FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Service Composition Using Knowledge Based Planning Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Information & Knowledge Management

eingereicht von

Jinze Li

Matrikelnummer 0027419

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: ao. Univ.-Prof. Dr. Jürgen Dorn

Wien, 26.09.2011

(Unterschrift Verfasser)

(Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Jinze Li
Lorenz Müller Gasse 1A/5521, 1200 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First and foremost, praise to my savior Jesus Christ for the undeserving grace bestowed upon me.

I am grateful to my parents, Guilan Xu and Zuoxing Li for their love, understanding and incessant support throughout my life.

My adivsor, Prof. Jürgen Dorn deserves my honest thanks for his candid advises and patient guidance. Without his helps, this thesis would not have been possible.

I am thankful to my beloved Leyi Shi for her motivation and caring during the years of my master study.

Last but not least I would like express my gratitude to Luke Chu Pastor, Ph.D who enlightened me during the trough of my life.

Soli Deo gloria

Abstract

Creating value-added services through composition of atomic service units has been gaining great momentum to date. The recent attempts of service composition emphasis on describing atomic services from both syntactical and semantic aspects and applying Web service as delivery method. This thesis addresses the composition issue by artificial intelligent planning techniques and argues the use of domain knowledge for representing service resources. To this end, methods of knowledge engineering must be used for providing rich knowledge models in order to solve planning problems in real life. Different planning paradigms are discussed in terms of suitability for solving planning tasks. In the thesis, a concrete knowledge domain is constructed as a planning domain to demonstrate the knowledge-based approach. Each atomic service in the domain is specified with prerequisites and outcomes. Based on the knowledge domain, a planning system is able to generate composite service that fulfills the preferences and goals defined by a user. Along with the domain, a prototype of service system integrated with the planning system is developed to foster practical application of knowledge-based planning tasks. A knowledge base serves as a domain repository that can be accessed on the Web. The prototype system along with the constructed knowledge domain has been successfully tested and evaluated on both functional and nonfunctional levels.

Kurzfassung

Die Erstellung der wertschöpfenden Dienste durch Zusammensetzung atomarer Serviceeinheiten hat bisher an großer Dynamik gewonnen. Der Schwerpunkt der jüngsten Versuche der Service-Komposition liegt in der Beschreibung atomarer Service aus beiden syntaktischen und semantischen Aspekten und der Anwendung von Web-service als Übergabesmethode. Diese Arbeit befasst sich mit dem Kompositionsproblem durch künstliche intelligente Plannungstechniken und argumentiert die Verwendung von Domäne-Wissen für Representation von Service-Resourcen. Zu diesem Zweck müssen die Methoden des Wissensengineerings für die Bereitstellung reichhaltiger Wissensmodelle zum Lösen der Planungsprobleme im realen Leben verwendet werden. Verschiedene Planungsparadigmen werden im Hinblick auf Eignung für die Lösung von Planungsaufgaben diskutiert. In der Arbeit wird eine konkrete Wissensdomäne als Planungsdomäne konstruiert um den wissensbasierten Ansatz zu demonstrieren. Jedes atomare Service in der Domäne wird mit Voraussetzungen und Ergebnissen angegeben. Basierend auf der Wissensdomäne, ist ein Planungssystem in der Lage, ein zusammengesetztes Service zu generieren, das die benutzerdefinierten Präferenzen und Ziele erfüllt. Zusammen mit der Domäne wird ein Prototyp eines Servicesystems mit dem Planungssystem integriert entwickelt, um die praktische Anwendung der wissensbasierten Planungsaufgaben zu fördern. Eine Wissensbasis dient als Domain-Repository, das in Web zugegriffen werden kann. Das System zusammen mit der konstruierten Wissensdomäne wurde erfolgreich getestet und auf beiden funktionalen und nichtfunktionalen Ebenen bewertet.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Since its birth in 1960s, the Internet has grown to become one of the important information infrastructures. From about 360 million Internet users in 2000 to 2 billion Internet users in 2011[1], we are witnessing an explosive quantitative growth in the last decade. Meanwhile, the Internet has been in the process of undergoing a rapid qualitative progress as well. Especially the Web has evolved from a ”web” of simple read-only hypertext documents to a platform where rich contents can be created and shared among users. We have the Web 2.0 phenomenon and face toward to Web 3.0[2]. The Internet changes the way we socialize. In addition to that, it also exerts strongly economic growth. Today the Internet accounts for about 3.4 percent of GDP in the large economies that create 70 percent of global GDP [Roxburgh and Manyika, 2011].

As the profound changes in the Internet keeps going, the services sector, which accounts for about 63 percent of world-wide GDP in year 2010[3], benefits from the introduction of ICT[4] and technological advances of the Internet. Development of the Internet in terms of speed and capabilities brings the services sector more innovations and yields higher productivities. Cloud computing, the Internet of Services and the Internet of Things are the keywords that reveal the potential of offering innovative customer services by the ICT industry.

