	53
Figure 4-13: OpenCyc Knowledge database	55
Figure 4-14: Microtheory in Cyc	58
Figure 4-15: Android System Architecture	63
Figure 5-1: Integrating agent with the web of linked data	65
Figure 5-2: Sample Tasks using Task Template	70
Figure 5-3: Web Services Manager	72
Figure 5-4: Web Service Mapping Example	73
Figure 5-5: Task Model	74
Figure 5-6: Modules within Smart Agent system	75
Figure 5-7: Input Processing Layer	76
Figure 5-8: Web Services Manager Layer	76
Figure 5-9: Knowledge Search Layer	77
Figure 5-10: Semantic Web Interface Layer	77
Figure 5-11: Device Integration Layer	78
Figure 5-12: Agent modules interacting with 5 integration layers	79
Figure 5-13: Sample report for weather request	82
Figure 5-14: Sequence diagram of input system for use case-1	83
Figure 5-15: Sequence diagram for task execution in usecase-1	84
Figure 5-16: Task Entry screen for 'Driving to Airport'	85
Figure 5-17: Task Entry screen to enter time and choose airport	86
Figure 5-18: Agent's plan for the task - 'Driving to Airport'	86
Figure 5-19: Sequence diagram for input system in usecase-2	87

1 Introduction

1.1 Motivation for this Thesis

1.1.1 Information overload on the web

Huge amount of information is being generated by online websites and is primarily consumed by users browsing the website. Websites have used HTML and XML based technologies to render the content on the user's screen and let the user make decisions based on his interpretation of that content.

In order to be discovered by the search engines, web sites include some meta-data about the content, enabling discovery of the web site's pages when a user makes a relevant query. This model puts user at the center to identify tools such as search engines in order to get to the relevant content for his need, filter from the list of available content and then read the content - making the task of consuming this information increasingly difficult and leading to scalability limitations on the amount of content that can be consumed by users, causing information overload.

Availability of this information coupled with its overload on users has created a need for applications that can act as a conduit to interface users to the digital world by using the online information for the benefit of its users, with a minimal intervention from the users.

1.1.2 Silos of information

Adoption of proprietary structure for the data by enterprises has led to proliferation and creation of silos of information that makes interpretation of this data difficult by other applications interested in that data. Even though large amounts of data and services are available, need for interpreting proprietary formats of the data makes it difficult to easily consume this data and so limits the spread of this information to interested parties who can provide richer value added services.

In contrary, adoption of standard specifications to represent this data and publishing the content as well as the meta data used to generate the content can enable creation of applications that can interpret this data on behalf of the users and make the consumption of the content easier as well as timely.

1.1.3 Using multiple applications to get one user task done

Proliferation of mobile devices and mobile apps running on these devices has led to specialized applications with a razor focus on helping user accomplish a very specific task, using some of the contextual information available from sensors of mobile device, but with almost no interactions with other applications that user may be using to accomplish other tasks.

This leaves the burden of breaking the tasks for a user's intention to complete a task solely on the user. In a way, user needs to manually manage the workflow of the system as well as data transformation and data flow by using multiple sets of applications to complete one task.

For example, a user trying to make a travel plans need to check for airport codes for nearby airports and then check travel sites for tickets between combinations of airports to reach the destination.

Another example is - instead of having a separate applications for every day information such as train timings, weather conditions, sports scores, news etc., it would be greatly useful if an application would take over this task and present the information to its user when requested or for any urgent scenarios as alerts to the user.

1.1.4 Need for a cleaner and efficient interface to the digital world, that wraps the complexity of interaction with digital world from the user

Complexity of getting a task done has been offloaded to the user as described in previous section 1.1.3. This helps the software companies to focus on a specialized problem domain while delegating overall management and engineering of the workflow to the user.

One of the biggest motivating factors in undertaking this thesis research has been to identify data sources and structures of data that can enable uniform definition of data and also extend this definition based on any specific needs of an application.

Some of the enabling technologies discussed in next section can be used to change this paradigm to a more user centered one with focus on achieving tasks for the user at a more holistic level.

1.2 Enabling Technologies

Semantic Web or Web 3.0 refers to availability of data in machine-readable form, along with self-contained information on the type of data that can be used to intelligently inspect and process this data. Adoption of linked data technologies such as RDF is enabling web to link different aspects of data together and moving towards Tim Berners-Lee's vision of single linked web of data. In addition, this data

is also structured in a homogeneous form that can be used to interpret the data and build interesting as well as intelligent applications that can discover and process this data.

Availability of semantic data and technologies to process this data provides an opportunity to build applications that can understand the tasks they are executing and plan these tasks by understanding the context around these tasks for the user.

In last few years, wide adoption of social networking sites such as Facebook, Twitter and LinkedIn has led to availability of vast amounts of user data that carries almost in real time, user activity information as well as their profile information. Data which was previously in non-digital form became digital and is now available to other applications interested in this data, with user's permission.

Devices such as smart phones, tablets and sensors in home/office appliances have enabled access to user related data that was not available electronically earlier. With ever more increasing reliance on these devices we are moving towards an Internet of things where devices will be able to query the web and applications can be written to leverage this information to provide contextual and intelligent solutions to the user.

There has been an expansion in availability of data in machine readable form which forms basis of this thesis to build intelligent applications such as smart personal assistant software that can leverage this data along with knowledge databases that carry common sense knowledge of the world and use this understanding to solve interesting problems in an intelligent way.

1.3 Objectives

Main objective of this thesis is to show feasibility of building a personal assistant software (a smart agent) using semantic data sources available on the web, user generated content, data from the sensors of user's mobile devices and providing knowledge from knowledge databases as well as from inference technologies of web 3.0.

To design a smart agent that has contextual information about the user and helps in managing and planning tasks, using semantic web technologies and open data available on the Internet. Contextual information about the user can be location, current time, calendar appointments, relation between tasks, decomposition of tasks, past history of tasks, user interests, likes etc. Agent can use data gathered about the user as well as environment data to better understand what each of the tasks mean and decompose the tasks based on sequence of steps stored in its knowledge base and then plan individual tasks.

Planning part of the agent will strive to optimize resources and try to improve productivity of the user. It can be used as a time management application as well as

a task management application. By combining, related tasks together that can be completed at the same time and around the same location, agent will optimize the user's resources to complete these tasks.

A feedback loop from the user will help the agent to make decisions when there are multiple paths and agent does not have sufficient information to make those decisions.

Assumptions, limitations and constraints in the solution will be highlighted and any additional infrastructure necessary as a complement to the system will be identified.

1.4 Organization of the thesis

Thesis report is organized into six chapters, from introducing the topic to current solutions, review of technologies, research methodologies, design and implementation of the system to summary of the achievements at the end of the thesis report. Each of the chapter summaries is described below.

1.4.1 Chapter 1: Introduction

This chapter provides introduction to the topic of a smart agents or personal assistant software, motivations and drivers for writing this thesis, objectives or goals set out at the beginning of the thesis. It also covers organization of the thesis, with a brief summary about the contexts of each chapter.

1.4.2 Chapter 2: Current Market trends and solutions

This chapter covers personal assistant software applications such as SIRI, ReQall, and Google Glass that help users interactions with the digital world. It covers high-level use cases handled by these software applications.

1.4.3 Chapter 3: Research Methodologies

This chapter covers research methodology to research the topic as well as software methodology used to build and test the agent.

1.4.4 Chapter 4: Overview of semantic web technologies

Chapter 4 of the thesis provides an overview of different technologies that can be leveraged to tap into the semantic and non-semantic data sources on the web, as well as some of the tools that could be leveraged to wrap these technologies to build an agent system. It explores the specifications, ontologies, knowledge databases, and online web services that provide access to data or services pertaining to it.

It begins with a overview of semantic web technologies as well as web technologies and tools that can be used to build the system, covering specifications such as RDF,

RDFS, OWL and SPARQL that are being widely used by the proponents of semantic web to represent, share and query the data.

It also covers some of the relevant ontologies to build the agent to manage user's every day tasks. Each of the ontology reviewed in this section represents a domain of interest and captures the knowledge of domain in standardized W3C form such as RDFS or OWL. Goal of this section is to capture relevant ontologies from domains such as social relationships, product definitions, e-commerce, task definitions, and knowledge databases with a view to leverage them while designing the agent.

It then provides a brief overview of knowledge databases and type of the knowledge that is currently embedded in them. These knowledge databases form the central building block in the design of the system, providing flexibility to the system in terms of capturing the knowledge and exposing this knowledge to the agent in order to make inferences and plans.

1.4.5 Chapter 5: Designing the smart agent

Chapter on designing the agent begins by identifying high-level requirements of the system, dives into more detailed use cases to highlight specific cases of the instances these requirements will be utilized. High-level requirements define the scope of the system pertaining to this thesis and use cases offer a way a path to utilize certain features of the system.

It delves into architectural considerations for building an agent that utilizes online data and services. And then the design chapter goes into defining the modules and each module's features, interfaces and interactions with other modules within the system.

This chapter uses sequence diagrams and data flow diagrams to show how the control and information is passed to implement a use case of 'Driving to Airport'. Sequence diagrams for this use case also highlight decision points within the process such as how the agent uses knowledge database to identify whether the task can be automated or is a manual task. The design identifies logic embedded within the agent that works with the data stored in linked databases and knowledge databases.

It provides low-level details on the implementation of the modules, interactions between the modules. This chapter includes assumptions on availability of data, services, as well as assumptions related to the user during the runtime of the agent.

1.4.6 Chapter 6: Assumptions, Constraints and Limitations of the system and ways to address them

This chapter reviews the assumptions used while building the system, and also looks at the limitations of the system. It provides improvements to the system to

overcome these limitations as well as to relax some of the assumptions and preconditions that are necessary for the current version to work.

1.4.7 Chapter 7: Conclusion and Future work

This chapter discusses how far the thesis assumptions on semantic data as well as knowledge databases proved to be useful in building the agent. It discusses catalysts that would enable further spread of the semantic web and related technologies and how this spread can fuel building such agents in future.

2 Personal Assistant Software in the Market

2.1 Goals of Personal Assistant Software

Goal of a personal assistant software is to act as an interface into the digital world by understanding user requests or commands and then translating into actions or recommendations based on agent's understanding of the world. This understanding of the world is modeled in a knowledge base that contains relationships, connections and rules between various concepts of the world. These agents, at least at present are not expected to replace humans but can be delegated mundane tasks that user would otherwise not be interested in doing or efficiently doing these mundane tasks by processing large amounts of relevant and real-time information on the web.

2.2 Different types of Personal Assistant Software

There have been multiple approaches to building personal assistant software, based on how the user enters the tasks and how the system interprets them.

2.2.1 Voice recognition as input entry medium

In this category of personal assistant software, focus is relieving the user of entering text input and using voice as primary means of user input. Agent then applies voice recognition algorithms to this input and records the input. It may then use this input to call one of the personal information management applications such as task list or calendar to record a new entry. ReQall application covered later in the chapter falls into this category of personal assistant software.

2.2.2 Voice recognition based task automation or information retrieval

In this category of personal assistant software, focus is on capturing the user input thru voice, recognizing the input and then executing the tasks if the agent understands the task. Software takes this input in natural language, and so makes it easier for the user to input what he or she desires to be done. SIRI and Google Glass fall under this category of software.

2.2.3 Planning

In this category of personal assistant software, focus is on understanding the task, sub tasks associated with it and then creating a plan for the user to complete the tasks. SIRI for certain supported tasks such as booking a reservation at a restaurant using web services such as OpenTable and the agent designed as part of thesis belong to this category.

2.3 Technology Enablers

2.3.1 Smart phones – CPU, memory, storage and gesture interface

Smart phones in the market today offer powerful computing environment in terms of CPU, memory, and file storage that can be used to run complex software applications.

Touch screen and video gesture interface being offered by these phones enables a more user friendly interface that can make the interaction with the smart agent easier as well as reduce the friction of using such an interface.

2.3.2 Voice recognition

Voice recognition software enables hands free use of the applications, lets users to query or command the agent thru voice interface. This helps users to have access to the agent while performing other tasks and thus enhances value of the agent itself.

2.3.3 Network connectivity

These smart phones also have ubiquitous connectivity thru WiFi and cell phone network, enabling distributed applications that can leverage other APIs exposed on the web without a need to store them locally.

2.3.4 Bandwidth

Availability of network bandwidth enables development of applications such as voice recognition software to be implemented on server side while expecting large amounts of voice input to be passed over the network in real time in order to convert voice into text.

2.3.5 Web services

Business as well as data services are being exposed on the web as web services that are implemented using standardized web protocols such as HTTP and use string based data interchange formats such as XML, JSON etc.

2.3.6 Sharing of data thru web services and Linked Data

Proliferation of web services is helping sharing of data across different applications across different networks. Web services can offer data from different data sources to provide an integrated standardized view into this data for all their consumers. This also helps delegate some of the data and information integration as well as business functionality to web services layer, thus simplifying implementation of software applications consuming these services.

Another source of data is linked data, where organizations are exposing the data they collect in semantic web form with specific ontologies that can be used to interpret the data as well as to link the data to other related data captured by some other organizations, making the information consumption as well as integration easier.

2.3.7 Personal Information integrated with information on the web

Mobile Operating systems host personal information software such as contacts, todo list, notes, email, music library, etc., that carry rich personal information relevant to the user, and provide context to the user's activities.

2.3.8 Task and Domain Models for specialized tasks

Specialized task and domain models such as booking a flight, or finding cheapest book online from various e-commerce sites and then providing an interface to book an order are helping automate search, discovery and online order operations.

2.3.9 Cloud computing

Cloud computing has commoditized computer hardware by providing access to servers in centralized data centers managed by cloud providers. This is helping enterprises to provision servers on demand for the duration they are needed, thus enhancing scalability while ensuring that enterprises only pay for the usage. Adoption of open standards in cloud computing environment is accelerating adoption of these technologies as it simplifies the process of building, deploying and maintaining applications.

2.4 CALO

CALO is an acronym for 'Cognitive Assistant that Learns and Organizes', a DARPA funded project that was awarded to SRI International with a goal to create cognitive software systems, that is, systems that can reason, learn from experience, be told what to do, explain what they are doing, reflect on their experience, and respond robustly to surprise. The project brought together leading computer scientists and researchers in artificial intelligence, machine learning, natural language processing, knowledge representation, human-computer interaction, flexible planning, and behavioral studies from 22 organizations.¹

CALO project was run by SRI international until 2008, and was spun off as SIRI to continue work on personal assistant software. Apple acquired SIRI in 2010 and integrated it into its mobile operating system iOS. So, parts of CALO framework as well as the research done on personal assistant software in this project are being

¹ "CALO Website - <http://www.ai.sri.com/project/CALO>."

used in SIRI as well as some other personal assistant software applications that have licensed the technology from SRI international.

2.4.1 Use cases Supported by CALO

Based on the information obtained from Wikipedia², CALO had six separate applications that together form the framework and provide the features of personal assistant software.

Organizing and Prioritizing Information: As the user works with email, appointments, web pages, files, and so forth, CALO uses machine learning algorithms to build a query-able model of who works on which projects, what role they play, how important they are, how documents and deliverables are related to this, etc.

Preparing Information Artifacts: CALO can help its user put together new documents such as PowerPoint presentations, leveraging learning about structure and content from previous documents accessed in the past.

Mediating Human Communications: CALO provides assistance as its user interacts with other people, both in electronic forums (e.g. email) and in physical meetings. If given access to participate in a meeting, CALO automatically generates a meeting transcript, tracks action item assignments, detects roles of participants, and so forth. CALO can also put together a "PrepPak" for a meeting containing information to read ahead of time or have at your fingertips as the meeting progresses.

Task Management: CALO can automate routine tasks for you (e.g. travel authorizations), and can be taught new procedures and task by observing and interacting with the user.

Scheduling and Reasoning in Time: CALO can learn your preferences for when you need things done by, and help you manage your busy schedule

Resource allocation: As part of Task management, CALO can learn to acquire new resources (electronic services and real-world people) to help get a job done.

2.5 SIRI from Apple

SIRI is personal assistant software that interfaces with the user thru voice interface, recognizes commands and acts on them. It learns to adapt to user's speech and thus improves voice recognition over time. It also tries to converse with the user when it does not identify the user request.

It integrates with calendar, contacts and music library applications on the device and also integrates with GPS and camera on the device. It uses location, temporal,

² "CALO Wiki Page - <http://en.wikipedia.org/wiki/CALO>."

social and task based contexts, to personalize the agent behavior specifically to the user at a given point of time.^{3 4 5}

2.5.1 User contexts supported by SIRI

2.5.1.1 Location

SIRI is aware of the current location, and is able to apply this awareness while executing tasks that are dependent on a location. For example, you ask for Italian restaurants and it applies location awareness and queries for Italian restaurants in the vicinity. This can be overridden by making queries based on explicit location.

2.5.1.2 Temporal Context

Temporal context refers to understanding of time, and enables agent to understand the dependency with other tasks, which are to be carried out before, at or after a certain time.

2.5.1.3 Social Context

Social context refers to understanding of user's relationship with his family, colleagues and friends. This information can be explicitly entered into SIRI, by identifying the person name and relationship during the data entry. This enables SIRI to link person names with social relationships. This understanding, even though could be a simple map of personal relations can help agent understand the requests or commands from the user such as ' I want to go to dinner with my wife today'. This helps user to use the same natural language with the agent that is used to communicate with people.

2.5.1.4 Context between other tasks

SIRI understands task context within a conversation with the user. For, example if the user is flying to New York in the afternoon to reach there in the evening, and then asks SIRI to look for nearby restaurants for dinner today, its able to understand the location as New York and suggest locations based on that understanding.

³ "How SIRI Works - Interview with Tom Gruber, CTO of SIRI : [Http://www.novaspivack.com/technology/how-hisiri-works-interview-with-tom-gruber-cto-of-siri.](http://www.novaspivack.com/technology/how-hisiri-works-interview-with-tom-gruber-cto-of-siri.)"

⁴ "SIRI Patent Information - [Http://www.patentlyapple.com/patently-apple/2012/01/apple-introduces-us-to-siri-the-killer-patent.html.](http://www.patentlyapple.com/patently-apple/2012/01/apple-introduces-us-to-siri-the-killer-patent.html.)"

⁵ "SIRI Demo - [Http://vimeo.com/5424527.](http://vimeo.com/5424527.)"

2.5.2 Use cases supported by SIRI

SIRI's approach to personal assistant is to convert user requests first into text, and then convert this text into a task model while using the contextual information. And it uses the task model to map input into API queries to the web services. In order for SIRI to respond to a query, it needs a task model and this also enables SIRI extensibility via new task models. Task models are coded into the software during development, and identify actions with set of web services capable of servicing the request. It's a specialized model that understands a specific task and how to execute it. Supporting new task entails creation of new task models during software development time, and then attaching to sequence of steps to complete that task.

Following are some of the use cases supported by SIRI:

2.5.2.1 Invoking iOS applications based on user request

- Call someone from my contacts list
- Launch an application on my iPhone
- Send a text message to someone
- Set up a meeting on my calendar for 9am tomorrow
- Set an alarm for 5am tomorrow morning
- Play a specific song in my iTunes library
- Enter a new note

2.5.2.2 Calling external services to serve user requests

- Search for text (on search engines – Google, Bing, Wikipedia, etc.)
- Search for a concept (in Wolfram Alpha)
- Get directions from my current location to home
- Tweet a message
- Post a message or photo to Facebook
- Check weather today at a location
- What movies are playing at AMC theater in Cambridge
- Get the latest score for Red sox game today
- Book a table for two at a restaurant in Boston

2.5.3 SIRI System Modules

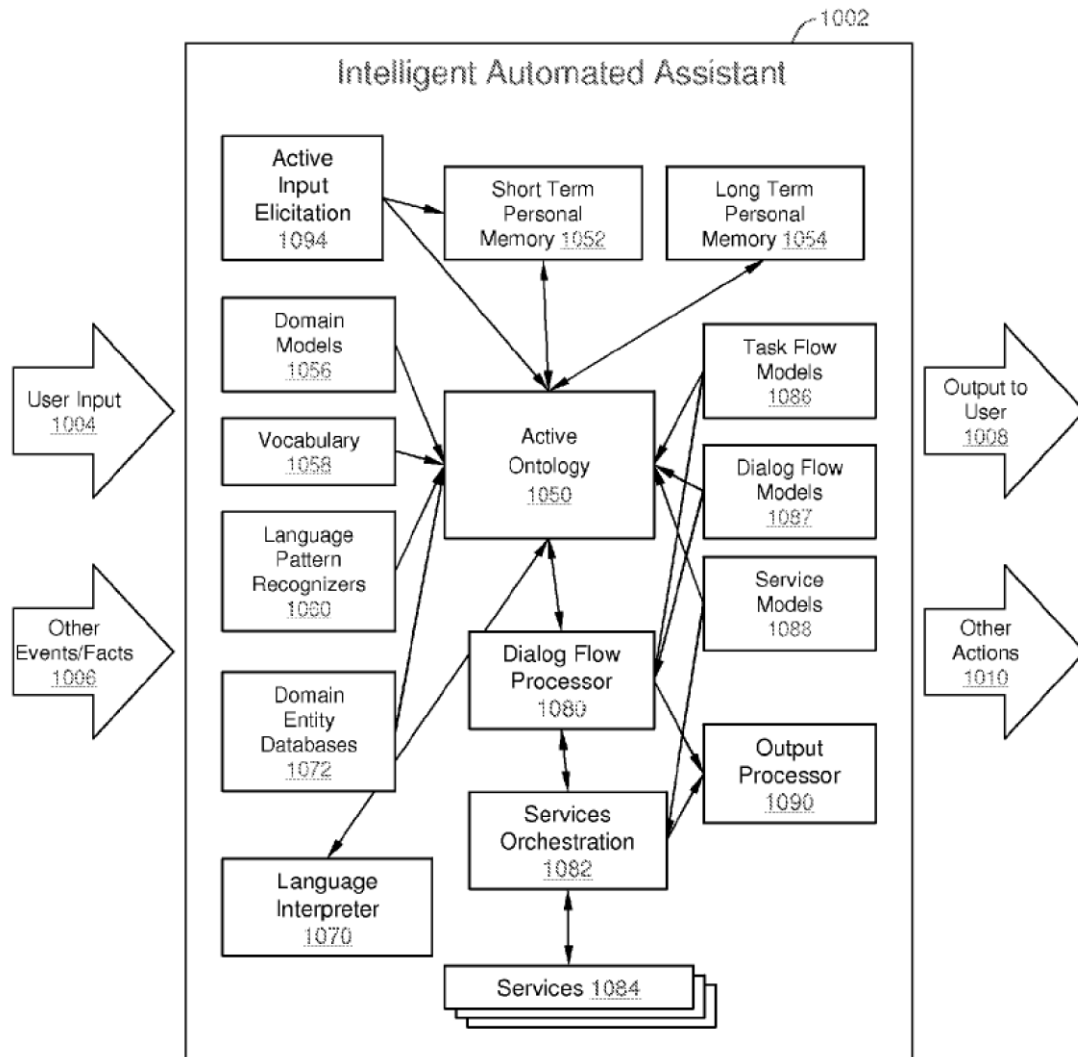


Figure 2-1: SIRI System Modules⁶

Inputs

SIRI takes explicit user inputs thru voice or textual interface. This input is combined with contextual information such as location, time, other task context etc., and fed into its system to identify the task that needs to be executed.

⁶ "SIRI Patent: US20120016678A1 - <http://www.google.com/patents?id=ISECAgAAEBAJ&printsec=abstract&zoom=4#v=onepage&q&f=false>."

NLP Modules

NLP modules include text parsers, vocabulary for the agent, language interpreter. This module is responsible for converting voice input into text.

Memory

It maintains short term as well as long term memory to provide intelligent and personalized task execution to the user. Short-term memory aids the agent to cache answers to related questions and avoid asking the same question multiple times to the user, thus optimizing the performance of the agent. It can use this short-term memory to store output of tasks to represent new state of the environment as well as contextual information of the user.

While long-term memory is used to store user's interests, and patterns in answers that can help agent to predict some of the choices user may make and thus focus on these patterns while querying the services.

Web Service Integration

SIRI depends on external data and web service providers to gather information on specific services available at a location as well as for generic search capabilities. Following are the set of web services it uses for different domains of questions posed by the user:

- **Restaurant and businesses:** OpenTable, Gayot, CitySearch, BooRah, Yelp, Yahoo Local, ReserveTravel, Localeze
- **Events & Tickets:** Eventful, StubHub, and LiveKick
- **Movies and tickets:** MovieTickets, Rotten Tomatoes, and the New York Times
- **Factual Questions:** Bing Answers, Wolfram Alpha, Evi and Wikipedia
- **Web Search:** Bing, Yahoo, and Google, Wikipedia for web search.
- **Maps:** Google Maps and Yelp! Search
- **iOS Applications:** contacts, calendars, clock, reminders, browser, phone call, SMS

Web services provide flexibility as well as modularity to SIRI, as specialized tasks can be offloaded to these web services while focusing on the user interface and integration with these APIs. Set of web services used by SIRI is hard coded into the task models, and so is known at design time as well as run time. This set of web services is updated in order to integrate with any web service supporting the existing tasks or new tasks defined by SIRI's task models.