In the context of a business environment, specific hardware and software components are in

1 Internet World Stats: `http://www.internetworldstats.com/stats.htm/` accessed in March, 2012.
2 Web 3.0: `http://www.labnol.org/internet/web-3-concepts-explained/8908/` accessed in March, 2012.
3 Data extracted from CIA - The World Factbook, 2010
4 ICT stands for Information and Communications Technology.

many cases not interesting for customers, but instead it is services that produce values. The ICT industry in response to it provides computer software in the form of services for customers. One reason of successfully applying this design paradigm boils down to the maturing of Web technologies. Services on the Web can be distributed as software components for availing service providers to accomplish business goals on the Internet. This way of delivering services is being widely adopted not only in enterprise level, e.g. in fields of EAI (Enterprise Application Integration) and B2B (Business-to-Business), but is also adopted by organizations that involve in collaborative computation across networks such as the Grid, or is used by individuals that need certain services on the Internet or need interactions with a system.

A stand-alone service on the Web can be located and accessed not only by people but also by different heterogeneous software systems. It may contain sets of classes and interfaces that act as black boxes to clients. This kind of service typically defines an atomic business object that executes a unit of work (e.g. provides customer data, operates a calculation or performs a language translation, etc.) according to client's request. However, the functionality of a single atomic service is very limited and cannot often satisfy client's needs in terms of complexity. To deal with possible complex functionalities of client's requests, processing and combining services in order to form a kind of new composite service is vital in realistic situations.

The traditional approach to developing composite service (Web service in particular) involves manual coding for the integration of available services. It usually relies on conventional programming languages like Java or C#, etc. The composition implementation is cumbersome and error-prone. The software industry commonly uses WSDL (Web Services Definition Language) to specify a Web service by defining service endpoint, communication protocol and syntax of input and output messages that are transmitted by a computer program. In addition to WSDL, languages like WSCL (Web Services Conversation Language), BPEL4WS (Business Process Execution Language for Web Services) and WSCI (Web Service Choreography Interface) have been proposed as the specification languages for Web service flow. Despite these efforts, the composition of service flows is still produced manually and the service composition job still remains a complex task. Regarding huge amount of possible available services on the Web, it is usually beyond the human capability to accomplish the composition process by hand. The ability to combine and produce services automatically or semi-automatically is an important step towards building of practical and complex applications efficiently and effectively . To this end, a considerable number of research efforts on this issue both academically and industrially have been made in the last decade.

One family of emerging techniques that has been proposed for the automated composition task is AI (Artificial Intelligence) planning techniques. AI Planning is a complex topic that has been extensively discussed as a critical part of AI in [Russell and Norvig, 2010]. As the authors stated in [Russell and Norvig, 2010], a state of the world is represented by a collection of variables. This world state can be changed by actions described by a set of action schemata that consist of preconditions and effects. When an action becomes executable, the corresponding preconditions of the action must be fulfilled. After the execution, the world state is changed according to the effects of the action. In an initial world state, a planning agent is able to find sequences of actions that result in a goal state. This process of transforming the world state to the goal state is particularly useful when it is applied to the service composition domain. Firstly, input and output data of a service can be treated as preconditions and effects of an action in the planning domain respectively. Secondly, a service also changes the world state in its application context. If published services are described as actions with preconditions and effects in the planning context, specified business goal states can therefore be satisfied by selecting and ordering services through appropriate planning algorithms.

1.2 Goals

Despite abundant literatures and projects on AI planning-based service composition, there is no uniform solution for the issue due to its nature of complexity in real life. Many related projects focus on solving specific scenarios. Existing composition frameworks are hard to customize to satisfy user-defined situations. The aim of the work presented in this thesis is to address major AI planning techniques suitable for service compositions and to break down a specific composition scenario into tasks that are manageable by an existing AI planning system.

The aforementioned standards such as WSDL, WSCL and BPEL4WS, etc. are designed to specify services on syntactic level. In this thesis we advocate a direction of formally describing services on knowledge level, i.e. specifying multiple types of domain knowledge explicitly and encoding the knowledge in a human- and machine-readable structure. We aim to augment an AI planning system with this knowledge and its domain model to support reasoning and solving of user-defined planning problems.

We try to promote the knowledge-based planning as a service enabling factor and implement an integrated prototype system for service composition in order to impact realistic problems. In this way, domain-specific knowledge can be shared, used and reused as resources on the Web. Depending on types and ranges of domain knowledge, the system may provide knowledge-intensive composite results that clients cannot solve by hand with ease.

1.3 Outline

The remainder of the paper is structured as follows:

- **Chapter 2**. We first discuss the theoretical definition of the term *service* and how services are incorporated systematically to co-create values. The following section of the chapter introduces the related standards and design patterns of web services. Typical approaches of service composition are given hereafter. We provide a detailed description of AI planning techniques related to service composition and discuss various planning languages that can be used for describing planning domains. After that we also discuss several forms of service composition from different perspectives and introduce some examples in areas where AI planning techniques come in handy. At the end of the chapter we briefly introduce the way of how to realize RESTful systems through the web application framework Ruby on Rails.

- **Chapter 3**. We first clarify the objectives we want to reach during the implementation phase. After that we devise a planning domain and specify it on knowledge level. This domain is used as an example for generating knowledge-intensive composite service that is enabled by the selected planning system SGPlan5. Test cases are given for demonstrating planning results. At the end, we formally analyze the service system integrated with the selected planner for service compositions and define the use cases the system should accomplish.

- **Chapter 4**. The design of the service system architecture is given at first. In the following sections we elaborate the details of how the system is realized. In particular, details of representing resources contained in the system are discussed. At the end implemented system features are illustrated with screenshots.

- **Chapter 5**. In this chapter we evaluated the achievements of the thesis both on functional and nonfunctional levels. Difficulties and problems we come across during the time of design and implementation are discussed.

- **Chapter 6**. We summarize the contents of the thesis and discuss the decisions we made in the implementation process. Finally we give an outlook on future work by giving directions of possible extensions and improvements.

- **Appendix A**: The source code of the domain file for learning design in PDDL is attached in the appendix.

- **Appendix B**: Three problem files as test cases for the domain of learning design in PDDL is attached in the appendix.

CHAPTER 2

Foundation

2.1 Service and Service Systems

The term *Service* has very differentiated meanings and can be defined from different perspectives and disciplines. Even IT-related terms like e-services, Web services, commercial services, etc., are widely used for referring to as just *Service*. As a matter of fact, as the quaternary sector[1] of the economy continuously grows, more services are implemented in the form of IT applications. Hence services in ICT industry often implicitly refer to e-services or Web services. However, the use of service-related terms has been a research subject in other domains long before IT and the Internet came along to the stage of influencing the world's economy to date.

From a traditional business perspective, a service is a business activity that is offered by service provider to its environment and produces intangible outcomes and benefits to its consumers [Baida et al., 2004]. Many researchers have taken various ways to characterize attributes of services. Kazuyoshi Hidaka concludes among the efforts of the researchers three common characteristics of services as follows[Hidaka, 2006].

- Intangibility: results of business activities operated by service providers are intangible and don't have a physical manifestation. Comparing to products in the manufacturing sector that can be handled physically, intangible nature of services makes service providers hard to price and manage services. In addition to these, It is difficult to measure and estimate productivity and quality of services as well.

1 The quaternary sector of the economy refers to the economic areas that are knowledge-based. They consist of industries providing information services, consultancy and R&D, etc. [Kenessey, 2004]

- Simultaneity: service delivery is considered as a process that includes simultaneous involvement of provider and consumer. In other words, unlike the traditional manufacturing of products in which the products are produced at first and then delivered to consumer, delivering and consuming services usually cannot be accomplished without involvement of users.

- Heterogeneity: services can cause different results and responses due to provider's factors and indeterministic and inconsistent factors of environments and users. Hidaka points out particularly the importance of quality of customer-provider interface. For instance, a simple flight service, which seems to be homogenous, can be customized for clients demanding extra flight comfort with business-class tickets or refundable tickets. The customer-provider interface in this case can be the employees of a flight service agency that is flexible enough to cater customer's needs.

Bitran and Lojo enumerate in [Bitran and Lojo, 1993] more concrete and unique characteristics of services: intangibility, perishability, heterogeneity, simultaneity, transferability and cultural specificity. The authors explain that the perishability of service offerings calls for complicated management of handling customer demands and service production capacity. For transferability, the authors state that customer experiences and expectations are transferable in the service sector. They illustrate this fact by an example in the retail sector. A consumer that purchases a tennis-racquet and a new automobile may compare the quality of salesperson's assistance, the response time of the organizations in which the sales are made, etc. Furthermore, cultural specificity also influences expectations and behaviors of service customers and providers. Dealing with gaps of different cultural orientations is obviously important for the success of service offerings in international contexts.

Considering these characteristics, a service usually involves interventions from clients and can be viewed as a kind of action or process in which values are exchanged between service provider and client. Interactive or concurrent actions are sometimes essential and unavoidable during a service lifetime, especially in case that the service is knowledge-intensive or requires customization and the accomplishment of the service depends on client's input and participation, whether through providing properties, labors or information [Sampson and Froehle, 2006].