Dialog Flow Processor

Dialog flow processor embeds itself within each of the user inputs and disambiguation phases while SIRI is trying to map user request to web service calls to external applications or internal iOS applications. This module enables the agent

to keep the context of the question in mind, and ask the user questions in order to gather any incomplete information to complete the task.

Outputs

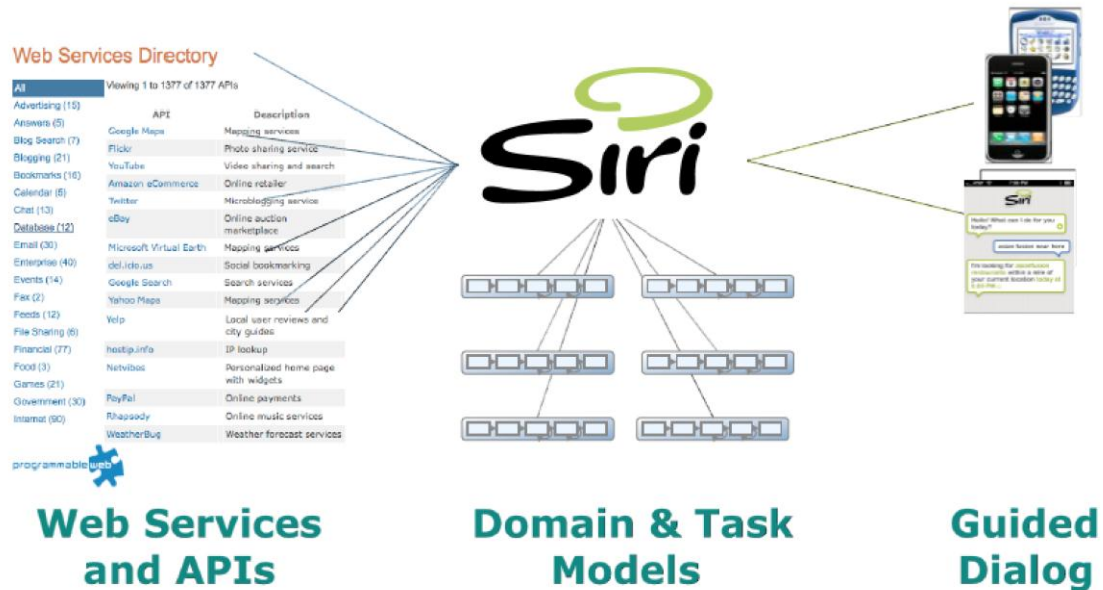


Figure 2-2: SIRI input and output interfaces

Above Figure 2-2 shows, task models as sequence of steps that involve calls to web services to gather information and then having a dialog with the user to complete this task.

Once SIRI identifies the task to be performed, output of the task could be in the form of information gathered by calling a web service or automating a task such as setting up a reminder or creating a new meeting invite. SIRI then maps this output to user consumable form.

2.5.3.1 Interaction between models within SIRI

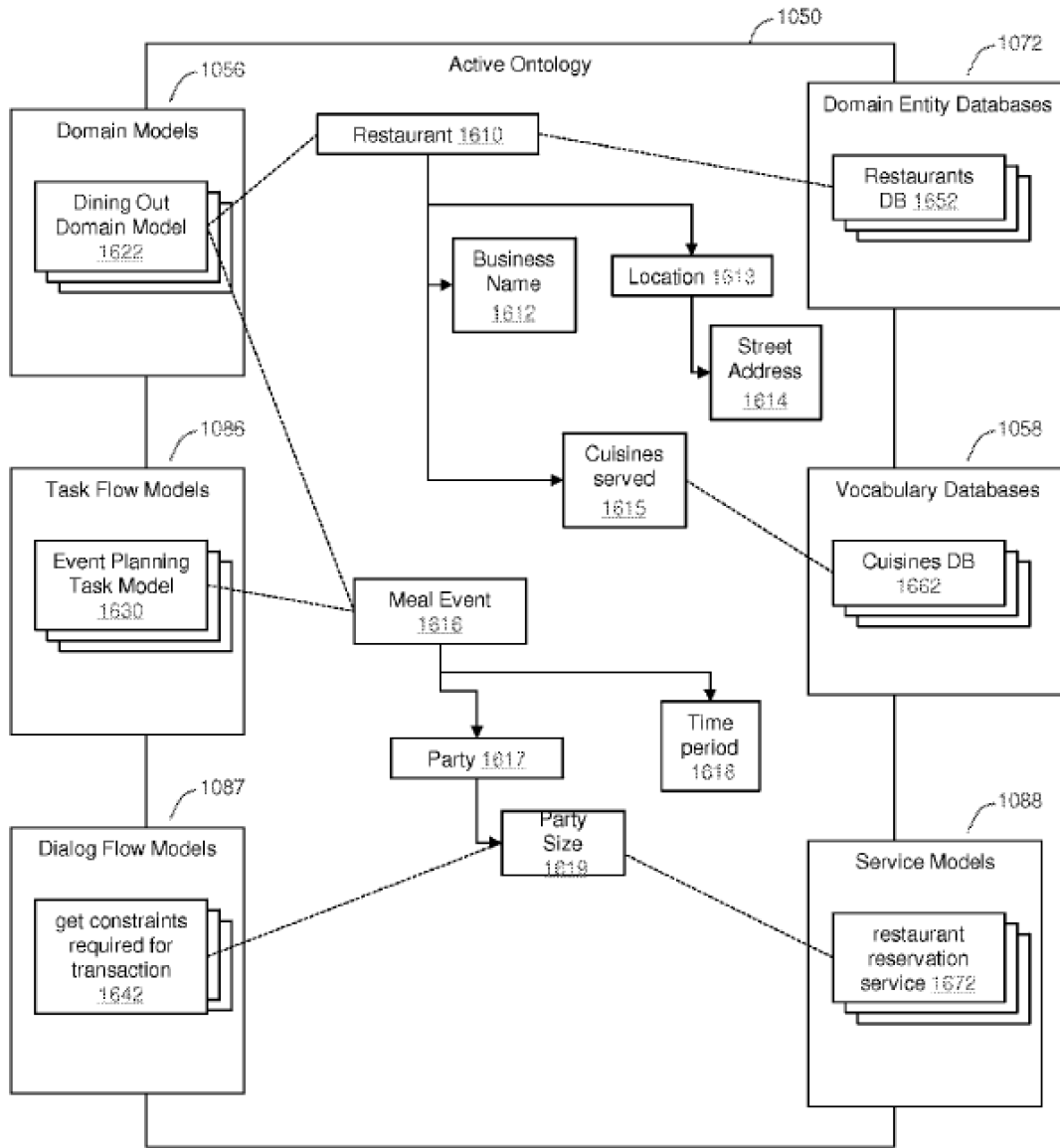


Figure 2-3: Interactions between models within SIRI

Above Figure 2-3 shows different models within SIRI that are designed to model the specific task domain, dialog flow, task flow, entities for the task and integration with web services.

Dialog Flow models shown in the figure 2-3 help the agent to have a conversational dialog with the user and collect information regarding the task to be performed. Any

ambiguities in user input are handled thru dialog. This helps the agent to collect user input in parts, and build on already collected information.

Task Flow models define the workflow for an identified task. These models define set of dependencies for the task, preconditions, task decomposition, and the post conditions. Specializations of Task model implement specific task such as Dining out domain model shown in the figure 2-3. This model enables the agent to plan for the task as well as collect all the relevant information for a given sub task at the time it's required. These models explicitly bind with the web services that can be used to complete the task. This binding of web service is done at the design time, so the agent at execution time knows before hand on set of web services it will be interacting with for a particular task.

Each model maintains a data model that contains data required to accomplish the task as well as the relationships and constraints between the data elements. This data is then used to map to the web services called and the response from the web service is fed back into the data model and converted into user-friendly form for displaying to the user.

This framework of models enables Apple to plug in any new tasks into the agent by defining the specialized model and then binding that to a set of services and then using the binding to execute the tasks at runtime. Agent delegates the process of understanding the task and executing the task to the specialized modules and the web services offered by external data and business service vendors provide the hooks into the external world to complete the task.

This framework enables a single user interface of SIRI to be used to accomplish multiple tasks, integrate information from multiple sources, automate set of tasks in a workflow by defining the domain models specific to the task within SIRI. This is a huge paradigm shift from existing mobile app paradigm of 'There is an App for that'. It makes a single application extensible by embedding the knowledge into the domain classes within the same user interface and application.

2.5.3.2 Salient Features and limitations of SIRI

- SIRI focuses on understanding the spoken language of the user and apply NLP technologies to disambiguate user's request based on the context of the user and its past knowledge of the user. In a way, it tries to engineer a personal assistant by mapping this disambiguated text into actionable tasks that are executed by calling external web services or internal applications running on the mobile device. It simplifies the process of data discovery by identifying and integrating various relevant data sources together and providing an intuitive interface to request for this information. It replaces multiple applications that may be used to individual tasks and automates this process by encoding it in its models and integrating thru APIs.

- SIRI does not maintain a knowledge database of its own and its understanding comes from the information captured in domain models and data models.
- Web service API based task execution leads to proliferation of APIs. In order to support new tasks, entails defining new task models and then mapping steps within the task to call web services. This is a design time activity, and lack of discovery of these services at execution time leads to proliferation of web service APIs within the agent.

2.6 ReQall⁷

ReQall is personal assistant software that runs on smartphones running Apple iOS or Google Android operating system. It helps user to recall notes as well as tasks within a location and time context. It records user inputs and converts them into commands, and monitors current stack of user tasks to proactively suggest actions while considering any changes in the environment. It also presents information based on the context of the user, as well as filter information to the user based on its learned understanding of the priority of that information.

It applies machine learning algorithms on past information of the user – activities, locations visited, routes taken, inputs, user selections and uses this information to proactively present to the user when it identifies a situation that needs user attention.

It can aggregate information from different sources such as emails, calendar, social networks, news feeds, etc., apply filters to this data and present a summary of the day to the user. In a way, it acts as a gateway to all the relevant information for the user. This helps user to avoid going to multiple applications in search of information. And it learns relevancy based on user actions and past user responses. It records user input, locations, other smart phone applications used, and then uses this information to identify patterns to make recommendations to the user.

It understands the activities of the user, and is integrated with user's personal information applications such as address book, calendar, email, and task list and is able to co-relate the impact of current activity on other activities listed in these other applications.

As per Don Norman, Chief Mentor of ReQall – major advances in technology in terms of processing power, memory, network bandwidth, cloud services and advances in AI and natural language processing are acting as enablers of building intelligent personal assistant software.

ReQall uses web services to various data and service providers such as Google, Yelp, Facebook, and Deal sites to find relevant information in a user context. It offloads

⁷ "ReQall Website - [Http://www.reqall.com](http://www.reqall.com)."

management of data to these third party companies who are domain experts in collecting, managing and providing value added services of that data. ReQall by integrating these services thru the API is able to manage tasks of the user while providing a consistent interface to the user across all these applications.

Use cases Supported by ReQall

- **Record voice and convert to text** - Voice command can include what action to be taken with the text – for example, remind me to call my doctor today at 2pm. This will setup a reminder at 2pm, and will ask the user to call the doctor at that time. But, this feature can also be used to transcribe ideas into text using the voice interface
- **Manage Calendar** – It can manage calendar for the user and alert the user of any delays to get to a meeting based on your current and meeting locations. It can read out upcoming appointments to the user.
- **Manage shopping list**
- **Manage task list**
- **Reminders** – reminders for preset calendar events
- **Location based tasks** – it scans thru the task list and presents tasks that can be completed while at a location
- **Learns from usage** – learns user interests, activities, frequency of activities and recommends activities based on location, time and activity context of the user. For example, it can automatically tag important emails by learning user responses to emails based on the information contained in the email. Similarly, it can learn frequency of phone calls to other people and remind user if the threshold identified by the agent is reached.

It integrates with following applications running on the smart phone:

- Reminders
- Email
- Calendar, Google Calendar
- Outlook
- Evernote
- Facebook, LinkedIn
- News Feeds

Salient Features of ReQall

ReQall focuses on collecting voice input from the user at any point of time and location with a goal of helping the user to remember that information at a later time. Contrasting with SIRI, focus is more on data recall, information gathering and partly on alerting for some well known plans such as driving are failing the deadlines. Its not focused on having a conversation with the user or automating certain tasks for the user.

2.7 Google Glass⁸

Google Glass is a wearable computer with a display mounted to the glasses worn by the user. It includes a voice interface to the agent application running on the computer, and the display is an augmented reality that shows the output of the commands.

Based on the queries it supports, it supports controlling on-board devices, invoking APIs to social networking sites such as Google and Facebook, as well as invoking APIs to search engines, maps, and information sites. It processes text input, and is able to convert to commands to the APIs its integrated with.

It handles single command at a time within the location and time context.

Supported Use cases

- Device Integration
 - Take a picture
 - Record Video
- Social Networking
 - Start Google Hangout
 - Send message
- Web services
 - Search for a text or pictures
 - Translate text to a different language
 - Get directions to a location
 - Get weather for a location
 - Get flight details for a flight number

Salient Features

Using augmented reality, Google Glass makes browsing the web easier and available at any time in a hands free manner thru voice commands. With an onboard camera and video camera, it enables the user to take video and pictures at any time by invoking the agent. It also leverages the web services offered by various websites to gather information and present to the user, thus simplifying the information retrieval process. It does not provide a conversational interface yet, and all the commands to the agent are in the form of singular requests, but it can be envisioned that contexts of tasks and social contexts will be added to the agent at a later time.

⁸ “Google Glass Website - [Http://www.google.com/glass](http://www.google.com/glass).”

3 Research and Design methodology

This chapter covers sequence of steps, tools, and processes used to define the scope of the smart agent project, research development tools, research process for looking at existing solutions as well as designing and implementing the system.

Various open source frameworks, platforms and tools were used during the research, review, design and implementation phases of this project and have been covered as part of this thesis in later chapters.

3.1 Research on Sources of Data

This project was started on the premise that there is sufficient amount of openly available data and information on the web that can be utilized to build an agent that has access to making intelligent decisions for routine user activities.

So, the first step in the process was to uncover these sources of data that are relevant to the use cases of the smart agent.

3.1.1 Semantic Web Ontologies

Semantic web ontologies formed a major portion of the research to build this agent. The goal of this exercise was to find existing ontologies that can be used to map the concepts in the agent to an existing concept defined in the ontologies so as to reuse this knowledge of the concept. W3C.org site was mostly leveraged to search for these ontologies. And as a classic example of semantic web, following the links of references of the ontologies used, gave exposure to related ones that could be potentially useful in designing the system.

Another goal of this exercise was to maintain consistency in the definitions of concepts, and that required examination of ontology definitions and references.

Ontologies, which were mapped to other existing ontologies such as mapping of concepts in Cyc to Wordnet as well as UMBEL, help to jump from one definition to the other and also to use each of these ontologies in their specialized use cases. During the process of researching on ontologies, preference was given to existing definitions of ontologies that were mapped to others in order to make the applications interoperable between the domains of ontologies.

Adoption of these ontologies by sources of data as well as authors of other ontologies is critical for its success. Thus, a more widely adopted ontology with a deeper connection and referencing links from other ontologies or data sources

provides a self-validation of the definition of its concepts and also extensibility of those concepts.

3.1.2 Semantic Web Data Sources

Semantic web data sources are the sites that expose their data in some kind of semantic web technology such as RDF. Availability of data in this form enables applications to connect this data to the semantic graph and query for resources. Geographic information from geo-names is an example of how definitions of places (cities, towns, countries) can be used in a standardized way thru the use of semantic data.

Accuracy of the data as well as frequency of updates to the data are also a key feature in deciding which sources to use.

3.1.3 Web services

Web services such as offered by Facebook, Google and Yelp offer a platform independent approach to consuming the data provided by these services thru simple REST APIs.

Underlying data sources for the web services, frequency of updates to the data, whether data used by the web service was available in real time, were some of the considerations while analyzing web services.

3.1.4 Identifying and Defining Scope

Defining scope was an overwhelming exercise as it involved collecting use cases where a smart agent would be useful for a person. Initially, list of use cases where a smart agent would come in handy as a personal assistant to manage or automate the tasks were identified and documented. This turned out to be a wish list for the agent and so specific boundaries were defined based on the availability of data sources, technologies and concepts that could be validated for these use cases. The initial list of use cases was then categorized based on user-agent interactions, and based on type of inputs and outputs.

3.1.4.1 Use case Definitions

Use cases where a smart agent can help the user to accomplish tasks efficiently were identified. One use case was chosen to highlight agent's ability to automate task execution, while another use case was chosen to highlight agent's ability to breakdown the task into sub-tasks and plan manual execution of the task.

Goal of this exercise was to enlist cases where an agent can truly make the use of mobile device platform, services on the web, its understanding of the tasks and other contextual information to execute the tasks efficiently.

3.2 Reviewing existing literature

Existing literature was reviewed in order to better understand the domain of smart agents, knowledge databases, semantic web and other technologies necessary for implementing a smart agent. This was an important exercise to layout the current landscape, get a deeper appreciation of the problem domain, and to look at different alternative solutions for building the agent.

One aspect of reviewing the literature was to get a better understanding of the problem thru examples of already implemented solutions and their feature set. One of the recent attempts to build personal assistant software was a project named 'CALO', an AI project funded by Defense Advanced Research Projects Agency (DARPA) to build personal assistant software. Framework developed as part of CALO project is being licensed to number of applications such as SIRI, TrapIt, Tempo AI. Existing literature was chosen for patents on existing solutions such as SIRI, as well as its core framework CALO. This literature review provided insights into scope of the problem these applications were attempting to solve and also their perspective on the technologies.

Another aspect was to understand the semantic web technologies and knowledge databases that were to be used in building the system. Existing literature in terms of online documentation, papers, books were reviewed so as to gain a deeper understanding of these technologies in order to design the smart agent system using these technologies.

In addition, work done on designing systems that used common sense databases such as CYC and OpenMind were also reviewed to capture some of the challenges in building systems using these databases.

3.3 Reviewing Existing Solutions

Review of existing solutions such as SIRI, Google Glass, ReQall etc., gave insights into features of these applications as well as deeper understanding of use cases supported by these systems. Purpose of this exercise was to identify the scope – use cases supported by these applications as well as use cases that were beyond the scope due to the technology choices made for the implementation of the system. It also provided references to data sources that could be potentially useful in our implementation of the agent. A better understanding of the technology and design choices were helpful in understanding the impact when these applications are adopted in the real world.

4 Overview of Technologies Reviewed and Used

This chapter contains details on technologies and tools evaluated with a goal of designing a smart agent. Semantic web technologies, knowledge databases, Android mobile development framework, cloud based APIs were evaluated for this purpose.

4.1 Semantic Web Technologies

4.1.1 Introduction to Semantic Web

“To a computer, the Web is a flat, boring world, devoid of meaning. This is a pity, as in fact documents on the Web describe real objects and imaginary concepts, and give particular relationships between them. For example, a document might describe a person. The title document to a house describes a house and also the ownership relation with a person. Adding semantics to the Web involves two things: allowing documents, which have information in machine-readable forms, and allowing links to be created with relationship values. Only when we have this extra level of semantics will we be able to use computer power to help us exploit the information to a greater extent than our own reading”

- A quote from Sir Tim Berners-Lee "W3 future directions" keynote, 1st World Wide Web Conference Geneva, May 1994

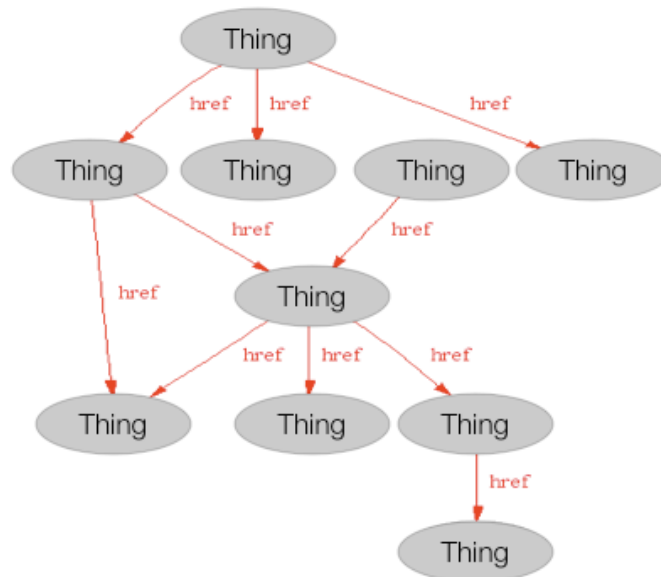


Figure 4-1: WWW - hyperlinked web pages

Figure 4-1 shows HTML pages on websites link to other pages using HTML hyperlinks, forming web of interlinked pages. In this case, links themselves are plain HTML links and do not encapsulate any relation between the pages. So, these links

do provide links to the user to browse the web, but do not integrate the data hosted by these sites in any meaningful way.

Web applications running on these websites use reference data generated by third party vendors by maintaining a copy of a version of this reference data in their databases and consuming it in their applications. This duplication of data results in having stale data when the original source of data changes. It has resulted in silos of databases hosted online, with their own copies of data in a proprietary format that is not easily accessible to others to integrate and use.

Web services overcome this limitation to an extent by defining a service and making the definition of this service available to interested users. Users of web service need to understand service definition that is generally defined in XML format with proprietary attributes defined by the vendor of the service and to carry out translations into their own proprietary attributes. This makes it difficult for machines to discover web services and readily process the data on the web.

However, standards and protocols have evolved to make the interchange of data easier to publish and consume. Standards organizations such as OASIS have defined domain specific vocabularies like ebXML to standardize how a domain data model is represented and how two systems can be integrated using this model. These vocabularies enable consistent data definitions that can be consumed in the applications operating on this data. But, it still needs an explicit contract between publisher as well as consumer to ensure data is interpreted correctly by the applications.

4.1.2 What is Semantic Web?

According to W3C.org,⁹ the Semantic Web is about two things. It is about common formats for integration and combination of data drawn from diverse sources, while the original Web mainly concentrated on the interchange of documents. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases connected not by wires but by being about the same thing.

Data available on the web as well in closed gates of enterprises is characterized by silos of databases containing valuable data about specific domain the applications generating this data belong to. Some of this data is in structured form such as relational databases, which is easy to connect and access but difficult to interpret since they are modeled in a proprietary form. Some of this data is represented in a proprietary format while some of it is represented based on models published by standards organizations.

⁹ "W3C Semantic Web Site - [Http://www.w3.org/2001/sw/](http://www.w3.org/2001/sw/)."

The premise of Semantic Web is to move from silos of databases into a linked database, by providing technologies and tools to enable building this linked database where representation and meaning of data can be transferred from sources of this data to consumers of this data. These technologies enable sharing of data as well as making changes to this data.

4.1.3 Who is contributing to Semantic Web?

Following graph shows number of organizations contributing to the semantic web. This data include government data, health care, media, user generated content, geographic data and knowledge databases.

Links between sites show sharing of data thru shared representations and links and color of the node indicates domain it belongs to.

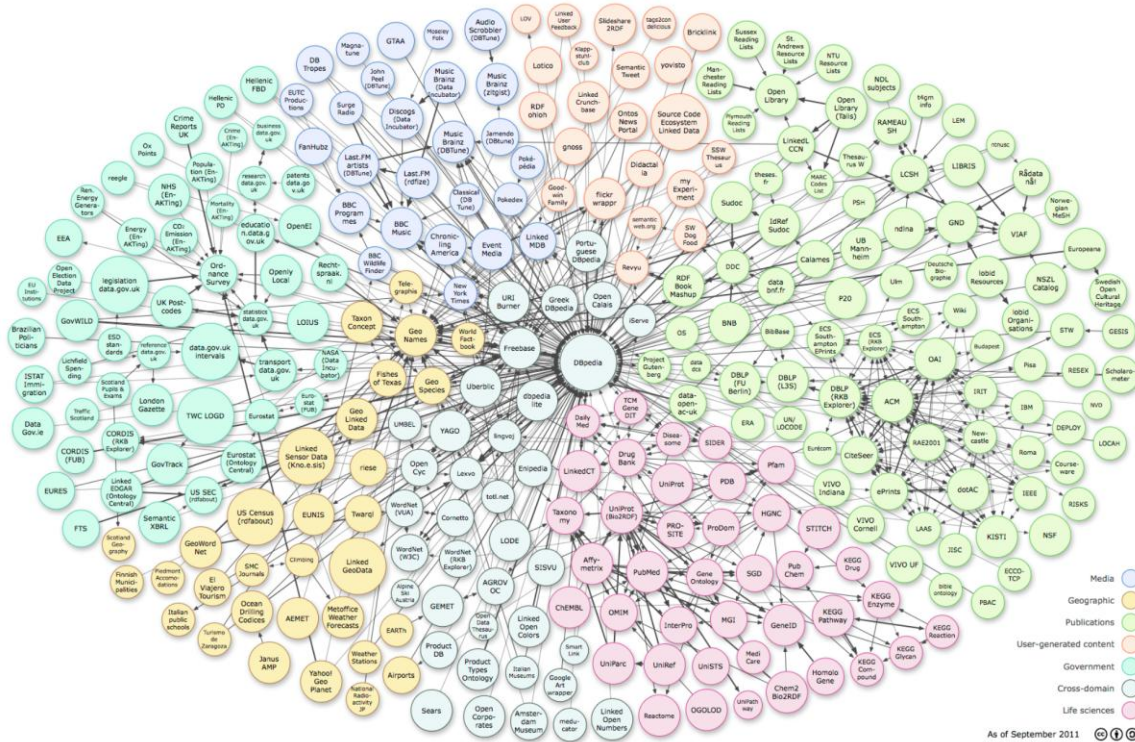


Figure 4-2: Linked Open Data Cloud Diagram¹⁰

4.2 What is covered in this chapter?

Technologies used to power semantic web such as RDF, RDFS, OWL and SPARQL are covered in this chapter. In addition, some of the vocabularies useful for building the

¹⁰ "Linked Open Data Cloud - http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.png."

agent are covered in detail. Knowledge databases OpenCyc is included in this chapter, complementing semantic web technologies.

In addition, this chapter includes sections on cloud based web service APIs that can be used to gather environment details such as weather reports, traffic conditions, store locations as well as Android platform that can be used for mobile device application programming.

4.3 RDF^{11 12}

Resource Description Framework (RDF) is a language for representing knowledge in a distributed world of semantic data. It is a recommendation defined by World Wide Web consortium (W3C) and is intended for widespread deployment of linked data using RDF as a standardized language.

4.3.1 RDF Graph Model

RDF is based on the idea that every object has properties that have values and is represented by using subject-predicate-object triples. Here, subject identifies object of interest, predicate is the property or relation to the object, and object is the value of property represented by predicate on the object.

This paradigm of triples of subject, predicate and object is used to compose snippets of knowledge. These snippets of knowledge (triples) form basis of linked data graph.

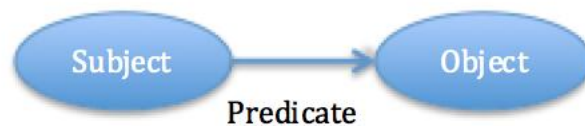


Figure 4-3: RDF Graph Model

RDF uses URI (Uniform Resource Identifier) to uniquely identify resources on the web. These could be network accessible resources such as URLs for web pages, images, or web-services or could be non-network accessible resources such as organization name, products on an ecommerce site or abstract relations defined by ontology vocabularies. Use of URIs enables uniquely identifying resources as well relations on the web, which in turn will lead to sharing of these resources and a common understanding of relations.

¹¹ "RDF Primer - <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>."

¹² "RDF Tutorial -

<http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/>."

4.3.2 RDF Examples

This section will introduce to RDF features thru few examples.

RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a “triple”). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.

Example1: Tim Berners-Lee is the founder of World Wide Web foundation
RDF triple:

http://dbpedia.org/page/Tim_Berners-Lee dbpprop:founder
dbpedia:World_Wide_Web_Foundation

In this representation,

Tim Berners-Lee is subject,
dbpprop:founder is the relation or predicate and
dbpedia:World_Wide_Web_Foundation is the object.

Two namespaces are being used in this representation, namely dbpprop and dbpedia to correspond to DBpedia properties and DBpedia specifications respectively.

This data contains links to definition of the data itself, so it's self-explanatory. This will help in data integration, discovery of data by following the links and other relations on these links

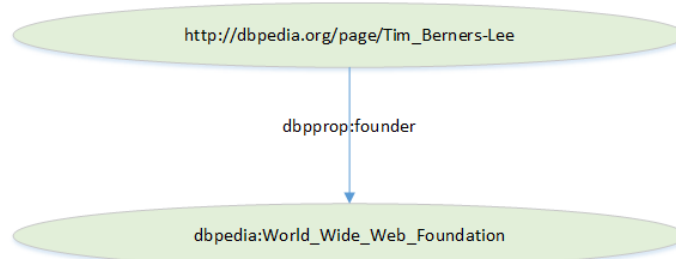


Figure 4-4: RDF Example 1

This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources that are represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.

RDF representation using XML for Example 1 – ‘Time Berners Lee is the founder of World Wide Web Foundation’

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:dbpprop="http://dbpedia.org/property/"xmlns:dbpedia="http://dbpedia.org/">
  <rdf:Description rdf:about="http://dbpedia.org/page/Tim_Berners-Lee">
    <dbpprop:founder>dbpedia:World_Wide_Web_Foundation</dbpprop:founder>
  </rdf:Description>
</rdf:RDF>
```

Example2:

Tim Berners-Lee is author of book - ‘Weaving the web: The Original Design and Ultimate Destiny of the World Wide Web by its inventor’. Harper Collins published this book.

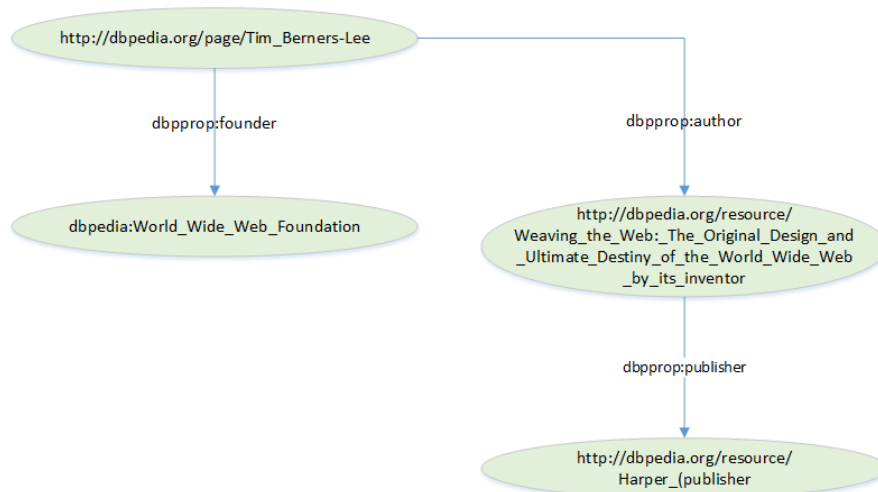


Figure 4-5: RDF Example 2

In this example, we are able to link a book to its publishing company. And an application or user once connected to the publisher URL, will be able to query for additional details of the publisher. This Publisher URL can be hosted by the publisher themselves, and so have updated information on other titles, their location etc.

This example shows how data can be traversed from one node to the other by following the relations.

4.3.3 Features of RDF

- RDF enables information to be represented in machine-readable form that applications can process.
- Data Reuse thru single copy of data
- Meaning of the data is always included with the data, making it easy for interpretation and processing.
- Data Aggregation from various sources
- Different parts of the data can be owned by different entities, but thru an integrated graph this data can be explored and processed
- In Semantic Web, change is not expensive. Since all resources are identified thru URI, changes can be made in one place and the values propagated along the graph to any one interested in the updated values.
- Supports the evolution of schemas over time without requiring all the data consumers to be changed.

As a general principle, always use URIs to represent entities as well as relations. And for representing relations, reuse existing vocabularies as much as possible, extending them only in special cases when required relation is not already defined. One of the challenges in reusing vocabularies is standardizing process while these vocabularies are being built. It is addressed by providing interoperability between two similar vocabularies defining same entities.

4.4 RDFS¹³

RDF data model defines relationships between resources as properties and values. It does not offer any mechanism to define the properties of resources or relationship between two properties or between properties and resources.

RDFS is RDF vocabulary description language, which can be used to define properties and resources. It defines the type system for RDF models by defining resources and properties such as `rdfs:Class` and `rdfs:subClassOf`, which are then used for defining relationship between resources and properties.

¹³ "RDFS - <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>."

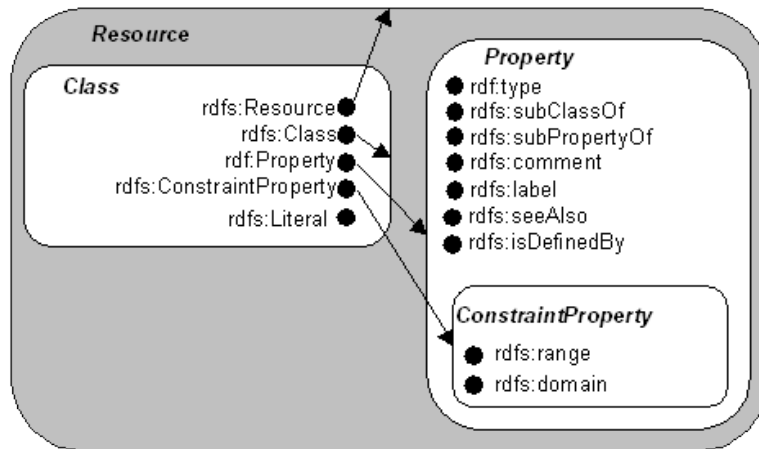


Figure 4-6: Classes and Resources as Sets and Elements¹⁴

Rounded rectangles are the classes and classes enclosed within a rectangle have a subClassOf relation with outer rectangle. Large Dots indicate they are resources, and the arrow from resource points to the defining class of the resource.

Core Classes

1. rdfs:Resource is the top level class in RDFS, and defines all things described by RDF expressions.
2. rdf:Property is a sub-class of rdfs:Resource and represents properties in RDF expressions.
3. rdfs:Class represents type of the resource. There is a rdf:type property link from resource to the class defining that resource.

Core Properties

1. rdf:type property represents membership of the resource to a class and specifies type of the resource.
2. rdfs:subClassOf specifies class inheritance relationship of a subclass. This property is transitive, as a sub class will inherit all properties of its parent as well all its grand parents.
3. rdfs:subPropertyOf is an instance of rdfs:Property and indicates that it's a specialized property.
4. rdfs:seeAlso specifies link to a resource that can provide additional information about the resource.
5. rdfs:isDefinedBy is a subPropertyOf rdfs:seeAlso and generally used to link to the schema defining the resource.

¹⁴ "RDFS Classes and Resources - <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/sets.gif>."

Constraints

RDFS defines constraints for end points of a property using domain and range constraints

1. `rdfs:domain` identifies instances of classes on which a particular property can be applied. It specifies which type of subject nodes can have this property.
2. `rdfs:range` identifies instances of classes which can take values for a particular property. It specifies which type of object nodes can have this property.

4.5 SPARQL^{15 16 17}

SPARQL stands for SPARQL Protocol and RDF Query Language, and defines protocol to connect to RDF graph data stores. SPARQL queries are executed over one default graph and zero or more named graphs that are specified by the URIs in the query.

It helps to

- Query RDF information and represent the results.
- Join disparate RDF databases using complex joins.
- Transform results from one ontology to another ontology.
- Explore the data and relationships stored in RDF data stores.

This section will cover syntax of SPARQL to query, filter and present the RDF data and also other uses of SPARQL such as constructing RDF graph or describing an existing one of RDF graph data stores.

SPARQL queries are executed against SPARQL endpoints, which provide an interface to one or many RDF datasets. A SPARQL endpoint accepts queries via HTTP and provides results in XML, JSON, RDF or HTML formats.

RDF data stores can expose a SPARQL endpoint using HTTP protocol. Any applications interested in RDF data can then connect using HTTP and RDF library that can work with RDF data and work with SPARQL queries.

4.5.1 Anatomy of a SPARQL Query

A SPARQL query contains five sections

1. Prefix declarations – used to abbreviate URIs so that the query looks shorter and cleaner
2. Dataset definition – identify RDF graphs to be queried
3. Result clause – identify what nodes or information to be retrieved from the graph

¹⁵ “SPARQL W3C Resources - <http://www.w3.org/2009/Talks/0615-qbe/>.”

¹⁶ “SPARQL Tutorial - <Http://www.slideshare.net/ldodds/sparql-tutorial>.”

¹⁷ “SPARQL Cheatsheet - <http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet>.”

4. Query pattern – conditions to be matched on the subject, predicate and objects and related entities. Identify exactly what kind of nodes are being queried in the underlying RDF graph
5. Query modifiers – ordering, limiting of the results

4.5.2 SPARQL Syntax by example

4.5.2.1 *Selecting triple pattern from RDF graph*

Following is an example to query DBpedia (RDF version of Wikipedia) for querying 100 distinct object types from DBpedia RDF repository. This is achieved by querying for a property `rdf:type` that captures type information of a resource, and filtering only unique 100 of these types in the result.

```
#prefix declarations
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

#dataset definition
FROM <http://dbpedia.org>

#result clause
SELECT DISTINCT ?Concept

#query pattern
WHERE
{?Instance rdf:type ?Concept}

#query modifiers
LIMIT 100
```

PREFIX is used to abbreviate the URIs. For example, RDF URI is abbreviated to `rdf` using PREFIX above. Predicate type declared by RDF can then be referenced by using the shortened notation - “`rdf:type`”.

SELECT command is used to query the RDF data store.

DISTINCT can be used with a node or a literal, and will take out the duplicates and return only unique results.

Select query also identifies values that should be returned by the query as results. In the above example, `?Concept` refers to any nodes that satisfy the criterion in where clause to be included in the results. `?Concept` is a SPARQL variable and can match any node or literal in the RDF dataset. In this query, only objects referenced by the predicate `rdf:type` will be assigned to `?Concept` variable. Variables can also be used to match the values taken by the variable during execution of the query.

FROM identifies RDF graph that is to be queried using publicly accessible HTTP interface. In this example, DBpedia's SPARQL endpoint is being queried.

WHERE clause lists set of conditions using a triple pattern of subject-predicate-object.

Triple pattern used in where clause - {?instance a ?Concept}, is like triples used in RDF, but can contain variables whose names start with "?".

This triple pattern, asks SPARQL query engine to match all nodes connected by predicate rdf:type. It names the subject node with variable name "?instance" and object node with variable name "?Concept".

LIMIT keyword tells the SPARQL engine to limit the number of results, in this case to 100.

4.5.3 SPARQL Query against BestBuy RDF product dataset

BestBuy is using GoodRelations ontology and has exposed its real time product catalog in RDF form using GoodRelations ontology. This lets applications to query this RDF dataset using SPARQL. GoodRelations Ontology is covered in section 4.7.1 in this chapter.

Following example can be used to query BestBuy dataset to get name and price information of all products having a name beginning with "Apple" and which are priced between \$100 and \$500.

```
PREFIX gr: http://purl.org/goodrelations/v1#
FROM <http://metis.bbyopen.com/sparql>
SELECT ?name ?price
WHERE {
    ?offering gr:includesObject ?object;
    gr:hasPriceSpecification ?ps .
    ?ps gr:hasCurrencyValue ?price .
    ?object gr:typeOfGood ?good .
    ?good gr:hasMakeAndModel ?make .
    ?make gr:name ?name .
    FILTER( ?price > '100.00'^^^xsd:float && ?price < '500.00'^^^xsd:float
    && regex(?name, '^Apple', 'i'))
} ORDER BY DESC(?price)
LIMIT 100
```

Offering is the root node for a product, and includes nodes for price specification and make and model of the product. These two nodes are separately used in the above query to query for products priced between a price range and by the name of the product.

4.5.4 Types of SPARQL Queries

In addition to querying for RDF graph from RDF data stores, SPARQL also supports Ask, Describe and Construct functions.

In addition to SELECT queries explained above, SPARQL also supports ASK, DESCRIBE and CONSTRUCT queries and these are covered below.

4.5.4.1 ASK – Ask RDF data store to ask a query and get a Boolean answer

Following query asks DBpedia RDF data store whether Amazon river is longer than Nile river, by getting the length attribute of both the rivers (referenced here by the URIs) and then comparing the length and returning result of comparison expression.

```
PREFIX prop: <http://dbpedia.org/property/>
ASK
FROM <http://dbpedia.org>
{
  <http://dbpedia.org/resource/Amazon_River> prop:length ?amazon .
  <http://dbpedia.org/resource/Nile> prop:length ?nile .
  FILTER(?amazon > ?nile) .
}
```

4.5.4.2 Describe

Describe clause is used to get description of an RDF node. Information returned is not standardized and is dependent on the implementation. Generally class information or short RDF graph of the node is returned.

```
#prefix declarations
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
#dataset definition
FROM <http://dbpedia.org>
#result clause
DESCRIBE ?Concept

#query pattern
WHERE
{?Instance rdf:type ?Concept}
```

4.5.4.3 Construct to create a new graph

Construct clause is used to construct a new graph based on the query results on an existing graph. This is useful when transforming an existing graph structure to a new one.

In the example below, an RDF database using FOAF ontology is queried for name, home page URL and title. All of these properties are defined in FOAF ontology.

Values of these properties on a resource are then used to create a new resource with the same values but using vCard ontology. In this query, new graph is created by using the values from existing graph and by mapping the properties between vCard and FOAF.

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?X vCard:FN ?name .
  ?X vCard:URL ?url .
  ?X vCard:TITLE ?title .
}
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
  OPTIONAL { ?X foaf:name ?name . FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:homepage ?url . FILTER isURI(?url) . }
  OPTIONAL { ?X foaf:title ?title . FILTER isLiteral(?title) . }
}
```

4.5.5 Limitations in current SPARQL 1.1 version, covered in new SPARQL 2¹⁸

1. Support for inserting, updating and deleting into RDF data store using SPARQL
2. Support for embedding sub-queries within SPARQL queries
3. Support for grouping results with aggregate functions such as count, min, max, average or sum
4. Support for negation in queries
5. Support for projected expressions allow query results to contain constants, functions, or any other expression.
6. Support for federated query to run SPARQL query on multiple endpoints and combine the results
7. Support for discovering SPARQL endpoint service in order to let the users discover the service and use it.
8. Support in SPARQL for OWL and RDF Schema

¹⁸ "Limitations in SPARQL 1.1 - <http://www.w3.org/2009/Talks/0615-qbe/>."

4.6 Ontologies¹⁹

According to W3C, ontology is a set of precise descriptive statements about some part of the world (usually referred to as the *domain of interest* or the *subject matter* of the ontology). Precise descriptions satisfy several purposes: most notably, they prevent misunderstandings in human communication and they ensure that software behaves in a uniform, predictable way and works well with other software.

Ontologies capture domain knowledge of a particular domain in an explicit form to foster shared understanding and reusing this domain knowledge. This domain knowledge of explicit specification of concepts, relations or properties, and constraints in precise descriptive form renders itself for machine processing. Machines can use captured knowledge from a given ontology to infer new knowledge and to validate that captured knowledge is consistent.

There are two W3C recommendations for defining ontologies, namely RDFS and OWL. These two recommendations are covered here.

RDFS defines vocabulary for defining classes, hierarchy relations between classes, properties and constraints on properties. Ontologies defined using RDFS, so are limited by this vocabulary.

4.6.1 Web Ontology Language (OWL)

According W3C, Web Ontology Language 2 (OWL 2) is a knowledge representation language designed to formulate, exchange and reason with knowledge about a domain of interest. It is used to represent knowledge about things, groups of things and relations between them.

It provides classes, properties, individuals or instances of classes and data values to capture domain knowledge. All this domain knowledge is represented in RDF and stored as semantic web document that can be referenced by other related and interested documents to extend the knowledge as well as knowledge representation.

4.6.2 OWL 2 Features²⁰

4.6.2.1 OWL 2 is a language for expressing ontologies.

Ontology consists of concepts that describe domain of interest. This is terminology knowledge and is particular to the domain.

¹⁹ "OWL2 Primer - [http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic Web Presentation](http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic%20Web%20Presentation),"

²⁰ "W3C OWL2 Specification - <http://www.w3.org/TR/owl2-syntax/>."

In order to precisely describe a domain of interest, it is helpful to come up with a set of central terms – often called vocabulary – and fix their meaning. Besides a concise natural language definition, the meaning of a term can be characterized by stating how this term is interrelated to the other terms. A *terminology*, providing a vocabulary together with such interrelation information constitutes an essential part of a typical OWL 2 document.

Besides this terminological knowledge, ontology also contains assertional knowledge that deals with concrete objects of the considered domain rather than general notions.

4.6.2.2 OWL 2 is not a programming language, but is declarative.

OWL uses description logic to describe the knowledge of the domain and is declarative. Tools can be built to process this knowledge and infer new knowledge, but the ontology language itself is not a programming language.

4.6.2.3 OWL 2 is not a schema language for syntax conformance.

OWL does not define syntactic constraints on the values of concepts it describes. XML schema (XSD) or DTD can be used to define syntactic constraints on the data.

4.6.2.4 OWL 2 is not a database framework.

DB schema can be compared with OWL terminology knowledge and DB content can be compared with assertional knowledge, but the two are quite different. DB schema does not have inference capabilities between types, and using closed world assumption – an absence of fact in database is considered false while in description logic, which uses open world assumption, an absence of a fact merely means fact is not known.

4.6.3 Modeling Knowledge

Axioms

Ontology modeling using OWL 2 captures explicit knowledge of domain using basic constructs such as statement or prepositions. Individual statements form basic part of the domain knowledge and are called axioms in OWL2. These statements evaluated to Boolean true or false under a given set of conditions.

Some example statements:

Every man is a mammal

Every week has seven days

OWL uses axioms as building blocks of knowledge, defining new axioms as consequence of existing axioms. OWL reasoning engines such as Apache Jena can be used to infer based on consequences.

Entities

All atomic constituents of a statement like = 'Mary is a female' are called entities.

Real world object in the statement that can be identified is denoted as individuals, category of the object is denoted as a class and the relation is denoted by properties in OWL.

There are three types of properties in OWL:

1. Object properties
Relationship between two objects is captured by object properties.
2. Data type properties
Age property of a person is an example of data type property
3. Annotation properties
Annotation properties annotate ontology itself. For example, linking the ontology to a class or author of the ontology to a class is an annotation property.

4.6.4 OWL by example ²¹

1. subClassOf relation between two classes
A statement 'Person is a mammal' is represented in OWL as:
Person rdfs:subClassOf Mammal

In OWL, all classes have owl:Class as the root class

2. Relationships between classes
In OWL, classes can have following relationships
equivalentClass
intersectionOf
unionOf
complementOf

Defining, Male and Female as two classes, we can use these class relationships to capture Male and Female relation in real world

```
Male a owl:Class.  
Female a owl:Class.
```

```
Person can be defined as a union of Male and Female classes.  
Person a owl:Class;  
Owl:unionOf(Male Female).
```

²¹ "OWL2 Primer - <http://www.w3.org/TR/2012/REC-owl2-primer-20121211/>Semantic Web Presentation."

In addition, we can make use of complementOf class relation between Male and Female classes.

```
Female owl:complementOf Male.
```

A parent can be defined as a subclass of a person

```
Parent rdfs:subClassOf Person.
```

Now, we can define Father and Mother classes as Parent class who are Male and Female respectively.

```
Father a owl:Class;  
    owl:intersectionOf(Male Parent).
```

```
Mother a owl:Class;  
    owl:intersectionOf(Female Parent).
```

3. Defining properties using constraints
RDFS domain and range properties can be used to constraint which class can be a subject or an object for a given property.

A person has a parent

```
hasParent rdf:type rdf:Property;  
    rdfs:domain Person;  
    rdfs:range Parent.
```

A person's father is a male

```
hasFather rdfs:subPropertyOf hasParent;  
    rdfs:domain Person;  
    rdfs:range Male.
```

A person's mother is a female

```
hasMother rdfs:subPropertyOf hasParent;  
    rdfs:domain Person;  
    rdfs:range Female.
```

These examples also show how properties can be related to other properties using subPropertyOf relation.

4. Types of Properties
 - a. Object Property defines relationships between two classes
hasParent property defined earlier can be redefined as an object property as it relates to two classes.

```
hasParent rdf:type rdf:ObjectProperty;
          rdfs:domain Person;
          rdfs:range Parent.
```

- a. Datatype Property defines relationships between instance of a class and a literal that can be represented using a datatype

Age of a person is an example of a Datatype property.

```
age a owl:DatatypeProperty;
    rdfs:domain Person;
    rdfs:range xsd:integer.
```

5. Types of Properties

- a. TransitiveProperty

A property is transitive if $P(x,y)$ and $P(y, z)$ then $P(x, z)$

```
relatedTo a owl:TransitiveProperty
```

- b. SymmetricProperty

A property is symmetric if the property is true in both directions.

$P(x, y)$ iff $P(y, x)$

```
hasSibling a owl:SymmetricProperty.
```

Joe hasSibling Carol; implies
Carol hasSibling Joe;

- c. FunctionalProperty

$P(x, y)$ and $P(x, z)$ implies $x=z$

```
hasBirthMother a owl:FunctionalProperty;
```

Joe hasBirthMother Alice;
Joe hasBirthMother Alicia;
Implies, Alice = Alicia

- d. InverseOfProperty

$P1(x, y)$ iff $P2(y, x)$

```
hasWife owl:inverseOf hasHusband;
Joe hasWife Amy;
Implies, Amy hasHusband Joe;
```

6. Property restrictions

New classes can be defined by placing restrictions on values of the class instances. For example, we can create a new class JoesSiblings by filtering all Person class objects whose hasBrother points to Joe.

```
JoesSiblings rdfs:subClassOf Person;
[ a owl:Restriction;
```

```
owl:onProperty brother;  
owl:hasValue Joe].
```

7. Cardinality constraints on property

New classes can be defined by constraining min, max or exact cardinality on the returned objects for a property.

```
PersonsWithTwoKids rdfs:subClassOf Person;  
[ a owl:Restriction;  
owl:cardinality "2"^^xsd:nonNegativeInteger;  
owl:onProperty hasParent].
```

4.7 Evaluating Ontologies for Smart Agent

4.7.1 GoodRelations

GoodRelations is a semantic ontology that defines e-commerce domain information model that enables e-commerce sites to publish information about their products or services in a form that can be discovered by search engines, product or service recommendation services or any other applications that understand this ontology. It defines four high level entities – BusinessEntity, ProductOrService, Offering and Location and number of other support classes to represent an online business, its products, specific offerings and locations that are supported in delivering these products. Using this ontology, a business can define all the its offerings, features or attributes of these offerings, payment options, delivery options, locations where its available as well as open times of this location.

Use of linked data enables these sites to use pre-defined ontologies to define product categories, payment services, delivery of services and avoid the duplication of data. For example, payment options such as credit card payment, check payment, cash on delivery, PayPal are defined in this GoodRelations vocabulary and an agent querying the web across multiple e-commerce sites will be able to compare products readily if the sites use a standardized vocabulary for defining the attributes of product offering in an e-commerce context.

Features of GoodRelations Ontology

- GoodRelations is an industry neutral way of representing an online business, supports a site selling products as well as services across different industries such as electronics, software, apparel, cars, electronics maintenance services, real estate, restaurants etc. This core feature of the ontology helps the standardization of tools and processes that can be used to deploy ecommerce sites for discovering the products offered by the site and processing related information.
- It supports RDF/XML, RDF-tuple, RDFa, and JSON to expose the ecommerce data on the web.

- It captures product attributes which can be queried to filter products on the site and also enables availability information at a particular store location, at a particular time for a given product based on attributes of the store and inventory information.
- It captures relationships between two products such as similar to, complementary to, variant of, consumable of, and accessory of, which can be used to recommend products to customers based on their choices in the shopping cart.
- GoodRelations ontology builds on classes defined by schema.org, FOAF, vCard and OWL.

The GoodRelations Ontology for E-Commerce
 Language Overview - UML Class Diagram
 Version 1; Release 2011-10-01
 Martin Hepp, mhepp@computer.org

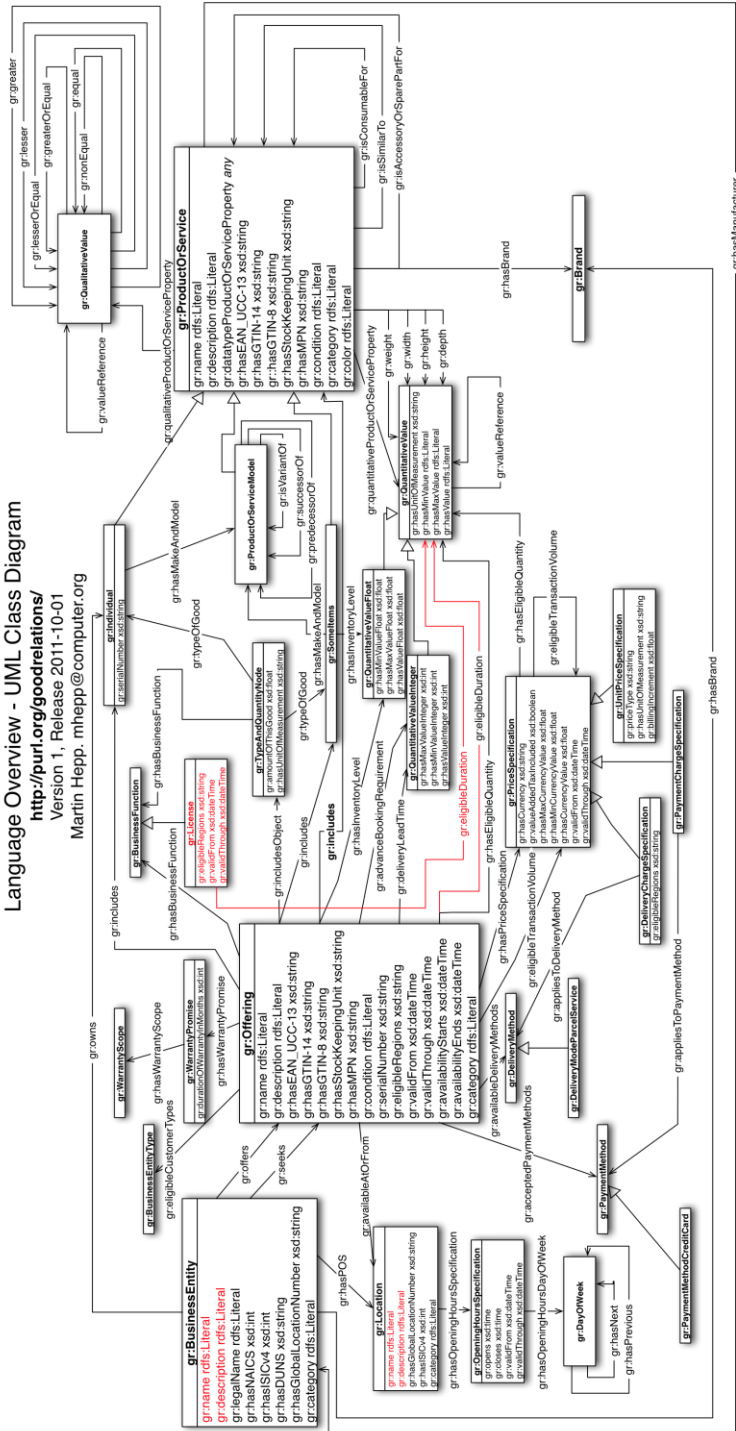


Figure 4-7: GoodRelations UML Diagram²²

Description of High-level classes:

22 “GoodRelations Ontology UML Diagram -
 Http://www.heppnetz.de/ontologies/goodrelations/v1#uml.”



- Notes:
- The following GoodRelations elements are only shortcuts for simpler annotation or querying. See the documentation at <http://purl.org/goodrelations/> for details:
 - gr:hasValue (shortcut for setting both hasMinValue and hasMaxValue properties to the same value in one turn)
 - gr:hasValueInteger (shortcut for setting both hasMinValueInteger and hasMaxValueInteger properties to the same value in one turn)
 - The following elements are now deprecated, but you can still use them, e.g. for staying compatible with older data consumers (e.g. Yahoo SearchMonkey):
 - gr:ProductOrServiceSomeInstancesNow (now gr:SomeItems)
 - gr:ProductOrServiceSomeInstancesFormerly (now gr:FormerlySomeItems)
 - gr:LocationOSalesOrServiceProviding (now gr:Location)
 - For the recommended cardinality of attributes, see the GoodRelations Language Reference at <http://purl.org/goodrelations/v1.html>.
 - For the recommended cardinality of objects, see the GoodRelations Language Reference at <http://purl.org/goodrelations/v1.html>.
 - gr:name and gr:description can now be attached to any GoodRelations type, but this is not shown here for readability.
- Red highlighting indicates elements added or changed in this release.

BusinessEntity class in this ontology is defined by creating a union of existing classes to define Organization (<http://schema.org/Organization>) and Person (<http://schema.org/Person>). This class represents an individual person or an organization that is selling its products or services online, and includes primary mailing address of the entity and geo-location information. By reusing Organization class of schema.org, it captures and references industry standard organization identifiers such as ISIC and DUNS.

BusinessEntity class also captures the relationships with locations this entity operates in as well as Offerings of products or services on the site.

ProductOrService class defines type of product or service that is being sold by the business entity. It is the superclass of all products and services offered by the site. Some of examples are – Car Rental, Duracell battery, Sony Camera etc.

Offering class represents an offer to sell a certain product or service by the business entity to a predetermined set of customers. Offering definition includes constraints on business functions offered, location, time, quantity, delivery method and pricing type. These constraints define particular constraints this offering is provided to the customers.

Location class represents locations where the product or service is available to the customers of this business, whereas BusinessEntity refers to legal entity of the business. A given business such as a car rental company will have one legal entity and multiple locations for renting out cars that are represented using Location class. Each instance of this class will correspond to one store and will capture address, geo-location as well as open hours of that location during all days of the week. This class is equivalent to <http://schema.org/Place>. This can be useful in case of personal assistant software application that is trying to locate availability of products within a certain distance from the user's current location.

4.7.1.1 How does it help spread of Linked Data?

Standardizing domain information and sharing this domain knowledge across different e-commerce sites while reusing the definitions of the products and entities across the web will help spread the use of linked data within ecommerce sites. This can also lead to defining a product such as a 300ml Coke in a single location and then referencing the product from multiple sites that sell this product. Product manufacturers and distributors can collaborate to capture product definitions using this ontology within a central repository, resulting in a single linked data repository with vast information on the products. An e-commerce site can leverage linked data about the products it sells and the e-commerce domain knowledge captured by this ontology and contributing to web of linked data by publishing availability of instances of its products and services in this format.

4.7.1.2 Example RDF/XML for a CD 'Feats Don't fail me now' on bestbuy.com

<?xml version="1.0" encoding="UTF-8"			
rdf:RDF	@xmlns:rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	
	@xmlns:foaf	http://xmlns.com/foaf/0.1/	
	@xmlns:gr	http://purl.org/goodrelations/v1#	
	@xmlns:owl	http://www.w3.org/2002/07/owl#	
	@xmlns:rdfs	http://www.w3.org/2000/01/rdf-schema#	
	@xmlns:review	http://purl.org/stuff/rev	
	@xmlns:xsd	http://www.w3.org/2001/XMLSchema#	
	gr:Busine...	@rdf:about	gr:offers
	(2 rows)		gr:legalName
	1 #BusinessEntity_BestBuy	gr:offers	@rdf:resource #Offering_413190
	2 #Manufacturer_Warner_Bros.		Warner Bros.
	gr:Offering	@rdf:ID	Offering_413190
	rdfs:label	@xml:lang	en
		#text	Feats Don't Fail Me Now - CD
	gr:hasPriceSpecification	gr:UnitPriceSpecification	gr:DeliveryChargeSpecification
	(3 rows)	1	gr:UnitPriceSpecification
		2	gr:UnitPriceSpecification
		3	gr:DeliveryChargeSpecification
	gr:availableDeliveryMethods	@rdf:resource	http://purl.org/goodrelations/v1#DeliveryModeFreight
	gr:availabilityStarts	@rdf:datatype	http://www.w3.org/2001/XMLSchema#date
		#text	1990-03-29
	gr:includesObject	gr:TypeAndQuantityNode	
	gr:hasBusinessFunction	@rdf:resource	http://purl.org/goodrelations/v1#Sell
	gr:eligibleRegions	@rdf:datatype	http://www.w3.org/2001/XMLSchema#string
		#text	US
	gr:eligibleCustomerTypes	@rdf:resource	
	(3 rows)	1	http://purl.org/goodrelations/v1#Enduser
		2	http://purl.org/goodrelations/v1#PublicInstitution
		3	http://purl.org/goodrelations/v1#Reseller
	gr:acceptedPaymentMethods	@rdf:resource	
	(5 rows)	1	http://purl.org/goodrelations/v1#Cash
		2	http://purl.org/goodrelations/v1#MasterCard
		3	http://purl.org/goodrelations/v1#VISA
		4	http://purl.org/goodrelations/v1#AmericanExpress
		5	http://purl.org/goodrelations/v1#Discover
	gr:availableDeliveryMethods	@rdf:resource	
	(3 rows)	1	http://purl.org/goodrelations/v1#UPS
		2	http://purl.org/goodrelations/v1#DeliveryModeMail
		3	http://purl.org/goodrelations/v1#DeliveryModePickUp
	rdfs:seeAlso	@rdf:resource	http://www.bestbuy.com/site/Feats+Don't+Fail+Me+Now+-+CD/413190.p?id=89357&skuId=413190&cmp=RMX&ky=2f7kYCrhpL10L9Wkf1ZMJIPh5LamXp60

Figure 4-8: GoodRelations example RDF data in XML form

This example shows how a product such as CD can be offered for sale on an ecommerce site like bestbuy.com. It identifies pricing information, delivery methods, product availability and payment methods that are relevant for e-commerce. This information when exposed in RDF form becomes machine readable and can be accessed by agent software to automatically check availability of products in different stores based on geographic location or any other filtering criterion.

4.7.1.3 Leveraging GoodRelations ontology

1. An e-commerce site can post all the products, along with product attributes, pricing, delivery, discounts information online such that all of this data can be discovered and queried thru the semantic web tools.
2. Search Engines can use this information to refine query results in the search results based on the meta-data gathered in this microdata format. Microdata is a set of standard tags that are understood by the search engines.
3. Applications can query for real time information of the retailers or e-tailers to find latest information on products, and repeat this across different sources of data that follow the same ontology.
4. Ecommerce sites can use GoodRelations vocabulary to manage their business and build a web of linked data for e-commerce.

4.7.1.4 How does use of GoodRelations help in building a smart agent

GoodRelations schema captures all the necessary details of defining entities in an ecommerce transaction. Applications can discover this data and process it for specific needs since this data available in linked data RDF form. It also fosters sharing common vocabulary in ecommerce domain, thus standardizing toolsets that use this ontology. From the smart agent's perspective, adoption of GoodRelations ontology that standardizes ontology used for ecommerce will enable automation of online e-commerce related tasks.

Adoption of GoodRelations ontology for representing products as well as on e-commerce sites to sell these products is a challenge. This adoption has been accelerated by providing plugins into databases of popular e-commerce platforms that can expose the product data captured in relational database into RDF form, and then exposing subset of this RDF data as microdata tags to search engines, enabling search engines to query on the product attributes of ecommerce site.

GoodRelations ontology defines sameAs property with other ontologies such as schema.org to identify properties in its ontology that are similar to other ontologies. This enables properties defined across ontologies to be interpreted consistently for semantic web tools that are traversing the linked data graph.

4.7.2 DBpedia²³ ²⁴

DBpedia organizes content from Wikipedia in structured RDF form representing the data in RDF classes and properties. Wikipedia consists of articles that contain both structured data in the form of info-box fields, page links to other articles, categorization, images, geo-coordinates, as well as unstructured data in natural language text. DBpedia uses data extraction tools to retrieve structured data from Wikipedia and store in semantic RDF form. This extracted information uses a consistent DBpedia ontology to represent information about persons, places, music albums, films, video games, organizations, biological species, and diseases. DBpedia uses an ontology that defines the hierarchy of classes for the information captured in Wikipedia along with predicates to define characteristics of the class.

DBpedia uses an automatic extraction engine that reads Wikipedia, resolves ambiguities in mapping different names of info-box properties to the same class and property in DBpedia and stores the data in RDF form.

DBpedia provides RDF links to other RDF datasets such as Wordnet, Cyc, UMBEL, Schema.org and Freebase.com. This cross-referencing enables reuse of information across these different datasets along with semantic meaning associated thru the ontologies being used.

DBpedia can be used to query this structured knowledge derived from world's largest online encyclopedia - Wikipedia, to build systems that can make logical queries on this data while understanding the ontology used by DBpedia. For example, applications can query DBpedia for places such as airports based on location attributes as well as the category of the place, and get a list of airports around the location.

DBpedia is a knowledge base that spans multiple domains of knowledge, is maintained by community of contributors and is available freely on the web.

Use of DBpedia for Smart Agent:

Structured knowledge stored in DBpedia along with SPARQL endpoints that make access to this data using logical queries, make it a good candidate for use with smart agent.

²³ "DBpedia Knowledge Base - <http://www.wiwiss.fu-berlin.de/en/institute/pwo/bizer/research/publications/Mendes-Jakob-Bizer-DBpedia-LREC2012.pdf>."

²⁴ "DBpedia Usecases - <http://wiki.dbpedia.org/UseCases>."

4.7.3 Wordnet^{25 26 27}

Wordnet is a lexical database of nouns, verbs, adjectives and adverbs in English language that are organized based on the lexical and semantic relationship of a given word with other words in the database, and forming a semantic graph that links entities, events and properties that are represented by these words. Related words in database are in close proximity in contrast to traditional dictionaries where proximity is based on the starting letter of the word.

It includes two kinds of relationships between words – lexical and conceptual. Lexical relations are based on word usage while conceptual relationships are based on the concepts a word represents.

4.7.3.1 Lexical Relationships:

Synonyms:

Wordnet captures synonyms for a given word in synsets that contain a set of words, which can be used to replace the word without altering the meaning of the sentence in a given context.

Antonyms:

Antonym is a lexical relation between word forms and not a semantic relation between word meanings. For example, words such as rise/fall are antonyms, but rise/descend are not antonyms, even though fall and descend are synonyms. This relationship between set of words is captured in Wordnet.

4.7.3.2 Conceptual Relationships:

Hyponymy/Hypernymy:

These semantic relationships capture hierarchy of concepts similar to inheritance hierarchy in object-oriented systems. A hyponym of a word inherits all the properties of its generic concept hypernym and is generally called isA relation. It also adds new set of features that distinguish it from its hyponyms and hypernyms.

Meronymy/Holonymy:

These semantic relationships represent part-whole relationship between words. For example, hand is a part of body is captured thru this relationship.

²⁵ “Wordnet: Design, Content and Limitations by Christian Fellbaum - <http://dydan.rutgers.edu/Workshops/Semantics/slides/fellbaum.pdf>.”

²⁶ “Introduction to Wordnet - an Online Lexical Database - <Http://wordnetcode.princeton.edu/5papers.pdf>.”

²⁷ “Wordnet Reference Manual - <http://wordnet.princeton.edu/man/wngloss.7WN.html>.”

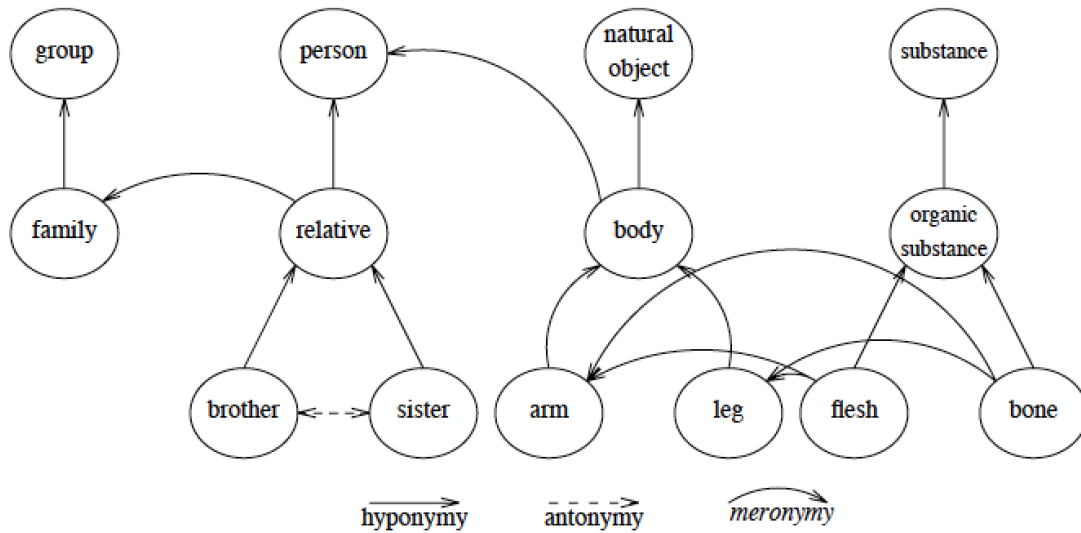


Figure 4-9: Wordnet hyponym, antonym and meronym relationship example

Figure 4-9 shows an example of hyponym, antonym and meronym relationships in Wordnet. 'Brother' is a 'Relative', who in turn is a 'Person' in this hierarchical tree capturing hyponym/hypernym relationships between words. On the other hand, 'brother' is an antonym of 'sister', and is captured by the antonymy link between the two words. Relationship between 'arm' and 'body' is a part-whole relationship that is captured by the meronym link represented as an arc.

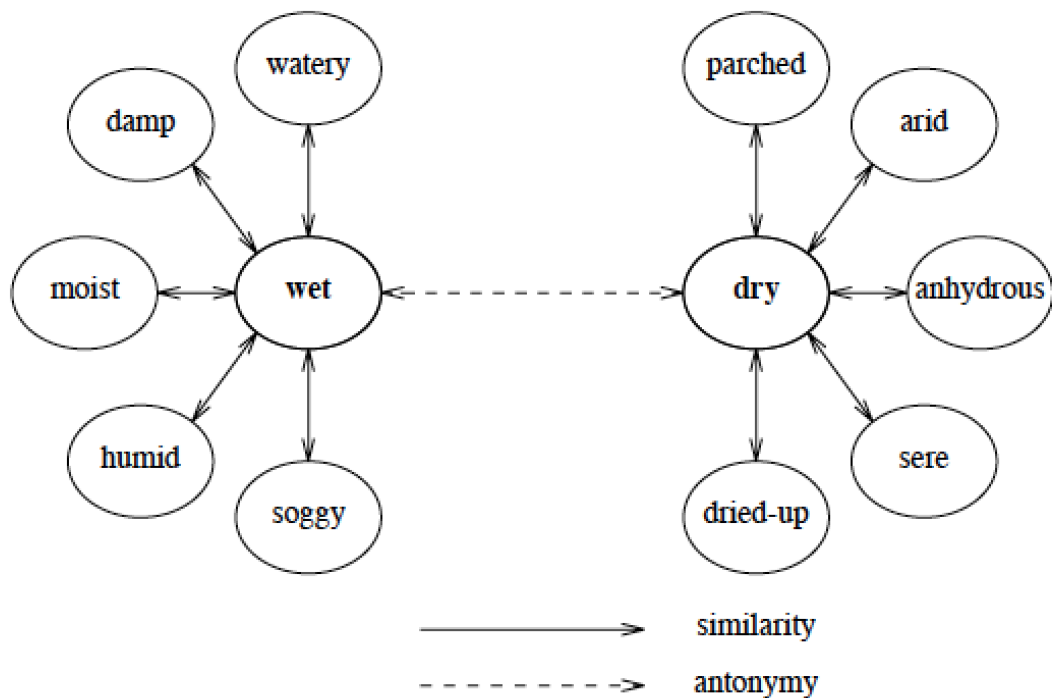


Figure 4-10: Wordnet synsets and antonyms example

This example shows synsets for two adjective words wet and dry, which are also linked with each other in an antonym relationship.

These relationships can be useful to parse the input text from the user and identify synonyms for the word in order to gather related information from other data sources.

4.7.3.3 Wordnet Ontology and RDF Data

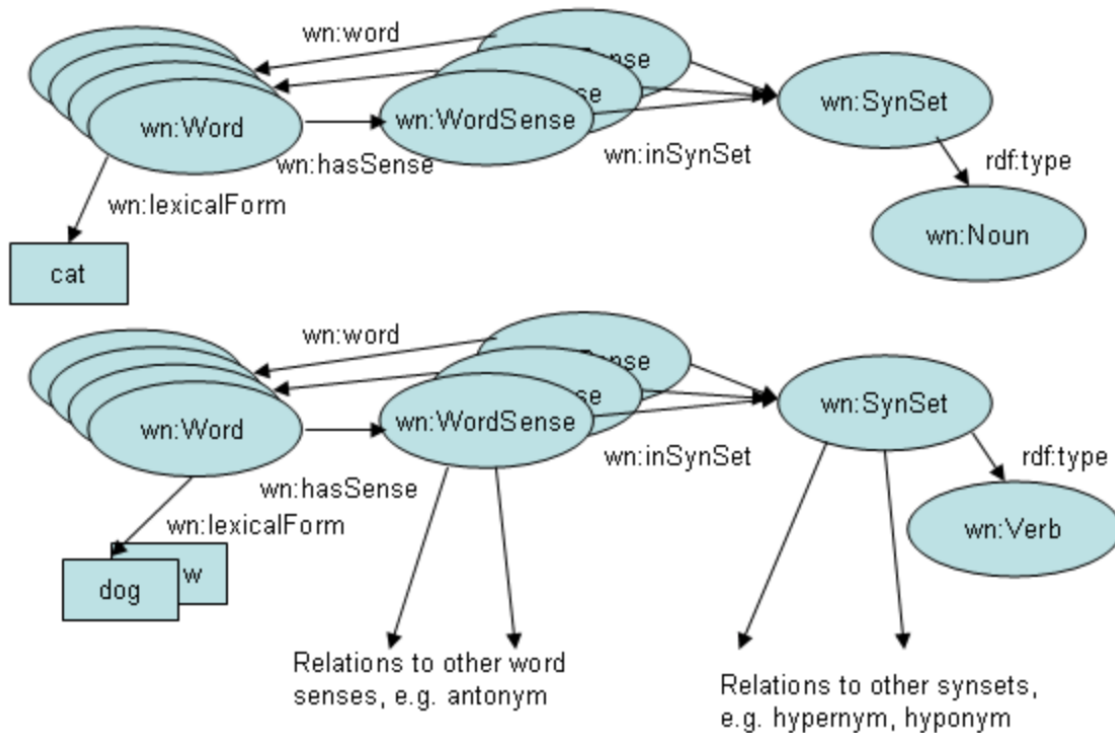


Figure 4-11: RDF representation of words in Wordnet

Wordnet OWL ontology captures lexical and semantic relationships captured in Wordnet. Wordnet database is also available in RDF form, making it easy to query it using RDF and OWL tools.

OpenCyc contains links to words in Wordnet. So, using Wordnet while parsing and identifying the word in synset links to the concept defined in OpenCyc. For a given text input, Wordnet to OpenCyc RDF links enable the agent to identify the concept in OpenCyc and gather more information about the concept.

4.8 Knowledge Databases

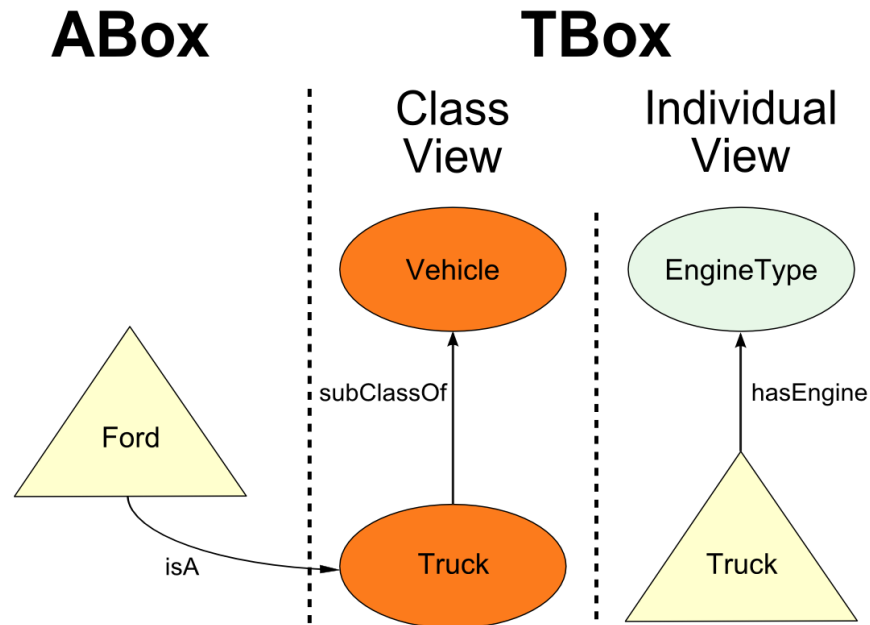


Figure 4-12: ABox and TBox Systems²⁸

Knowledge based systems have two components associated with knowledge and are categorized into ABox systems or TBox systems based on whether they capture knowledge related to instances of real world objects or just contain domain knowledge which is also called ontology.

TBox refers to terminological knowledge, contains taxonomy or ontology information about the concepts and relationships between concepts in that domain. In the above diagram, a TBox view will have an understanding of EngineType and Truck and relationships between them, but no knowledge of any occurrence of those concepts in the real world. These systems capture the domain knowledge and can be considered as reference ontology that can be used to populate the instance data in order to build a knowledge data with the assertions from the real world instances pertaining to this ontology.

Another type of knowledge system refers to ABox, where A refers to assertions. It uses TBox definitions as the basis to populate instance specific assertions, and relationships between instances. TBox enable building the structure of knowledge while ABox uses this structure to capture the knowledge about real world objects such as persons, organizations, places, events etc.

UMBEL is an example of reference ontology that is derived from Cyc ontology,

²⁸ "Knowledge Databases - [Http://www.mkbergman.com/913/metamodeling-in-domain-ontologies/](http://www.mkbergman.com/913/metamodeling-in-domain-ontologies/)."

containing more focused concepts and focuses primarily on ontology development and not on knowledge capture of instances. While, Cyc is an ABox as well as the TBox system, as it captures both the type information in its knowledge building process and then populates that with instance specific information.

4.8.1 OpenCyc

Cyc's concepts are organized into one big "ontology". An ontology is like a taxonomy, but with much richer interconnections between terms. It contains both implicit as well as explicit knowledge of the domain.

OpenCyc is open source version of Cyc, a knowledge database and an inference engine. The knowledge database was created by manually encoding the knowledge about the world using a knowledge representation language called CycL. Core of the idea is to encode everyday knowledge in a structured language within logical expressions that can be used for reasoning. A knowledge base can be used to encode knowledge about various concepts and their inter-relationships, in terms of facts, and rules expressed in a knowledge representation language.

A knowledge base such as Cyc adds background knowledge about the concepts with in a domain that has been digitized thru careful codification of interaction as well as the real world concepts by the engineers who understand the Cyc architecture and language. It includes 500 thousand terms, 17 thousand relationships between these concepts, and seven million assertions regarding these terms. In addition to the explicitly entered assertions, inference engine enables creation of new assertions thru the relationship between existing terms, relationships and assertions.

Inference engine enables applications to be built without hard coding rules, and to generate new assertions based on the already existing knowledge and the facts known at the time to the application.

We can use smart agent as an interface for human computer collaboration, while leveraging knowledge stored in knowledge databases. These knowledge databases store knowledge in machine understandable form. Computers with high computational power and memory are better equipped with processing this knowledge in a given context. Humans are good at reasoning with incomplete information. Smart agent connects strengths of humans and machines using some of the available technologies that enable a real time availability of the information.

4.8.1.1 Why use logic to represent knowledge? ²⁹

Cyc uses CycL language to represent knowledge. A logic based reasoning such as CycL makes it possible to express the knowledge using logical expressions precisely without any ambiguity that is generally associated with natural language. In

²⁹ "Why Use Logic? - Presentation at [Http://www.cyc.org](http://www.cyc.org)."

addition, this embedded knowledge is use neutral and can be queried across multiple uses for reasoning using reasoning engines that can infer from the captured knowledge.

In CycL, the meanings of statements and inferential connections between statements are encoded in a way that is accessible to a machine. At the present time Natural Languages are virtually meaningless to machines. For example, a sentence such as “all animals have spinal cords. All dogs are animals. My pet is a dog.” From these sentences, a *person* can infer that my pet has a spinal cord, but a *machine* cannot, at least not until a machine can understand English sentences. A knowledge engineer can encode these assertions in Cyc so that inferences can be made about the dog when needed.

4.8.1.2 What kind of knowledge is encoded in OpenCyc

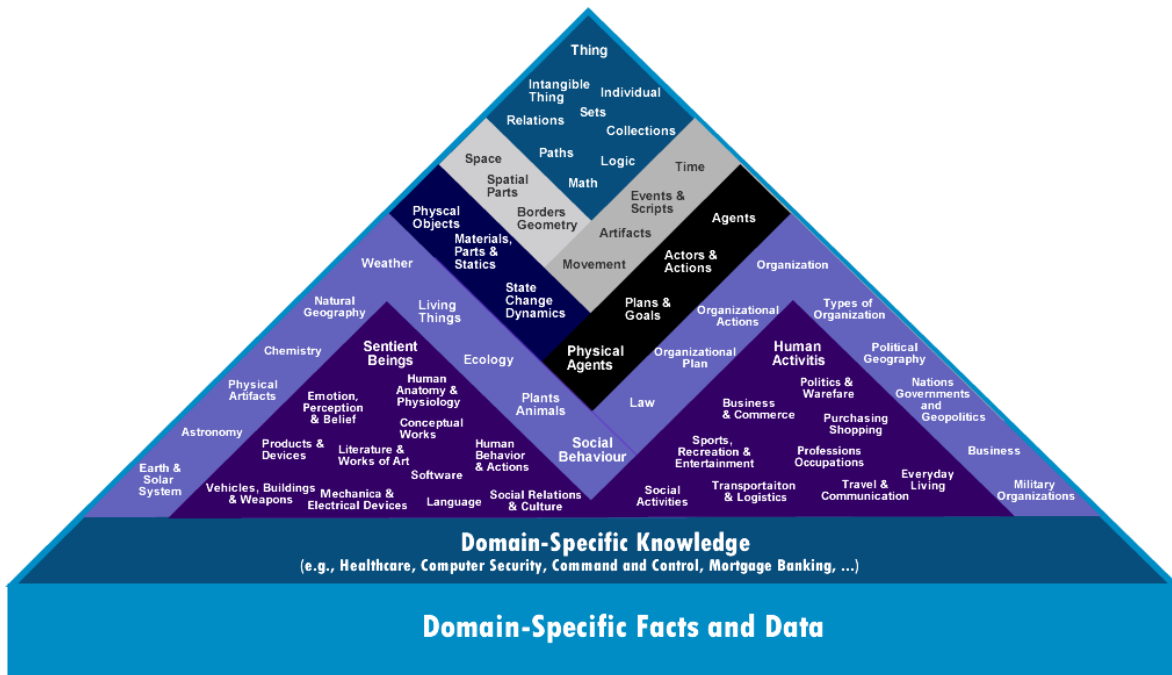


Figure 4-13: OpenCyc Knowledge database

This diagram shows Cyc Knowledge base consisting of a massive taxonomy of concepts and relationships between concepts. It represents the knowledge arranged by degree of generality with the top level representing abstract general concepts and real world facts at the bottom.

At the top level called upper ontology, “Thing” is Cyc’s most general concept. Everything in CYC is an instance of a “Thing.” Top level represents very abstract concepts - For example, it contains the assertions to the effect that every event is a temporal thing, every temporal thing is an individual, and every individual is a thing.

Below the upper ontology, Cyc contains a layer of knowledge called Core theories. These theories contain facts about space, time and causality and are useful for reasoning about the facts defined in lower layers.

Domain specific theories contain more specific theories than the core theories. They capture knowledge about specific domains such as finance, chemistry, weather, organization etc., which are useful in understanding the domain.

Final layer is the Domain specific facts and data layer, which contains real world facts and use instances of knowledge domain objects defined in layers above.

4.8.1.3 Reasons for choosing OpenCyc

Following are the reasons behind choosing OpenCyc as knowledge database to complement with semantic version of Wikipedia (DBpedia)³⁰

Venerable and solid — through an estimated 1000 person years of engineering and effort over more than 20 years, the Cyc structure has been tested and refined through many projects and applications

Community — there is a large community of Cyc users and supporters from academic, government, commercial and non-profit realms

Comprehensive — no existing system has the scope, breadth and coverage of human concepts to match that of Cyc (however, Wikipedia now exceeds Cyc as a source of reference information on instances and individuals)

Common sense — Cyc has set out to capture the common sense at the heart of human reasoning. This objective means codifying generally unstated logic and rules-of-thumb that leads to a solid basis for its reasoning and conceptual relationships

Power and inference — Cyc has about a thousand microtheories governing its inference domains, giving it a contextual scope and power unmatched by other systems

Broad functionality — its knowledge base capabilities can be deeply leveraged in such areas as entity extraction, machine translation, natural language processing, risk analysis or one of the other dozens of specialty modules

Free and open — OpenCyc is a free and open source version that has been downloaded more than 100,000 times

Upgrade path — OpenCyc has an upgrade path to the more capable ResearchCyc, full Cyc and the services of Cycorp.

³⁰ “UMBEL for Ontology Development - [Http://fgiasson.com/blog/index.php/2008/08/29/umbel-as-a-coherent-framework-to-support-ontology-development/](http://fgiasson.com/blog/index.php/2008/08/29/umbel-as-a-coherent-framework-to-support-ontology-development/).”

4.8.1.4 Components of Cyc

4.8.1.4.1 Concepts

Cyc uses CycL to capture knowledge in its knowledge base. It stores ontology as well as assertional knowledge and so belongs to ABox knowledge systems.

It uses Collections class to capture type information and Individual class for instance specific information. It supports identifying these Collections and Individual classes using a unique names.

Cyc uses CycL sentences are used to capture assertional knowledge using predicate functions or logical terms on arguments. Logical terms in these CycL sentence can be - and, or, not, forall, implies, thereExists etc.

CycL sentences can be used to capture knowledge in terms of predicate logic, as shown in examples below. All the terms in CycL begin with #\$.

Example1: Earth is a Planet

```
(#$isa #Earth #Planet)
```

#\$isa - predicate function that checks if the first argument is a type of second argument. In this case it will return true if Earth is a type of Planet.

#\$Planet - Planet type defined by Cyc

Example2: Earth orbits around the sun

```
(#$orbits #Earth #Sun)
```

#\$orbits - predicate which checks if the first argument orbits around the second argument.

Example 3: There is at least one planet that orbits around the sun

```
(#$thereExists ?PLANET
```

```
  (#$and
```

```
    (#$is ?PLANET #Planet)
```

```
    (#$orbits ?PLANET #Sun)))
```

This CycL sentence states that there is at least one instance of something named PLANET, which is an instance of Planet type and that orbits around the Sun.

4.8.1.4.2 Microtheories

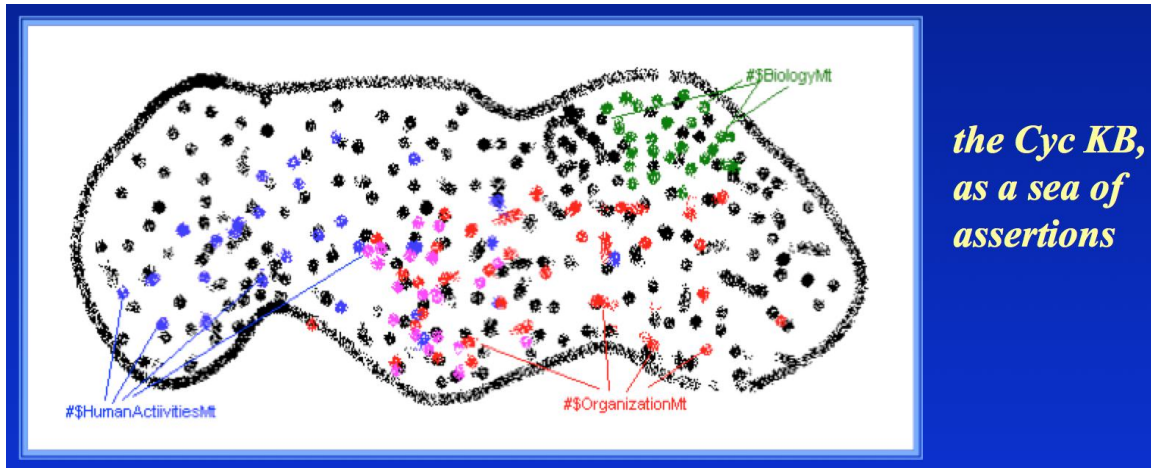


Figure 4-14: Microtheory in Cyc

The Cyc Knowledge Base (KB) can be thought of as a vast sea of assertions. A microtheory in Cyc is a set of assertions from this sea of assertions, grouped together based on shared topics, assumptions, sources or other features.

Each microtheory holds set of consistent assertions related to the scope of the topic that microtheory addresses. So, each microtheory will hold assertions at granularity intended to serve its purpose for the user of microtheory. An assertion is visible in all and only the microtheories that inherit from the microtheory in which it is placed.

Collecting assertions by grouping them in these microtheories enables modularization of knowledge topics, improving efficiencies in managing this assertion knowledge as well as in search and inference process. Microtheories make knowledge base building more efficient.

Microtheories provide a context of the topic, time, assumptions within which assertions can be captured and new ones inferred. This enables inconsistent assertions to be placed in different microtheories based on assumptions and timelines of their specific microtheory.

4.8.1.4.3 Events

Events in Cyc are represented as individuals that belong to a collection called `#$Event`. Examples of Events in Cyc include reading, transportation, negotiating, buying, planning, campaigning etc. These events occur over time and have temporal context.

Components of Events:

Events can have performers or **actors**, and there can be devices that performers use during the events.

Events can occur at places, and those places are somehow involved in the events. Events take place at time, and times of events are also somehow involved in events.

Each of these events can have **sub-events** and can be modified as our understanding of the events change. Events are organized in a hierarchical fashion deriving from generic ones such as `TransportationEvent` into specialized events such as `LandTransportationEvent` or `AirTransportationEvent`.

Roles connect Events to its Components:

Components involved in events are related to the events using **Roles** predicates. Role predicates can capture the constraints on type of roles that can participate in an event. In addition, these predicates can themselves be part of a hierarchy thus inheriting high-level knowledge already captured about the type.

4.9 Online Web Services

Some of the enterprises are exposing APIs to their proprietary databases and business logic in order to spread the information to other developers who can build more value added applications by including their data as part of their applications and reusing the data already collected by the vendors of this API. These web services generally need developers to sign up with the vendor and get an Application key to identify their application while making the calls to these APIs. In cases where application developer needs access to user data that is not openly available to anyone other than the user, vendors provide authentication protocols such as OAuth that enable the users to grant permission to application developer to gain access to user data, with an understanding that it will not be compromised by the application developer and will be used for providing value added services to the user.

These web services are valuable sources of information and can be embedded with in personal assistant software to make use of relevant data in order to manage and plan the user tasks.

4.9.1 Yelp: Local Business Database API ³¹

Yelp is an online web site that captures business directory information along side user generated review content about these businesses. They maintain a list of categories of businesses for various urban locations within US as well as in Europe. It hosts a community of users reviewing the services offered by businesses to help other users while choosing businesses.

³¹ "Yelp Web services Documentation - [Http://www.yelp.com/developers/documentation](http://www.yelp.com/developers/documentation)."

Yelp provides a web services REST API to this information that can be used to search for specific categories of places based on location, some search terms, user ratings or timings of operation.

Yelp API supports searching for business based on the name, category of the services provided by the business or any keyword related to business and then providing a location name or location co-ordinates. In response, Yelp API will return all the businesses that match the search criteria of business name or category of service as well as associated businesses in the location that are similar to the one searched by name or by category. So, searching for a specific Indian restaurant in Boston will also return related Indian and Asian restaurants around the area as they serve the same product category.

This type of an API can thus be useful to help the agent find relevant businesses within the context of task and location that can be further filtered based on user preferences and past history.

4.9.2 National Weather Service web service to weather information

National Weather service has an open API to developers interested in getting weather information. This information is provided as part of open data initiative by US government at opendata.gov and is available as a SOAP web service that provides weather information in XML form.

Web service API supports querying for weather conditions at a specific location for a given day or range of days. And also provides weather conditions for a locations enclosed in a grid. This can be useful when user is interested in weather conditions along a travel path.

Weather information is also available thru web sites such as weather.com, who gain this data from NWS and expose it as their own web service. This information is useful while planning for activities such as walking to a place or driving to a place that are impacted by the weather at the place.

4.9.3 Google Web services³²

Google provides web services to manage user personal information such as emails, calendar, task list, contacts etc., on Google apps. These web services can this be used to get access to user data and also to user friends' data who have shared these apps with the user. These web services can help the agent in managing collaboration between these people.

³² "Google Webservices Documentation - [https://developers.google.com/maps/documentation/webservices/.](https://developers.google.com/maps/documentation/webservices/)"

4.9.3.1 List of Google web services for personal information:

- Google Calendar: Google Calendar API lets applications to access user's calendar to create new events, edit or delete existing ones and to search for events. This API can also be used to access any user's public calendar or access a user's private calendar with authorization. This API will help agent to gain access to user as well as his network's calendar.
- Google Task list: Google Tasks API lets applications to access user's tasks to create new tasks, edit or delete existing ones and search for tasks. It also provides access to work with task lists, reorder a task in a task list. This API will help the agent gain access to tasks and task list of the user.
- Google Contacts: Google Contacts API lets applications to access user's contact list to create new contacts, edit or delete existing ones and to search for contacts. It also lets users to manage contact groups, by assigning contacts to contact groups.

4.9.3.2 Google Maps and Directions API

Google Directions API calculates directions between two locations with an optional departure time from the origin to destination and considers traffic conditions in its calculations. This API supports driving, walking, bicycling and for some locations also supports public transit mode of transport. This API responds with the directions along with the duration of the travel.

4.9.3.3 Google Search Engine

In addition to the personal information management APIs, Google also provides access to Google Search engine thru REST API. This API can be used to query any search query from the user to the agent.

4.10 Mobile Devices

Mobile devices with a host of personal applications such as calendar, task lists, emails, address book provide rich information about the user and his activities. The location data coming from GPS sensors on mobile device provides location information of the user, which can be used as contextual information by the agent.

Choice between Android versus other mobile operating systems

Building a smart agent involves integrating with applications running on the device, as well as with semantic data sources on the web. This involves use of multiple APIs for NLP, semantic databases, external API integration that would be part of the agent running on the mobile device. Availability of these libraries on Android platform as well as easy portability of open source libraries into the Android platform makes Android a preferred mobile operating system compared to Apple's iOS or Microsoft's Windows 8 which do not have same kind of support.

Android mobile OS ³³

Android is an open source operating system managed by Google and built for mobile devices such as smart phones and tablet computers. Android enables device manufacturers to fast track mobile device development by reusing the android operating system, and its ecosystem of application framework, tools, developer community and market place.

Android is built for user interfaces based on direct manipulation - device interfaces where user's touch inputs simulate real world action are used as inputs. For example, user actions such as touch screen swiping, pinching, and tapping are used as application inputs from the user. It's designed to provide an immediate feedback to the user using vibration or audio capabilities of the device. Android devices and operating system also support sensors such as gyroscopes, accelerometers, proximity sensors, light sensors, weather related sensors that are used to customize output on the screen as well as input to the application to respond appropriately to the user context.

Android includes application framework that provides a standardized application framework in Java programming language that application developers can use to build applications supporting multiple devices that run android operating system.

Application development environment

Android application development is done using Java, which has a large community of developers. Android comes with plugins for open source Eclipse IDE which is widely adopted as integrated development environment by number of Java developers. Thus, android taps into existing set of developer resources as well as skills and leverages these skills to build specialized mobile applications on its platform.

In order to make testing and debugging easier for developers, android platform includes emulators for multiple android devices that can be used to emulate the hardware within the development environment. This enables developers to test their application faster and get an immediate feedback without waiting for running the application to run on the device itself. In addition, emulator supports multiple devices as well as multiple versions of the android platform. All the combinations of android version and devices can be tested on the emulator first before testing on the device itself.

³³ "Android OS - [http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))."

Android System Architecture

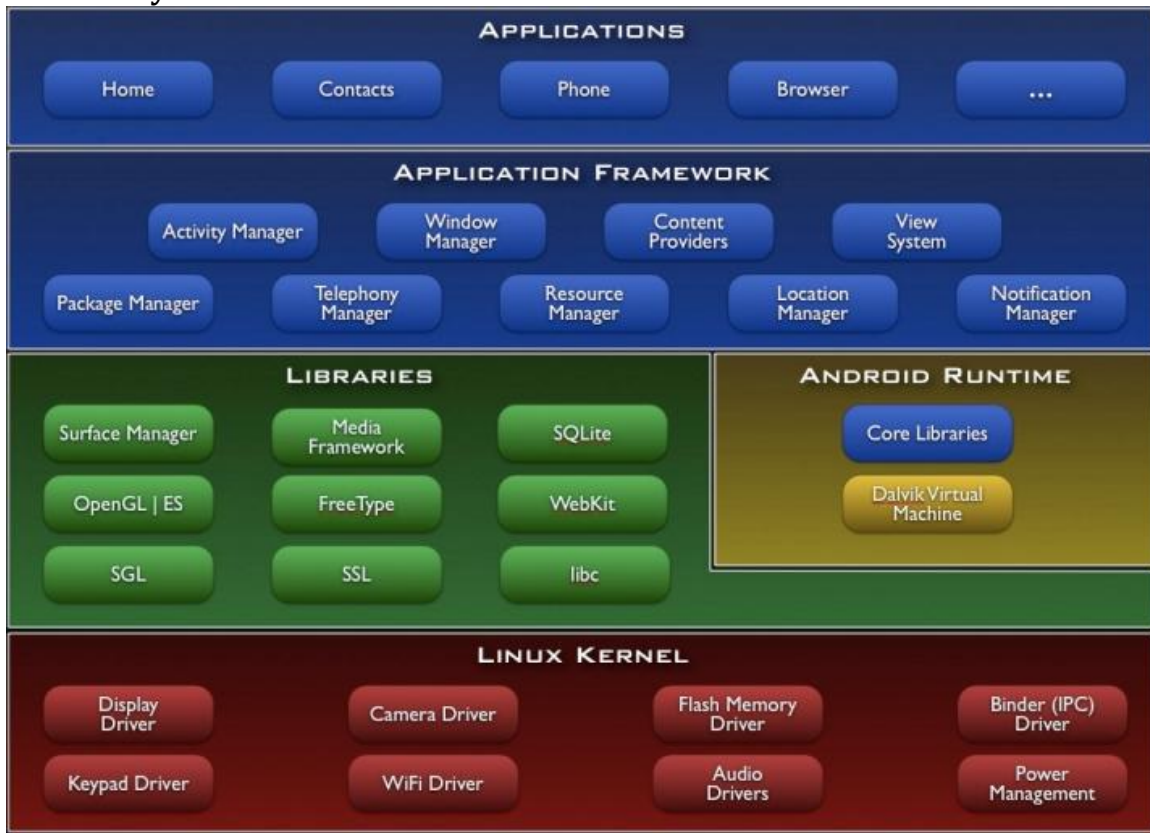


Figure 4-15: Android System Architecture³⁴

Linux Kernel:

Android is based on open source Linux operating system kernel version 2.6 at the core, is written in C and drives the device hardware. It provides core operating system services such as file management, memory management, process management, power management etc., with hooks to add device specific hardware drivers. Hardware device manufacturers can take advantage of this open source operating system by building their hardware specific drivers and supplementing android operating system. Even though it's based on Linux operating system kernel, some of the libraries that are available on Linux are not available on Android, making it difficult to port some of the existing Linux applications directly into Android, since underlying dependent libraries are not yet ported.

Android Libraries:

Android libraries layer consists of libraries such as graphics rendering engine, media framework to encode/decode file types, secure sockets layer, SQLite database etc., all optimized for the native hardware of the device.

³⁴ "Android System Architecture Diagram - [Http://developer.android.com/images/system-architecture.jpg](http://developer.android.com/images/system-architecture.jpg)."

Android Runtime:

Android applications are programmed in Java language, compiled to Java bytecode and then converted to an optimized bytecode called Dalvik bytecode. Dalvik bytecode is an optimized bytecode for android OS. It is primarily designed to make the application code compact in order to conserve space and runtime CPU time on power constrained mobile devices. Dalvik runtime uses direct register addressing for accessing variable values compared to using stack to store and retrieve variable values in Java and other stack based systems. This enables Dalvik runtime to execute bytecode faster, within the constraints of mobile device.

Application Framework:

Application framework provides classes and design patterns to build applications for android. It is written in Java and provides abstraction to underlying layers of the operating system.

Applications:

Application layer consists of applications built using the Android application framework, and deployed on the device. These can be applications bundled with Android, device manufacturer applications or third party applications deployed by the user of the device.

Relevance to the Personal Assistant software:

Personal assistant software needs access to the user in order to be used for routine tasks. Designing it as a mobile application achieves this purpose, while giving application access to all the personal information available on the device thru the APIs.

5 Designing Smart Agent

5.1 Semantic Web as a building block

Semantic web is envisioned as a single logical web of meaning that is interconnected between various sites and has self-defined meta-data so that any one willing to use this data can interpret it and extend it. It makes the data machine-readable and using semantic web's inference technologies, tools can be built to process this data and build applications that can interpret the data.

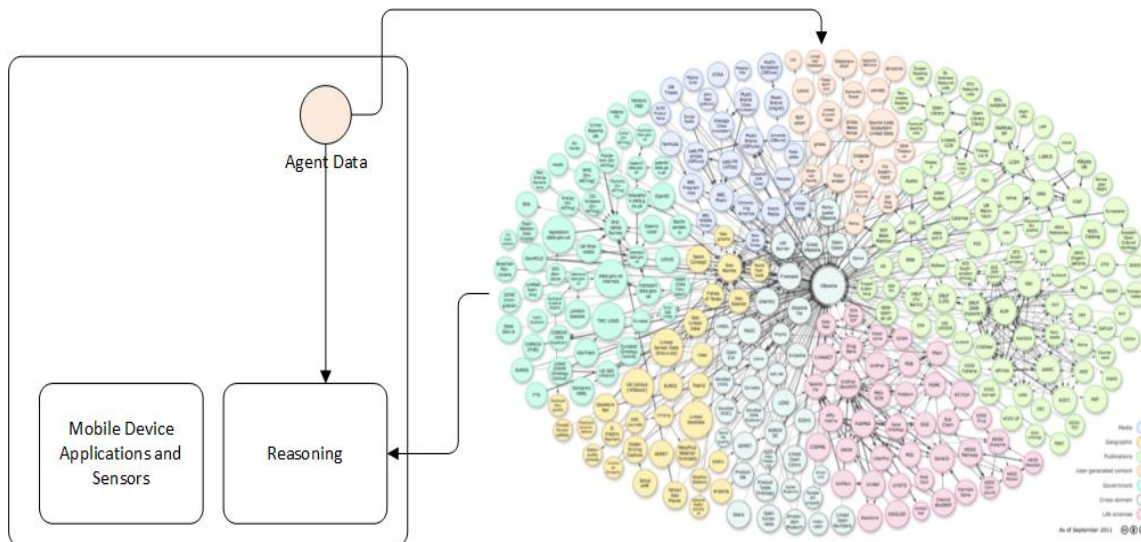


Figure 5-1: Integrating agent with the web of linked data ³⁵

Conceptually, smart planning agent is designed to work with existing web of linked data, leveraging the ontologies as well as the data that is being collected in these web sites using linked data technologies such as RDF and OWL. It reuses the ontologies defined by sites such as OpenCyc, GoodRelations, Wordnet, DBpedia etc., and in doing so can retrieve vast number of facts related to day to day activities collected by OpenCyc and process them using their interconnected links to other ontologies such as Wordnet, DBpedia etc.

Use of semantic data brings out best practices in managing data and enables data reuse by having a single reference to data entities and creates a single unified data that can be extended by application providers by defining new ontologies using the base ontology of the data and adding new attributes on it.

A reasoning engine uses the information obtained thru linked data, applying the contextual information of the user that is available on the mobile device and make inferences as well as plans for the user.

³⁵ "Linked Open Data Cloud - [Http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.png](http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.png)."

Agent stores inputs as well as outputs from these interactions in a local RDF database that can be used in later interactions as a historical data about the user profile. In addition, this local data store can extend ontologies defined on the web by adding attributes and relations more relevant to the agent context. This enables a single set of semantic technologies to be used to query knowledge databases on the web as well as historical data and facts collected in the local database.

5.2 Designing Smart Agent

5.2.1 Requirement Analysis

- i. **Interpreting user input** – usability of the agent is one of the high priority requirements for a smart agent application. It needs to be more helpful to the user and ask for fewer inputs. In addition, it needs to support input in regular English language.
 1. Supporting voice recognition: Ideally, a smart agent needs to support voice interface in order to make the agent useful while the user is unable to type the request on the device. But, this requirement was purposely out of scope in order to focus more on the other aspects of building such an application while assuming that agent can later integrate with a voice recognition technology which can convert spoken text into textual commands for the agent to process.
 2. User input should be processed to identify the task to be performed. Define templates for the supported tasks. These templates should help the user to type required information for the task so that agent can identify the task and related parameters.
- ii. **Understanding the task at hand** – Traditional task list applications treat the tasks as strings and do not associate any semantics to the task. The smart agent should exhibit an understanding of the task entered thru the system. This is a critical requirement as it enables the agent to use this understanding of the task to plan for the task.
- iii. **Contextual awareness** – Identifying the context of the user for any given task enables the agent to better serve the user's intended goal and possibly asks fewer questions to plan for the task. There are multiple contexts associated with an agent
 1. **Location Context:** This context provides current location as the input and lets agent reduce the search path for a given task. This location context refers to low level geo coordinates offered by GPS sensors in mobile devices as well as mapping these co-ordinates to higher level entities such as home, office, airport, mall etc. Location context information coupled with understanding of the task helps the agent to identify type of tasks that can be or cannot be accomplished at this location.

2. **Time Context:** Time context for an agent includes an understanding of common time terminologies such as morning, afternoon, evening, today, yesterday, tomorrow etc., and using current time or time of the task to filter tasks relevant in this context.
 3. **Activity Context:** An activity context refers to the understanding of an ongoing activity or a planned activity and identifying attributes of the task that need to be managed. For example, if you are executing a task of driving to the airport to catch a flight, then an understanding of catching the flight on time is important. At the same time, generating alerts to the user if the flight is delayed is very useful and utilizes activity contexts.
 4. **Social Context:** This refers to understanding of social interactions between the user and other people in his network. With this contextual information, agent will be able to service requests based on a specific relation mentioned in the task and will be able to use information on people accompanying the user in an activity to suggest sub-tasks for a task or an activity.
- iv. **Understanding external environment context:** Agent needs to plan for tasks that interact with the external world. So, any changes in external world such as weather conditions, traffic conditions, delays in other people schedules should be coordinated and provided as input to the agent for planning the execution of tasks.
 - v. **Task decomposition:** Agent should capture task models that include specifying a given task by decomposed steps that contain sub-tasks, which in itself can be planned by the agent. In this task model, agent should be able to specify ordering of the subtasks, any dependencies between the sub-tasks, pre-conditions as well as post conditions of completing the sub-tasks. It needs to provide preset task decompositions for supported tasks and let the user customize some aspects of these sub-tasks. Part of this task decomposition is driven by the extent of understanding of the task within the agent's knowledgebase, so extensibility of the knowledge base will drive adaptability of the agent to new tasks. Agent should provide a framework to capture sub-tasks for a given task by getting input from the user.
 - vi. **Planning tasks:** Agent should be ask questions to the user in order to disambiguate any input related to the tasks and to identify specific task in its repository and also to identify the subtasks. Once its done with this process, it should analyze sub-tasks involved in the process, sequence of these tasks, any dependencies, pre-conditions and post conditions and create a plan for executing these tasks either manual thru the user or some other person or automatically thru the agent.

- vii. **Automating task execution:** Agent should be able to identify the tasks or subtasks that can be accomplished by the agent itself by automatically invoking modules within the agent capable of completing the task or sub-task. These automated tasks could be for gathering information from variety of data sources the agent is capable of connecting to, or to invoke APIs to book an order for an item at online e-commerce store.
- viii. **Delegating tasks:** Each task will be either a manual task or an automated one. Agent should be able to identify the type of the task and execute automated tasks with the information entered by the user and using contextual information of the user within in the agent. Agent should be able to identify number of actors for manual tasks and should try to identify if the user or the agent can be the default executioner of the task. If it's a manual task, agent should identify if the user generally completes it or delegates to others and use this information to assign a default delegate.
- ix. **Record User Interactions:** Agent should record user task entries, location, time, and user selections from list of choices, in a database. This information can be used to train the agent to identify paths that user is more likely going to take based on the past historical data.

5.3 General Architectural and Design Principles ³⁶

5.3.1 Modularity

A system with modular architecture can be decomposed into well-defined subsystems. A modular subsystem with well-defined interfaces explicitly specifies expected inputs and outputs of the system and wraps all the complexity associated with implementation internally and hides it from the subsystems interfacing to it. This also reduces coupling between the subsystems.

A modular architecture leads to higher reuse of existing components, adaptability to change, shorter time to market, scalability of product design and reliable testing cycles.

5.3.2 Extensibility

System should designed such that new features can be added to the system as well as existing features can be modified without a major overhaul to the architecture and ideally by few simple changes. Modular architecture and loose coupling between sub-systems plays a major role in ensuring an extensible system.

³⁶ "ESD.34 System Architecture Course at MIT – Principles of Architecture Assignment."

5.3.3 Complexity

A good architecture takes into consideration performance and quality attributes of the system and also considers all the -ilities such as extensibility, modularity, scalability, availability etc., expected of the system. Complexity of the system is thus driven by the essential functionality that need to be implemented to satisfy essential user needs and by these performance and quality attributes that need to be supported by the system. A good architecture will be able to deliver on all these promises by choosing concepts with low essential complexity and by using abstraction, decomposition, hierarchy and recursion to keep the actual complexity to the essential complexity.

5.3.4 Falling back to the user in case of ambiguity or unknowns

System should be architected such that in case system is subjected unsupported cases then it should respond gracefully, and in case of personal assistant, should fall back on user for clarifications or choose from available choices. This introduces human component back into the system, reducing complexity that would be otherwise required for the system.

5.3.5 Adopting Open Standards and Data Reuse

A system design that promotes adoption of mature open standards enables quick development and deployment as it leads to sharing of the technology, tools as well as other resources developed by the community.

Adoption of semantic web technologies, tools, and ontologies relevant to the smart agent makes the implementation of the agent easier thru reuse.

5.3.6 Integrating with external data sources and Services

An architecture that offloads functionality to external components such as web services or external data sources simplifies the design of the system itself at the same time leveraging the expertise of the external systems. It also introduces dependencies on these external systems and fallback mechanisms should be added in order to use alternative paths when one of the dependent paths is not available.

5.3.7 Early Detection

Instead of building the whole system at once, iterative development approach insists on breaking up the release artifacts into manageable elements such that each phase results in delivery useable and higher priority features to the stakeholders. Design, development and testing cycles are executed for each phase and any issues with implementation are found during these cycles. User feedback is also ascertained to ensure that features are being implemented as per the user needs and intended goals of the system. This feedback provides any correction in the implementation or re-prioritization of the goals of the system.

5.4 Abstract Models in the agent

5.4.1 Task templates

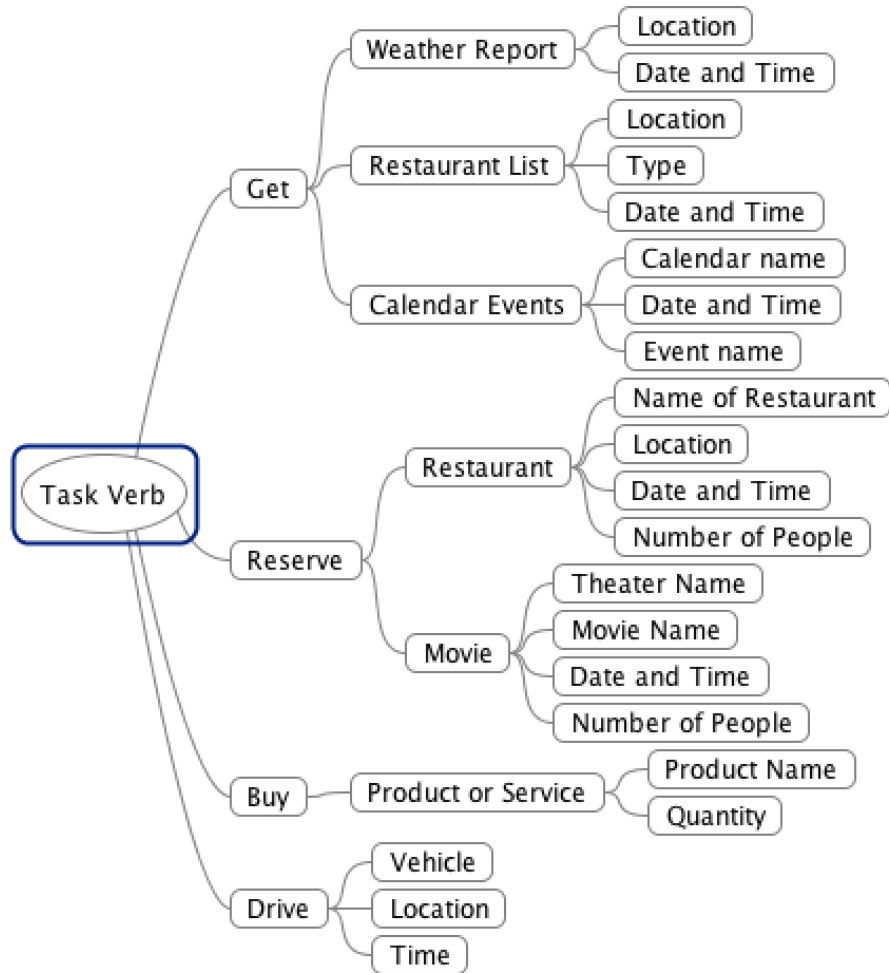


Figure 5-2: Sample Tasks using Task Template

Input to agent is driven by a set of templates that detect the pattern in the input text and ask the user to enter specific fields. Template is driven by data organized in a tree structure shown in Figure 5-2 above. It begins with a verb for the activity, and then based on the selected verb, subsequent parameters are presented to the user for entry.

Following is a set of sample verbs supported by the agent:

1. **Get** is an information gathering activity that can be executed by the agent by using web services or linked data sources. Agent then asks for specific information needed by the user, and asks the user to select from a dropdown list which has items such as 'weather', 'restaurant list', 'restaurant list – Indian' etc.

2. **Reserve** is a verb used to identify online reservation tasks that can be automatically executed by the agent. For example, reserving a specific restaurant for a group of people using web services such as OpenTable to book restaurants.
3. **Buy** is a verb that can be used to capture general day-to-day buying activities related to buying groceries, books, electronics etc., that involve shopping at a retailer store.
4. **Drive** is a verb that can be used with tasks related to driving to particular locations. Agent will follow up with questions on vehicle, location to drive to and when the user wants to start the driving activity.

Input System, uses the first word entered by the user to detect the pattern, and dynamically updates the UI to capture rest of the input parameters for the task.

Hierarchy of nodes in the task template is used to follow a naming convention to capture attributes of the task, as well as to capture input and output parameters of the task. For example, 'Reserve restaurant' task is identified by the name Reserve.Restaurant and has input parameters Name of the restaurant, location of the restaurant, date and time of reservation and number of people going to restaurant. Web service manager module will use the task template name to capture integration details of the task with the web services that offer capability to complete this task.

5.4.2 Web Service Integration

Agent uses web services to query for information as well as to execute actions by calling the external web services offered by third parties thru the web service manager module. There are two types of web services agent integrates with – informational web services, which are used to gather information given a set of input parameters, and another set of web services to execute actions on behalf of the user.

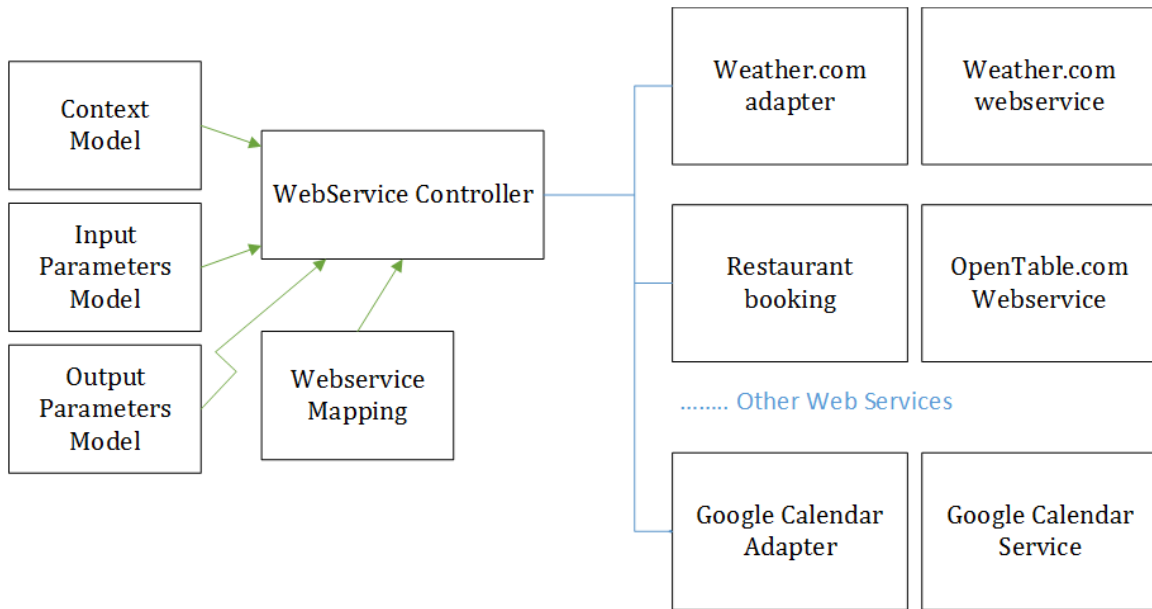


Figure 5-3: Web Services Manager

Figure 5-3 shows sub-systems involved in integrating agent with web services and forms Web services Manager module.

Web Services controller is the central component that orchestrates web service integration within the agent. It takes inputs from Input Parameters Model as well as from Context Model and then uses Web service mapping module to identify which web service to call, and also what parameters to be passed. It will then call web service adapter with the input parameters, and parse the response.

Context Model contains contextual information about the user such as user's current location, current time, calendar appointments, other tasks currently in progress, past history of tasks, user interests, and likes. This model provides input to the web services module, parameters that are not explicitly entered by the user, but are relevant for the web service.

Input Parameters model is a generic model that stores all the information collected by the agent from the user. In cases of use cases such as 'Get weather report for a location' – this would include location name, date and time of the report.

Output Parameters model captures expected responses from the web services, format of this output. For example, weather service will respond with weather conditions for a location such as temperature, humidity, wind, sunrise/sunset timings, etc.

Web Service Adapters handle connectivity to specific web services, passing input parameters in the desired form and transforming the output from the web service in the desired form for the agent. In addition, it will also translate any data elements

between the two sides so that each side understands the other. Some examples of these adapters include Google Calendar adapter, Google Contacts web service adapter, Weather.com web service adapter, OpenTable web service adapter etc.

Web Service Mapping module, will map a task to a web service and the input parameters to specific parameters accepted by that web service. This module, in conjunction with specific web service adapters enables invocation of external web services from the agent. It can be conceived that, support for new web service can be enabled by adding a new adapter to the web service and then adding a new mapping information regarding the service and its input/output parameters.

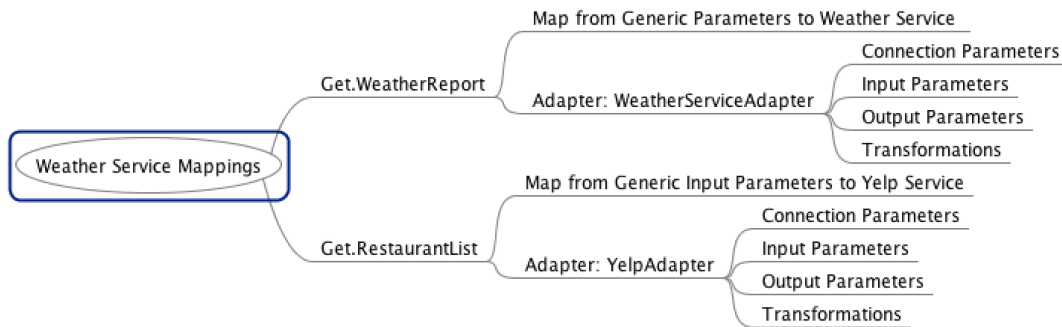


Figure 5-4: Web Service Mapping Example

Figure 5-4 shows how web service mappings are arranged for each of the supported tasks. It contains mapping of generic parameters such as location, date and time captured in the task template model to specific parameters expected by the web service that needs to be invoked by the agent. It also contains details about web service adapter that needs to be called along with parameters that need to be passed for input as well as parameters that are expected as output.

These mappings enable the agent to bind generic data gathered in the UI to specifics of the web service on the input side, and transform the response from the web service to display output to the user.

5.4.3 Task Model

Task Model is used to capture information related to the task such as task name, actor, start time, end time, duration of task, urgency, importance, dependencies with other tasks, sub-tasks, and status of the task.

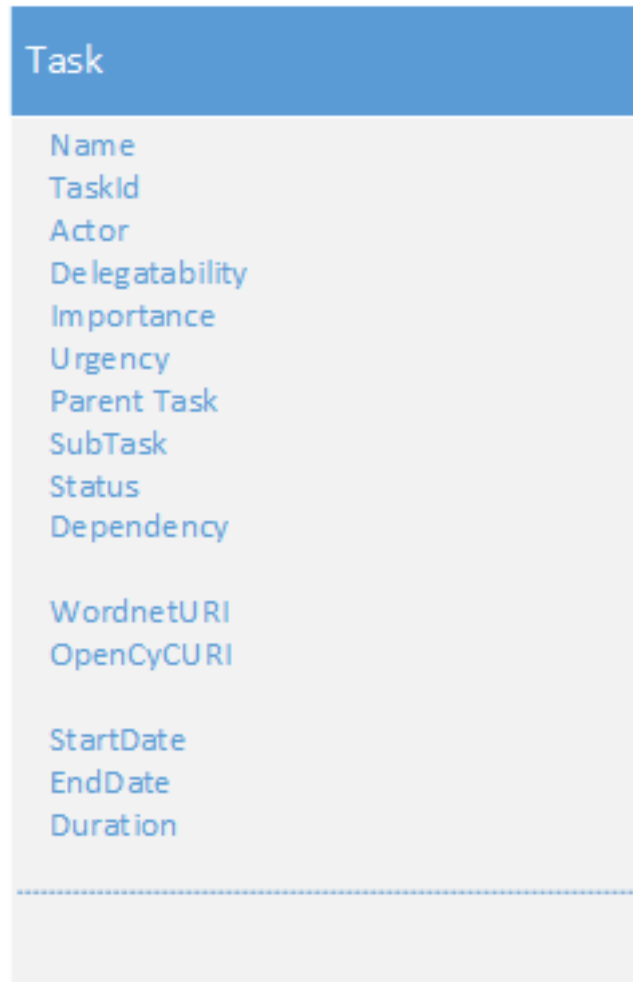


Figure 5-5: Task Model

Task Model is used to capture details of the task, its sub tasks, dependencies with other tasks, status of the task. It also includes information on importance and urgency of the task used to prioritize the task during planning stage. Actor and Delegatability fields enable the agent to identify who this task can be assigned to. It also includes references to Wordnet and OpenCyc to gather additional information on the task. Knowledge databases provide useful information on sub-tasks associated with the task that are then form part of this task model.

5.5 Modules within the system

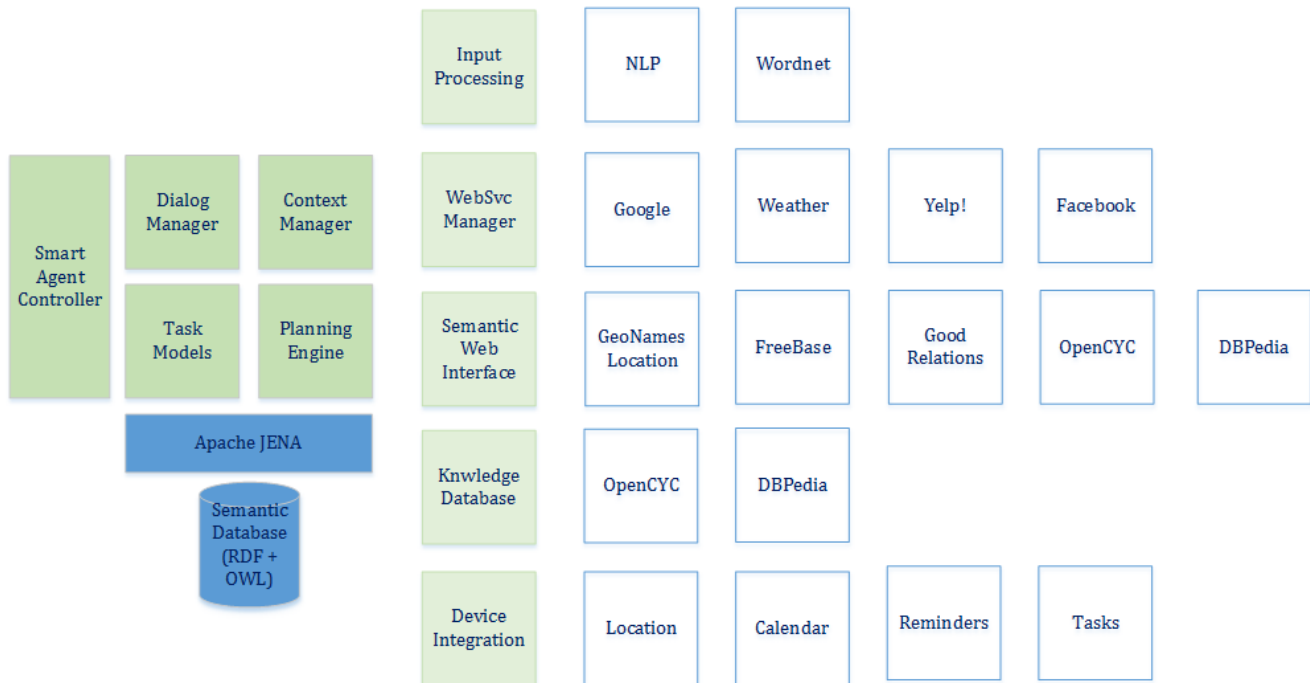


Figure 5-6: Modules within Smart Agent system

This section describes modules or sub-systems within the smart agent.

Smart agent consists of five layers of sub-systems, with each layer interfacing with a specific set of technologies and providing a high level functionality to the agent. Input Processing, Web Service Manager, Semantic Websearch, Knowledge Search and Device integration are the five layers and the modules shown in green are the facades to connect to external entities shown in blocks without any color. Agent modules such as Dialog Manger, Task Definitions models, Context Manager and Planning agent use facades to each of these layers to connect to individual external entities for data.

Following section describes each of these five layers in detail and then describes the agent modules that interact with these five layers to provide agent functionality.

5.5.1 Five Layers for connecting to external systems

1. Input Processing Layer



Figure 5-7: Input Processing Layer

This layer deals with parsing all the text entered thru the GUI using NLP libraries. It uses task templates defined in the previous section to gather task information from the user. These templates are designed to ask users to select an activity and then gather relevant information for the activity. Relevant information needed for the activity is coded into the templates. This enables the agent to ensure required parameters for the activity are gathered by the agent.

Natural Language processing toolkits like OpenNLP that understand English grammar are used to parse text into parts of speech for the text entered by the user. This parsed information will be tagged with parts of speech information that is used to map entered text to task that can be planned by the agent. In addition Wordnet is being used to find synonyms for words that don't have direct mapping in agent dictionary.

2. Web Services Integration layer



Figure 5-8: Web Services Manager Layer

This layer integrates with web services to support user queries to specialized data or service providers. These services wrap the business services to query weather at a location, search for a business on Yelp, or issue a search query to a search engine thru the web service APIs provided by companies such as weather.com, Google, Yelp and Facebook. This module deals with all the low level details of connecting agent into information sources.

This layer gives access to external environment conditions of the user that are needed while planning tasks. Some of these environmental factors include

weather conditions in the locations related to the task at hand, traffic conditions if a task includes driving to a location etc.

3. Knowledge Database Search layer

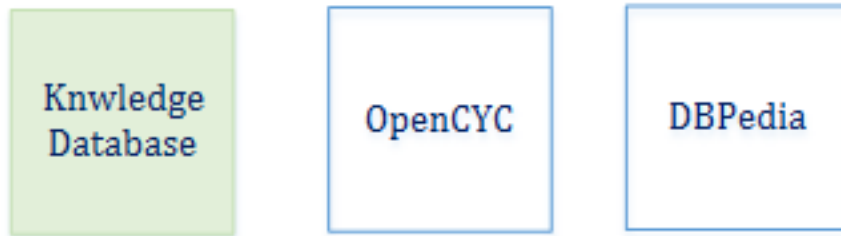


Figure 5-9: Knowledge Search Layer

This layer interacts with knowledge database OpenCyc to query and update the knowledge concepts and assertions within this knowledge database. It also searches for concepts in DBpedia.

Knowledge Databases such as OpenCyc and OpenMind store vast amounts of common sense data and are available thru API in RDF form. This information is used to understand the task and dependencies it may have on the environment to be executed.

Each agent will have access to a dedicated copy of OpenCyc instance to store user specific information.

4. Semantic Web Interface Layer

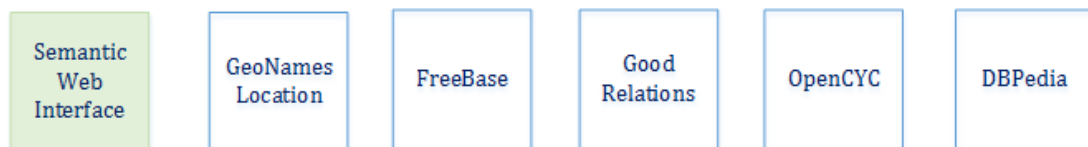


Figure 5-10: Semantic Web Interface Layer

Semantic web interface layer connects agent to external RDF data sources such as GeoNames, GoodRelations as well as RDF based knowledge bases OpenCyc and DBpedia.

GoodRelations ontology is used to query ecommerce sites for specific products and product availability, enables linking the concept of a product in a knowledge database with real time availability and pricing information on the ecommerce sites.

Consistent use of semantic web technologies across different components used by the agent enables the agent to easily interpret any new data that is exposed using these technologies. User profile ontologies like FOAF and FB open graph are being evaluated to represent user and user's social network connections.

Wordnet already links into concepts defined in OpenCyc. This enables agent to parse the task parameters and then link task into Wordnet and then into OpenCyc concepts using the pre-defined links between Wordnet and OpenCyc.

Following set of ontologies are currently used by the agent:

- OpenCyc
- DBpedia
- Wordnet
- GoodRelations
- GeoNames
- FOAF

5. Device Integration Layer

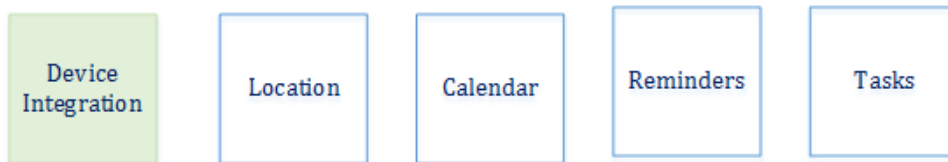


Figure 5-11: Device Integration Layer

This layer interacts with the device operating system as well as personal information management applications thru the APIs exposed by the platform. Mobile apps such as calendar, reminders and notes are used to gather user's specific environment information along with location data from the GPS sensors.

This layer exposes internal environment of the user and is accessed thru sensors and applications on the mobile device. This enables agent to read user's current location from the GPS sensors, access user's scheduled meetings from the calendar, and access user's contacts thru address book.

5.5.2 Agent modules orchestrating interactions with five integration layers

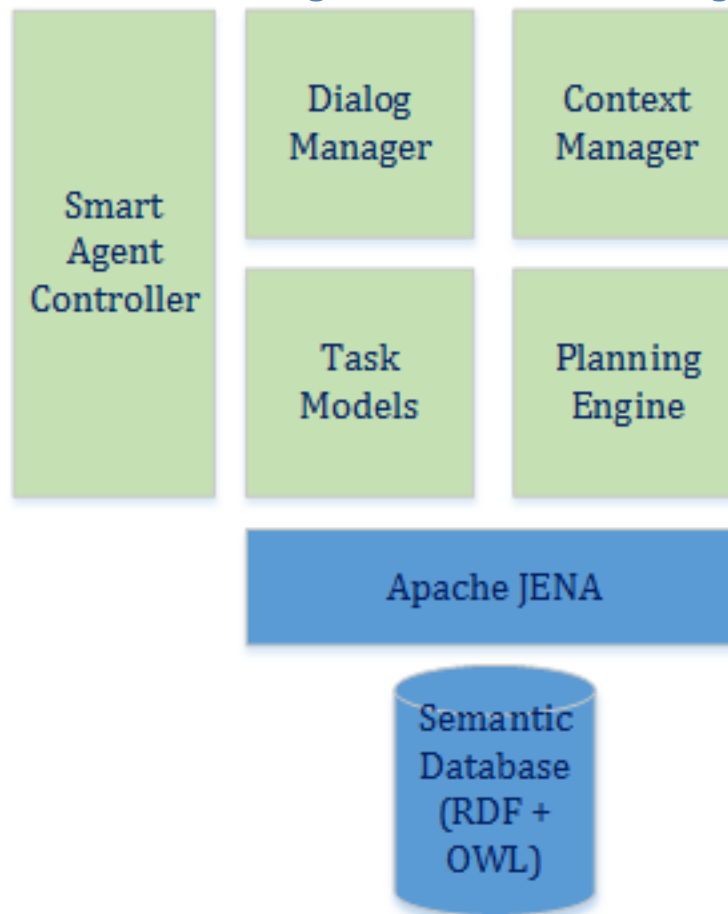


Figure 5-12: Agent modules interacting with 5 integration layers

Apache JENA framework is used to interface with RDF data stores, for SPARQL queries. It also includes an inference engine that can draw inferences on semantic data using OWL ontologies. It also includes an RDF data store along with the framework that can be used to store task related data gathered from the user in RDF form.

1. **Smart Agent Controller** orchestrates interactions between the user and other modules within the agent. It operates as a central module making decisions on which modules to be called during the user data gathering process, integrating with five layers of the agent, invoking relevant task models, getting the right contextual information from the context manager and using the planning engine to plan the task for the user.
2. **Dialog Manager** – will present user with multiple choices where agent is not able to make a decision on any specific path to be taken for a plan. For example, if the user has a task for going to airport and there are multiple airports around

the user's current location, then agent will present the user available choices and record the selection.

3. **Context Manager**

- a. Time Management module will try to optimize tasks by grouping related tasks together and by finding optimum time for executing tasks
- b. Location Management module uses location of the user to optimize and group any other activities of interest at the same or nearby location, thus optimizing the planning paths by location.

4. **Task Models** - will decompose a given task into multiple steps of sufficient granularity. It uses dialog manager to gather input on any tasks that agent is unable to decompose into multiple steps and needs user intervention. This module will add new knowledge of task decomposition into the knowledge database and will extend the knowledge repository. These models connect the task definitions with the services available at the five layers described above in order to execute the actions of a sub-task.

5. **Planning Engine** is the core part of the agent that interfaces with other modules described here to collect the information related to a given task user wants to execute. It applies the rules on the information collected within the supported domains and builds a graph of possible scenarios. If the number of paths to choose from is greater than one, it will use the dialog manager to ask the user to select a particular path.

Inference and Planning is implemented using Apache Jena components. These components enable the agent to work with semantic web technologies and support defining inference thru a customizable rules interface. This component will begin with the understanding of domains related to initial use cases defined in the design chapter of the thesis. For example, it will have information on how to execute some of the automated tasks thru API calls to external service providers such as weather.com or Google.

5.6 Use cases

Agent will be a smartphone application that has access to GPS sensors, calendar application, and has a user interface for the user to interact with. GUI will be used to explicitly enter tasks of the user – that are to be achieved immediately or later by the user or the agent. GUI provides templates for the task entry and along with NLP tools will help the agent understand specific task that is entered in its goal to plan the task for the user.

5.6.1 Use case 1: Agent is requested to get weather report for my parent's hometown

This use case is to highlight tasks that can be automated thru the agent and also how it integrates with internal as well as external RDF data stores. In a way, agent can

gather information online using the data sources it's integrated with and service the request.

- a. User chooses "Get" activity from the UI template. Get activity relates to information gathering tasks that agent can automatically perform for the user. UI will then load list of items it supports - for example 'weather report', 'restaurant list', 'stock quotes' etc.
- b. User will select 'weather report' from the list of available services for the agent.
- c. Agent will then ask the user to enter location. Current location is the default location.
 - a. This is a text field, and in this case user enters 'parent's hometown' as input text.
 - b. Input processing system will parse the text, and identify parent as a relationship in Wordnet and then use contacts interface to fetch parent->hometown information. If that is not found, Wordnet provides father and mother as alternatives. Input system will then use father->hometown or mother->hometown to gather hometown information.
- d. Agent will then ask the user to enter date and time for which weather report is being requested. Default is today.
- e. Input module that takes these inputs from the user passes user entries for further processing.
- f. Agent uses webservice manager to execute this task, as task template identifies it as a task supported by webservice manager based on the user selection. Web service manager identifies web service adapter to be called, required input parameters, and calls the adapter, which in turn invokes the web service.
- g. Agent displays the weather report in the UI. By default, the weather report returns current weather conditions, and high/low temperatures for today as shown below. Adapter can define output transformations to transform the data obtained from the web service into different formats.



Figure 5-13: Sample report for weather request

5.7 Sequence Diagram & Data Flow during execution of task

This section covers sequence diagrams and data flow diagrams for use case 1 – ‘get weather report for my parent’s location’, which is an automated use case where smart agent is accomplish the task with out any user interventions after the user input. This use case is divided into tow parts – first part deals with getting user input for the task and the second part deals with executing the task.

5.7.1 Task Input

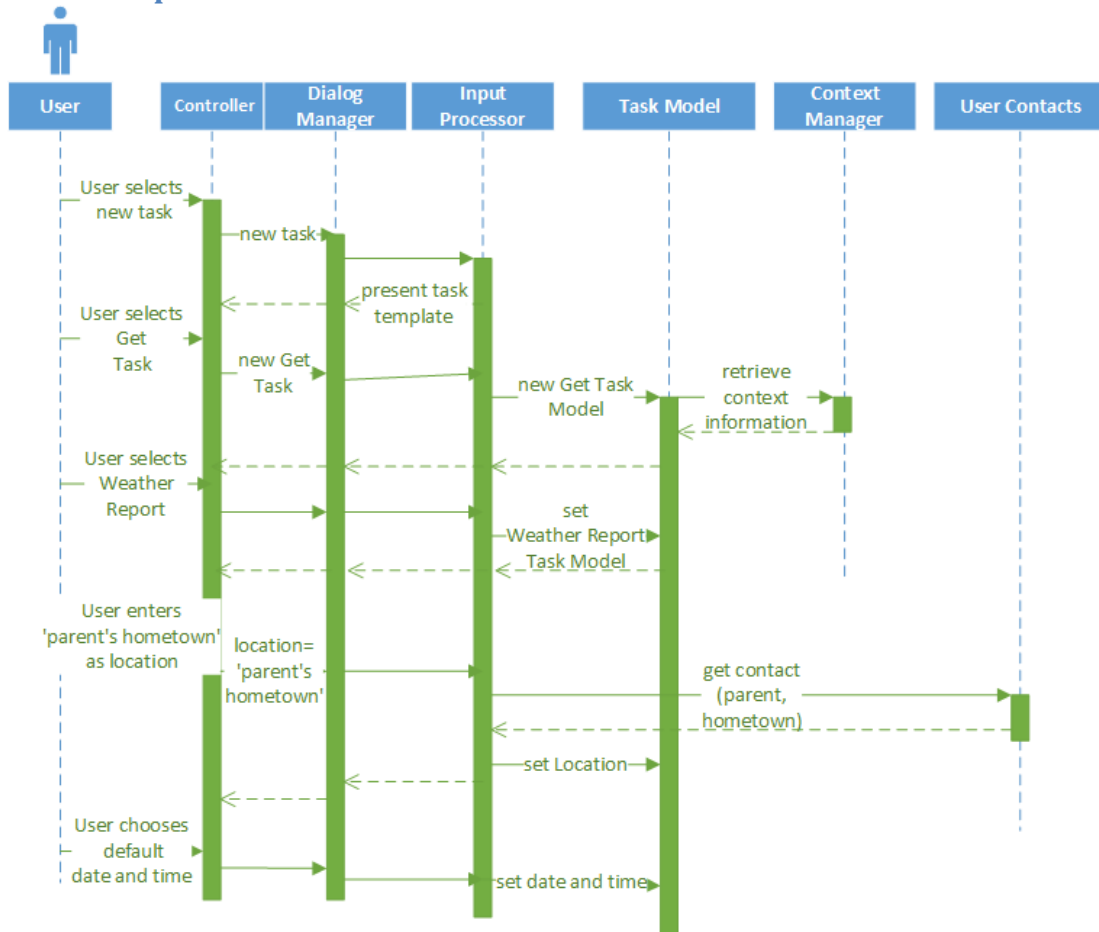


Figure 5-14: Sequence diagram of input system for use case-1

Sequence diagram in Figure 5-14 depicts interactions between different modules within smart agent for this use case during the task input process. It involves Dialog Manager, Input Processor, Task Models, Context Manager and User contacts modules within the agent system.

1. User begins with asking the agent to create a new task
2. Agent presents supported task list such as 'Get', 'Reserve', 'Buy' and 'Drive'
3. User chooses 'Get' as the task. This triggers input system to create a new task model instance which supports 'Get' type of tasks
4. Task Model then gets context information regarding the user from the context manager. This information will include user's current location, and current activity.
5. Dialog Manager, based on the selected task model asks user to select the service from the list of supported web services.
6. User selects 'Weather report'.

7. Task Model for weather report is selected by the Input processing system and this will in turn drive the input parameters that are needed to be collected from the user. This task needs user to enter location as well as date and time of the weather report.
8. User enters location as 'parent's hometown' as input text.
9. Input processor parses this text, and identifies parent as a relation and hometown as type of address. It then queries User's contacts to get the name of the location for the contact person who is the parent of the user, and finds the hometown. For this step, its assumed that user has populated his contact database with his parent's contact information, including the hometown.
10. User then chooses default value of today as the date and time input for the task.
11. Agent now has an instance of instance of a task model of type that supports 'Get Weather report' and has input parameters - location and date/time populated.

5.7.2 Task Execution

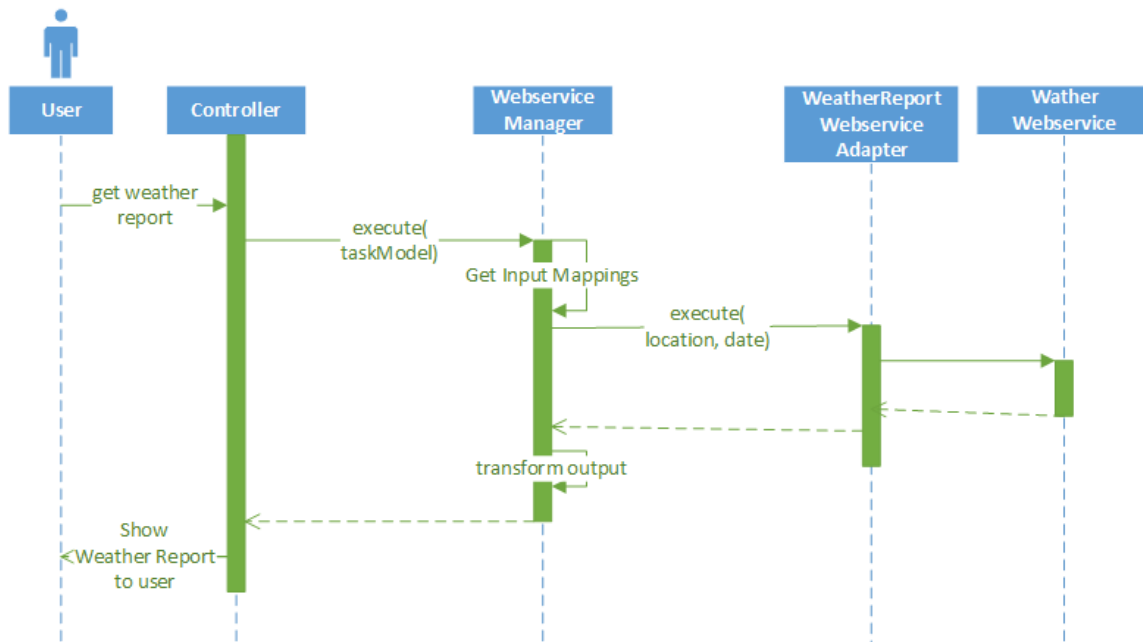


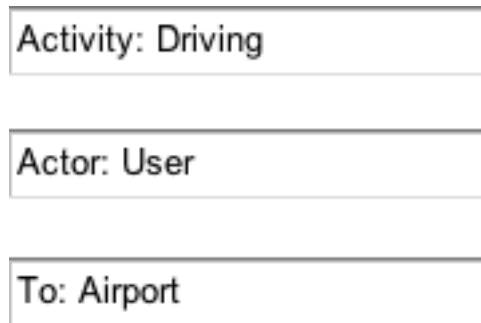
Figure 5-15: Sequence diagram for task execution in usecase-1

Sequence diagram in the figure above depicts sequence of steps involved in using the input parameters captured in the input step to invoke the web service, to automatically execute tasks on behalf of the user.

Web service manager uses Web service mapping module to identify specific web service adapter that needs to be invoked to execute this task. This mapping module also contains details on mapping input parameters collected in the task model to inputs required by the specific web service adapter. Web service manager calls the web service mapping module and then calls specific instance of web service adapter that supports this task – in this case its Weather Web service adapter, and passes required input parameters. Details on how to connect and invoke the web service are part of the adapter. Output from the adapter is then transformed and then sent to the user. Agent then displays the output to the user.

5.7.3 Use case 2: Planning the task – Drive to Airport

1. User chooses the task as Driving
2. Agent asks the user to enter destination location. User enters Airport as the location.
3. Agent parses the input and identifies Airport as an entity. From the task model it also knows that driving is the activity for the task that needs a person as an actor. By default, it chooses user as the actor.



The figure shows a task entry screen with three input fields. The first field is labeled 'Activity: Driving', the second is labeled 'Actor: User', and the third is labeled 'To: Airport'. Each field is a simple rectangular box with a thin border.

Figure 5-16: Task Entry screen for 'Driving to Airport'

4. Agent uses DriveDomainModel to drive the user inputs for this task. It also identifies User needs to drive a vehicle to get to the airport. It asks the user to select from a possible list of vehicles it has stored in its database that he will be driving. It also, searches for nearby airports and presents the list of airports to select from. This list of inputs is predefined while creating the task template.

Activity: Driving

To: Airport

When: 4pm Today

Actor: User

Choose Vehicle: My Car ▼

Choose Airport: Logan ▼

Figure 5-17: Task Entry screen to enter time and choose airport

5. User chooses the airport and the vehicle to drive. At this point, task is well defined. Agent runs its planning algorithm and identifies set of tasks to reach the airport by 4pm.

Activity: Driving

To: Airport

When: 4pm Today

Pick up the car at 3pm

Route based on traffic

Figure 5-18: Agent's plan for the task - 'Driving to Airport'

During the planning process, agent is able to identify in order to drive to the airport in the car

- user needs to be near the car and user needs to pick up the car at 3pm in order to reach airport at 4pm

5.8 Sequence Diagrams for use case 2: Drive to airport

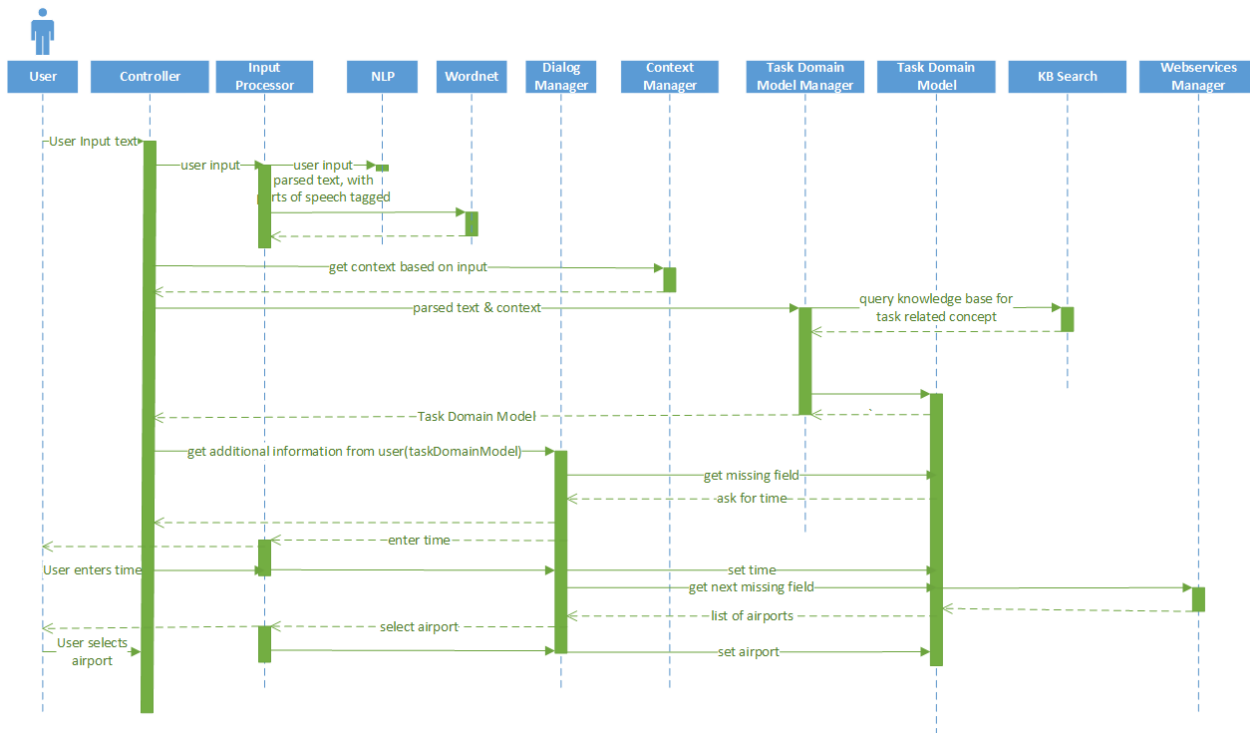


Figure 5-19: Sequence diagram for input system in usecase-2

This section covers sequence of steps executed by the agent in order to parse, map and plan a task that is to be carried out manually.

5.8.1 Parsing the input

User selects 'Driving' as the task template and enters Airport as the location. Agent parses the location and identifies Airport as noun.

Input system then queries Wordnet to identify the relevant word in Wordnet to concept in OpenCyc. Conceptually, this is the step when the text input is being mapped to semantic world. This stage has challenges related to ambiguity in input, but for the sake of simplicity as well as since NLP is not the core to this thesis, its assumed that agent will be able to map the input text to words in Wordnet.

In order to make this translation, word sense within Wordnet needs to be constrained to transportation domain. Task Model that is used to handle drive task – DriveDomainModel is setup to use following SPARQL query to identify the Wordnet ID corresponding to word 'drive'. It can then be used to link to OpenCyc concept on driving.

SPARQL Queries used for identifying Wordnet words corresponding to drive:

```

PREFIX id: <http://wordnet.rkbexplorer.com/id/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX akts: <http://www.aktors.org/ontology/support#>
PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>

```

```

SELECT * WHERE {
  ?s rdfs:label "drive" .
  ?s wn:hyponymOf ?o .
  ?o rdfs:label "transportation" .
} LIMIT 1000

```

SPARQL query to identify word within wordnet corresponding to airport.

```

PREFIX id: <http://wordnet.rkbexplorer.com/id/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX akts: <http://www.aktors.org/ontology/support#>
PREFIX wn: <http://www.w3.org/2006/03/wn/wn20/schema/>

```

```

SELECT * WHERE {
  ?s rdfs:label "airport" .
  ?s wn:hyponymOf ?o .
  ?o rdfs:label ?so .
} LIMIT 1000

```

This returns with the ID of the word 'airport' that can be referenced from OpenCyc.

5.8.2 Identify the task domain model

Task domain models are linked to microtheories within OpenCyc, which captures domain knowledge relevant for a domain. This domain knowledge helps the agent to identify pre-conditions and sub-tasks necessary for planning a task.

OpenCyc concepts contain mappings for Wordnet words. So, Wordnet URI can be used to identify the corresponding concept in OpenCyc. In this case the mapped concept in OpenCyc is **TransportInvolvingADriver**.

At this stage, agent uses the information from input system and context manager to identify the task model that will be able to handle the input task of the user.

A task model is intended to capture task information, query the knowledge database for sub-tasks, pre-conditions, and post-conditions as well as to keep a mapping of how to execute the automated tasks or map manual tasks to relevant concepts in knowledge base.

In this use case, a DriveDomainModel object will be activated. This domain model is integrated with web services linked to finding weather and traffic information that is relevant in planning the driving activity.

5.8.3 Create a new task object

Explicit information gathered from the user as well as deduced information is used to create a task object which captures all relevant references to other objects for the agent.

This task object contains following details:

- Name of the task
- Importance of the task
- Urgency of the task
- Due date
- Delegatability (inferred by the agent)
- Who will perform the task
- When will this task be performed
- Where will this task be performed
- Subtasks for this task

5.8.4 Collecting additional information from the user

DriveDomainModel that models driving activity, includes parameters needed for driving activity and uses that information to gather input from the user thru the UI. It uses DialogManager to ask user to enter this missing information from the UI.

DriveDomainModel then iterates thru next piece of missing information, and identifies that specific airport is not entered by the user. It then uses web services to identify airports near the current geo-location, and presents this list of airports to the user. This web service is presented with current location of the user, and the approximate radius to search for airports.

Results from the web service call are then presented to the user in a dropdown list and will contain following airports, assuming user is currently around Boston, MA:

- Logan International (BOS)
- Manchester Regional Airport (MHT)
- Worcester Regional Airport (ORH)

By the end of this stage of input processing, agent has collected required information in order to plan the drive activity for the user.

Next stage of steps within the agent is related to getting sub-tasks associated with the activity and then planning the task.

5.8.5 Get sub-tasks for the activity from knowledge base

OpenCyc knowledge base enables use of existing knowledge in structured form that can be used to create a framework of task domain models that can rely on this pre-existing information to drive the task planning activity of the agent.

DriveDomainModel contains mapping to knowledge base entry TransportInvolvingADriver, which contains pre-conditions, parameters, actors, and sub-events related to executing this activity.

TransportInvolvingADriver

This concept in Cyc is mapped with synset-drive-verb-1 in Wordnet, and is identified as the relevant concept to be used for planning the task.

This concept identifies a driver as a required actor to accomplish driving activity. In turn, driverActor is a type of actor defined within Cyc to be a person. This enables Cyc to infer that we need a person to complete this activity and so this task can't be automated.

Pre-conditions:

- TransportInvolvingADriver needs a TransportationDevice. Cyc database will be populated with the user's vehicles as an instance of TransportationDevice type. User will choose one of these vehicles for driving in this activity.
- Driver needs to be at the location of the car in order to start driving
- This activity is a Situation-Localized type, which means that it can only be started from a specific location.

Parameters for this activity:

- TransportInvolvingADriver is a TemporalThing that has a
 - Starting time is the time when the driving activity will be started
 - Duration for the activity
- TransportInvolvingADriver is a TransLocation, which has a fromLocation and toLocation.
 - fromLocation is the starting point from where driver will begin driving in this activity
 - toLocation is the destination for this activity. In this use case, it will be the airport user has chosen as destination.

Sub-Events to decompose the task into steps for the agent

Cyc knowledge base is augmented with sub-events information that captures sequence of steps needed to plan and complete this activity.

- Go to the vehicle: user needs to be at the location of the vehicle. This sub-event will involve asking the user from going from his current location to location of the car
- Start driving vehicle from starting point to destination. Assuming user is driving the vehicle, directions will be provided to the user
- Park the car. Agent can identify the parking locations near the airport and help user to park the car.

5.8.6 Planning the activity

In order to plan for the driving activity, agent needs to process pre-conditions to begin this activity. Goal of this activity is to reach destination at say 4pm. Agent will work backwards using backward chaining planning algorithms to recommend actions to the user at specific times.

Activity plan for the user will look like:

- Begin activity (2:40pm)
- Walk to the vehicle (duration = 10mins)
- Driving from current location to airport (duration = 60mins)
- Parking at airport (duration 10mins)
- Goal: In airport at 4pm

Agent will remind the user to begin this activity at 2:40pm, in order to reach the goal of being at the airport by 4pm.

5.8.7 Monitoring the plan

Agent will be periodically checking for any changes in conditions such as weather or traffic conditions that can impact the plan, to ensure that plan remains feasible with the changes in conditions that can be sensed by the agent thru the use of web services that expose this information in real time.

5.8.8 Plan for the activities at the airport

Agent plans for 'Drive to Airport' as a driving activity and then at airport activity. Agent can use this activity as a sub-event in driving activity, and plan out the activities to be accomplished at the airport. This will enable the agent to recursively plan for the activities.

5.8.9 Summary of this use case

This use case demonstrates how different data sources such as Wordnet, OpenCyc, and web services that expose information via APIs in real time can be leveraged to parse, interpret and plan simple tasks for the user. It shows, specifically on how

some of the information such as actors, sub-events, and pre-conditions from OpenCyc can be mapped in domain models of the agent. In essence, integration of data across different sources as well as understanding of this data thru the use of semantic data mappings enables a reuse of these data sources as well as knowledge databases in building tools such as this agent, that can exhibit an understanding of the tasks they are planning.

6 Assumptions, Constraints and Limitations of the system and ways to address them

This chapter highlights limitations of the architecture and design of the smart agent, and how these limitations impose constraints in terms features provided to the user

6.1 Personal assistant can be built for specific domains, but integrating this with the knowledge of the domain is the challenge.

Knowledge database such as Cyc carries knowledge across multiple domains while the agent is built for particular task domains such as travel, shopping, entertainment etc. This requires partitions within the knowledge base in order to focus on relevant aspects of the knowledge that should be used and interpreted by the agent. Microtheories in Cyc encapsulate this partition by separating different domain knowledge into different buckets of microtheories. Even then, depth of knowledge may be different for different concepts and coupled with level of information needed for the agent, it becomes necessary to review the assertions and relationships contained in the knowledge base when implementing support for a new activity in the agent.

6.2 Voice interface for the agent

Personal assistant software should have simple interface, ideally with a speech dialog between the agent and the user. This is a natural interface for the user and does not involve typing commands in the agent's user interface. In addition, this speech interface should be able to interpret regular language speech without any need for using specific tailored commands. Task templates described in this agent are a limitation on usability of the agent.

Voice interface was not considered for this agent in order to keep the focus on data sources and in combining the data sources to solve the problem of task planning. A voice interface can be added to the agent system, by integrating voice recognition software into the input system, providing user to either use text input via user interface or thru the voice interface.

Voice recognition software can then be driven to provide a dialog interface with the user whenever a follow up information is needed from the user. This part is already designed into the dialog manager sub-system.

Voice interface will make the system usable and coupled with mobile device's touch interface enables a more intuitive and easier user interface enabling the user to involve agent in planning for more routine tasks.

6.3 User interactions with the agent

Agent is intended to help the user in managing his day-to-day tasks and make access to information faster. At the same time, agent needs input from the user when multiple choices are available and also needs to remind the user when specific deadlines are approaching. There is a need for a usability testing on what is the right level of interaction with the user, to keep the user engaged in using the tool and drawing value from it, without being too distracted from other activities.

Another open question is, what questions to ask the user while he starts venturing into action. Do we ask the user all the required questions to be able to complete the tasks? Or wait till we reach that sub task related to this task? These user interactions need to be fine-tuned and the agent should not ask for user's attention when he or she is not expecting one.

6.4 Stronger NLP to capture user input unambiguously

On the input side, user input via voice or text needs to be improved by using NLP algorithms that are trained on user tasks related data, and improve based on the user input to follow up user responses. This will help the agent to perform user activities, without a need for user to speak the language of the agent.

6.5 Performance of semantic web endpoints

Semantic endpoints such as DBpedia, GeoNames, FreeBase, OpenCyc that provide semantic data are less than reliable at this time in terms of availability, and also in terms of performance when the sites are available. This unreliability makes the system fail, especially in scenarios where these systems provide unique data that is necessary for keeping the agent operational.

One of the approaches of interacting with slow performing semantic endpoints is to cache some of the data, so that it's already available. And another alternative is to make these calls asynchronous and not be in the direct path of interactions with the user.

However, its expected that in the long run these semantic end points will become much better - performance as well as reliability wise.

6.6 Data accuracy and completeness

Semantic web endpoints provide data in RDF form to consumers, but this set of data may not be accurate at the time or may not be complete in terms of coverage. This leads to unreliability due to data quality. There needs to be a Quality of Service contracts on how the data provided by these endpoints is tested before hosting online. Unfortunately, these rules and policies will result in friction in exposing data online. But, will result in more resilient applications that use this data.

6.7 Proliferation of web services

Task models use web service managers to interact with third party web services to retrieve information from the web. An increase in supported use cases can result in proliferation of web services, with ever increasing integration points.

This can be addressed by having plugin architecture for integrating task models with web services. In this model, the plugin architecture will define a standardized interface for web service and its input and output parameters. And will expose an API that can be plugged into any task model that can generate this standardized input. Plugin provider will then extract the information from standardized form and translate into the form needed for this web service. This introduces an extra layer that can be used to outsource the integration of agent task models with web services, to third party vendors.

6.8 Task Models

Agent needs task domain models to understand a task, and integrate with different data sources that can be consumed to accomplish the task. These task models need to be implemented one at a time, and hierarchy of concepts within Cyc can help the agent plan the activity such as 'drive to theater' using the same framework that is used for 'drive to airport', even though subsequent activities after reaching the destination are different in both cases.

New task models for example, shopping related activities can be added by identifying related verbs in Wordnet, their corresponding concept in Cyc, pre-conditions, actors and sub-tasks within Cyc. And then using this information within the agent's domain model framework to integrate with real time data sources and planning the task.

In order to truly make this framework easily extensible, framework needs to be used in multiple use cases exposing to different user scenarios, contexts as well as data sources to be integrated with.

6.9 Planning

Tasks are modeled in Task domain models, and are mapped in Cyc for their decomposed sub-tasks. One of the challenges for the agent is to understand on how deep it needs to go in order to plan this task. Part of this information is hard coded by customizing the knowledge base for the supported tasks. But, in general giving the agent a notion of planning at an appropriate level for the user will enable the agent to remain useful to the user without being too detail oriented.

6.10 Sharing task decomposition with other users

Knowledge database provide flexibility in the design of the system, and let the agent be customized based on the user's planning approach to specific tasks. Sharing this knowledge across users can let the best practices be spread to different users, and thus helping formation of community of users sharing their individual knowledge about task planning with others, replicating productive behavior.

6.11 Ability to add new facts and task decomposition to database

In the current design, task domain models use knowledge bases to drive the task planning. But, there is no explicit interface to modify the knowledge base. One of the key features will be to enable the user to add content to knowledge database that can in turn influence how the activities are planned. This will in turn make the agent learn from the user, and become a personalized assistant thru the use of knowledge base.

6.12 Learning from user input and actions

In addition to getting user input for updating the knowledge base, user selections and patterns in tasks can be used by the agent to learn user preferences, affinities based on location and time. This information can then be used to improve the user experience by reducing input selections based on the prior selections of this specific user in similar use contexts.

6.13 Personal information on the mobile phone

User information is spread across different applications such as address book, emails, call records, calendar etc., on the mobile device. This information needs to be consolidated and provided in a single ontology that can enable applications to access this information in more meaningful ways, with a better understanding of the data they hold.

6.14 Integrating with social networks, and delegating tasks to other users in the network

Many users use social networks such as Facebook, to capture their routine activities and so turn out to be a digital diary of the person. In addition, these social networks also capture personal connections of the users with his friends and family, who are generally involved in helping him with the tasks.

Integrating the agent with social networks will enable delegation of tasks to people in the network, and can also help the agent system to be a part of social network with visibility into other connected user's tasks and planning information.

7 Conclusion and Future work

7.1 Personal Assistants are the future

Personal assistant software improves user productivity by managing routine tasks of the user and by providing information from online sources to the user. As discussed earlier, technologies such as web services, sharing of data, linked data, shared ontologies, knowledge databases, and mobile devices are proving to be enablers for tools such as personal assistant software.

Building an agent that can replace a human assistant has been a holy grail for software industry, especially in the field of artificial intelligence. Difficulties associated with capturing human intelligence in models that can be used to drive the agent have been one of the primary bottlenecks in building such agents. With the availability of data in semantic form, where the data carries itself the meaning and data sources are interlinked with each other, provides an opportunity to first capture human knowledge in this form and then apply reasoning engines that can interpret these models to make inferences for simple tasks.

This thesis work included conducting research on semantic web technologies, data sources that are available in semantic web form as well as web services, knowledge databases with a view to model simple day-to-day tasks of users using these technologies and to design personal assistant software that can leverage these technologies.

7.2 Supporting Tasks

As part of the thesis, two simple use cases were considered to demonstrate the viability of the solution to manage tasks that can be automated by the agent or planned by the agent for manual execution. Further research needs to be done to identify user tasks that can be managed by the agent. In the current design, task templates and models need to be created in order to support a task for the user. This can be improved by having a dynamic creation of these models based on the knowledge about the tasks in the knowledge databases. But, this will need a common ontology to be used by the web services, data source providers, knowledge databases and the agents in order to identify task attributes, its input and output parameters at all levels and using this common understanding to make the integration with the agent seamless.

In the absence of this commonly adopted ontologies, we are left with implementing mapping schemes at various levels to support tasks. This involves identifying the task and then integrating with the web services that can support the task. This has been a successful strategy with the existing personal assistant software such as SIRI, but can lead to proliferation of web services at the agent level.

7.3 Evolution of semantic web ecosystem

Current set of semantic web data sources have issues ranging from performance of the SPARQL endpoints, availability of these end points and quality of the data. This needs to be addressed by improving the database engines used to host semantic data. In addition, having more mature, fault tolerant systems to enable high availability systems that can be used reliably in mission critical applications.

Semantic web also suffers from a presence of large variety of ontologies and a few commonly used ones. Proliferation of ontologies is creating interoperability issues between similar data sources using different ontologies. It remains to be seen how the industry tackles this issue. One of the approaches would be to have standards organizations drive building frequently used ontologies and let everyone adopt the standard ontologies. Another approach is to create bridges between widely used and related ontologies so users of either ontology can avail of the features in the other one.

Another issue associated with slow pace of adoption of semantic web technologies is the lack of reliable tools that can be used to design and implement solutions using this data as well as to manage this data.

True power of semantic web will be utilized when these concerns are addressed and enterprises as well as people expose data in this format.

7.4 Building the agent

As part of future work, agent will be built based on the design elements in this thesis in order to prove the validity of the design and as part of this process improvements to this design and architecture will be made. A simple and intuitive user interface needs to be built and tested to make the interactions with the agent simple and productive.

7.5 Knowledge Databases

Knowledge databases such as OpenCyc do not provide a reliable SPARQL endpoint for consuming its data in RDF form. The RDF interface is only provided on the central servers managed by Cyc and is not provided in the open source version. This creates an unnecessary dependency on central servers managed by Cyc to implement this solution. Alternative strategies should be explored to correct this issue.

Sub-Tasks captured in Cyc are used in the design of the agent. This information for most part is not currently part of the knowledge base and needs to be added into the knowledge base. This information needs to be reviewed for any new task that is supported by the agent.

In order to scale the agent to support new tasks easily, task decomposition within Cyc needs to be enhanced by using tools that can identify related task and sub-task dependency relationship and populating the sub-tasks so as to meet the agent requirements for planning these tasks.

7.6 Planning Algorithms

Agent should be able to model complex task dependencies and use these models to recommend optimized plans for the user. It needs to be tested for finding optimum paths when a task has multiple sub-tasks and each sub-task can have its own sub-tasks. In such a case there can be multiple solutions to paths, and the agent should be able to consider user preferences, other active tasks, priorities in order to recommend a particular plan.

Agent can prove to be a useful companion for the user if it can take these considerations into account while suggesting plans to the user.

7.7 Reasoning

This design of the agent relies on logic incorporated within the task models of the agent along with ontology descriptions of the concepts in semantic data sources as well as in OpenCyc to make inferences. This can be improved by defining the rules needed by the agent as assertions and rules in OpenCyc and using OpenCyc inference engine as part of the agent to drive the logic in the agent.

8 Bibliography

- "Android OS - [http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))," n.d.
[http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system)).
- "Android System Architecture Diagram -
<Http://developer.android.com/images/system-architecture.jpg>," n.d.
<http://developer.android.com/images/system-architecture.jpg>.
- "CALO Website - <http://www.ai.sri.com/project/CALO>," n.d.
<http://www.ai.sri.com/project/CALO>.
- "CALO Wiki Page - <http://en.wikipedia.org/wiki/CALO>," n.d.
<http://en.wikipedia.org/wiki/CALO>.
- "DBpedia Knowledge Base - <http://www.wiwiss.fu-berlin.de/en/institute/pwo/bizer/research/publications/Mendes-Jakob-Bizer-DBpedia-LREC2012.pdf>," n.d. <http://www.wiwiss.fu-berlin.de/en/institute/pwo/bizer/research/publications/Mendes-Jakob-Bizer-DBpedia-LREC2012.pdf>.
- "DBpedia Usecases - <http://wiki.dbpedia.org/UseCases>," n.d.
- "ESD.34 System Architecture Course at MIT – Principles of Architecture Assignment," n.d.
- "GoodRelations Ontology UML Diagram -
<Http://www.heppnetz.de/ontologies/goodrelations/v1#uml>," n.d.
<http://www.heppnetz.de/ontologies/goodrelations/v1#uml>.
- "Google Glass Website - <Http://www.google.com/glass>," n.d.
<http://www.google.com/glass>.
- "Google Webservice Documentation -
<Https://developers.google.com/maps/documentation/webservices/>," n.d.
<https://developers.google.com/maps/documentation/webservices/>.
- "How SIRI Works - Interview with Tom Gruber, CTO of SIRI :
<Http://www.novaspivack.com/technology/how-hisiri-works-interview-with-tom-gruber-cto-of-siri>," n.d.
<http://www.novaspivack.com/technology/how-hisiri-works-interview-with-tom-gruber-cto-of-siri>.
- "Introduction to Wordnet - an Online Lexical Database -
<Http://wordnetcode.princeton.edu/5papers.pdf>," n.d.
<http://wordnetcode.princeton.edu/5papers.pdf>.
- "Knowledge Databases - <Http://www.mkbergman.com/913/metamodeling-in-domain-ontologies/>," n.d. <http://www.mkbergman.com/913/metamodeling-in-domain-ontologies/>.
- "Limitations in SPARQL 1.1 - <http://www.w3.org/2009/Talks/0615-qbe/>," n.d.
<http://www.w3.org/2009/Talks/0615-qbe/>.
- "Linked Open Data Cloud - Http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.png," n.d.
http://richard.cyganiak.de/2007/10/lod/lod-datasets_2011-09-19_colored.html.

- “OWL2 Primer - [http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic Web Presentation](http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic%20Web%20Presentation/),” n.d.
[http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic Web presentation.](http://www.w3.org/TR/2012/REC-owl2-primer-20121211/Semantic%20Web%20Presentation/)
- “RDF Primer - <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>,” n.d.
<http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- “RDF Tutorial - <http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/>,” n.d.
<http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/>.
- “RDFS - <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>,” n.d.
<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/>.
- “RDFS Classes and Resources - <http://www.w3.org/TR/2000/CR-rdf-schema-20000327/sets.gif>,” n.d.
<http://www.w3.org/TR/2000/CR-rdf-schema-20000327/sets.gif>.
- “Reqall Website - [Http://www.reqall.com](http://www.reqall.com/),” n.d. http://www.reqall.com.
- “SIRI Demo - [Http://vimeo.com/5424527](http://vimeo.com/5424527),” n.d. <http://vimeo.com/5424527>.
- “SIRI Patent Information - [Http://www.patentlyapple.com/patently-apple/2012/01/apple-introduces-us-to-siri-the-killer-patent.html](http://www.patentlyapple.com/patently-apple/2012/01/apple-introduces-us-to-siri-the-killer-patent.html),” n.d.
<http://www.patentlyapple.com/patently-apple/2012/01/apple-introduces-us-to-siri-the-killer-patent.html>.
- “SIRI Patent: US20120016678A1 - <http://www.google.com/patents?id=ISECAgAAEBAJ&printsec=abstract&zoom=4#v=onepage&q&f=false>,” n.d.
<http://www.google.com/patents?id=ISECAgAAEBAJ&printsec=abstract&zoom=4#v=onepage&q&f=false>.
- “SPARQL Cheatsheet - <http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet>,” n.d. <http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet>.
- “SPARQL Tutorial - [Http://www.slideshare.net/ldodds/sparql-tutorial](http://www.slideshare.net/ldodds/sparql-tutorial),” n.d.
<http://www.slideshare.net/ldodds/sparql-tutorial>.
- “SPARQL W3C Resources - <http://www.w3.org/2009/Talks/0615-qbe/>,” n.d.
<http://www.w3.org/2009/Talks/0615-qbe/>.
- “UMBEL for Ontology Development - [Http://fgiasson.com/blog/index.php/2008/08/29/umbel-as-a-coherent-framework-to-support-ontology-development/](http://fgiasson.com/blog/index.php/2008/08/29/umbel-as-a-coherent-framework-to-support-ontology-development/),” n.d.
<http://fgiasson.com/blog/index.php/2008/08/29/umbel-as-a-coherent-framework-to-support-ontology-development/>.
- “W3C OWL2 Specification - <http://www.w3.org/TR/owl2-syntax/>,” n.d.
<http://www.w3.org/TR/owl2-syntax/>.
- “W3C Semantic Web Site - [Http://www.w3.org/2001/sw/](http://www.w3.org/2001/sw/),” n.d.
<http://www.w3.org/2001/sw/>.
- “Why Use Logic? - Presentation at [Http://www.cyc.org](http://www.cyc.org/),” n.d.
- “Wordnet Reference Manual - <http://wordnet.princeton.edu/man/wngloss.7WN.html>,” n.d.
<http://wordnet.princeton.edu/man/wngloss.7WN.html>.
- “Wordnet: Design, Content and Limitations by Christian Fellbaum - <http://dydan.rutgers.edu/Workshops/Semantics/slides/fellbaum.pdf>,” n.d.
<http://dydan.rutgers.edu/Workshops/Semantics/slides/fellbaum.pdf>.

“Yelp Webservices Documentation -
Http://www.yelp.com/developers/documentation,” n.d.
<http://www.yelp.com/developers/documentation>.