As knowledge-intensive services continue to grow in the foreseeable future, authors of [Spohrer et al., 2008] propose an abstraction for the service revolution that is ongoing in 21st century: the *service system*. A service is defined as an application of resources (including competences, skills and knowledge) for benefits of others. Based on this definition, a service system is defined as collections of resources that co-create values with other service systems through shared Information. In [Spohrer et al., 2007] key types of shared informations are stated explicitly for language, law and measures. A language or a standardized encoding of information serve as alignments for coordination of different systems. Laws or contracts ensure all those that service systems can communicate with each other by a shared language and follow shared laws of the governing authority. Measures such as prices can be used to estimate the value of services. The reasons to propose and analyze service systems are to understand how to arrange operand

and operant resources for co-creating values effectively and methods and mechanisms of service system changes. An Operand resource in this context refers to a resource that performs actions on other resources, an operant resource is a resource that these actions are operated on [Spohrer et al., 2008].

Authors in [Spohrer et al., 2007] propose three areas that should be taken into account for understanding service systems and benefitting from them on theoretical level, i.e. science, management and engineering.

- For science, since a service system is usually made up of people, technology and internal and external service systems, identifying and describing the role of each of these components as well as the role of customer inputs and applications of competences are vital to further analyze the evolution and trend of service systems. As the service sector became the major part of developed economies and is becoming the major one in developing economies, more human labors have been replaced by applying technologies as substitutions. People use technologies based on the Internet such as online banking or e-commerce website rather than contacting with employees directly in banks or shops. The proportion of knowledge-intensive activities is increasing not only in the ICT industry, but also in other areas of the service sector such as in retailing and tourism.
- For management, a framework that evaluates returns on efforts of system transformation for improvements is proposed. A service system can be compared horizontally with others in terms of efficiency, effectiveness, sustainability by using shared information. Meanwhile, a service system can also be compared vertically with its current and previous states by using private internal information. The purpose of the framework is to understand possible improvements of a service system.
- For engineering, a main focus of this area is on researching conditions under which a service system improves itself. To this end, the nature of key resources, i.e. people, technology, shared information, influences the capability of improvements of a service system in different proportions, since these key resources make up all kinds of services that a service system is integrating to. A service system in which human expertise is intensively required is hard to scale up unless there are enough experts available in the market or there are enough resources to train more people with corresponding knowledge and skills. Technological resources also bring a service system advantages for improvements. By offering services on the Web, access to a service system is not constrained by geographic distances anymore. If a service system consists of informational resources, costs of creating, duplicating this kind of resources are in many cases almost ignorable. Organizing the three key resources in proper proportions is therefore a prerequisite toward designing a service system that can be improved with ease in the future.

The structure of a service system can be contrasted by front stage and back stage activities. The front stage represents the interaction that a service has with a service consumer that again can

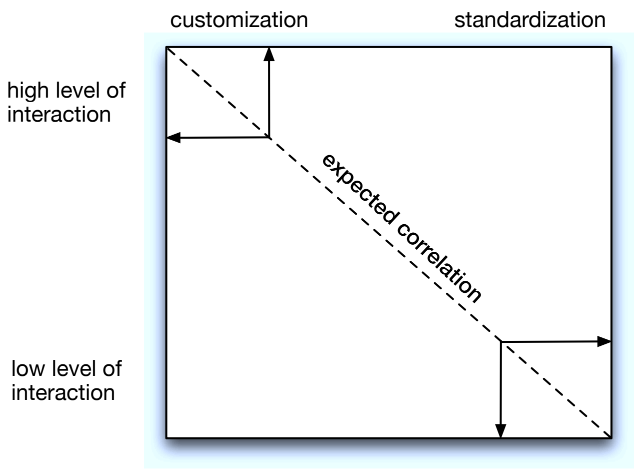


Figure 2.1: The service-intensity matrix

be a service as well. An interface of the front stage can be a receptionist of a hotel that directly contacts with customers or a web form that sends request contents of a user to a server. The back stage processes information or materials to create service products needed by the front stage. Teboul illustrates in [Teboul, 2006] the relationship between interactions and knowledge intensity involved in a service through a matrix (cf. Figure 2.1). This matrix can further be adopted to analyze service systems. If a high level of interactions is needed on the front stage, the customization through expertise of people is often required on the customer side. If standardization through shared information is more needed, the amount of interactions of a service system must be reduced.

2.2 Web Service related Paradigms

2.2.1 Service Oriented Architecture

For designing IT-related service systems, one architecture model that uses software components as modularized services is called Service Oriented Architecture (SOA). Each service is designed for being reusable. If a new business process requires the same task performed by a service, this new process can again employ the service that was used for a previous business process. Under the guidance of this architecture, these services can be coordinated to bring an expected business process to life.

In a SOA environment, system collaborators, i.e. service and service consumer (that again could be a service to others) should follow the following principles [Bean, 2010].

- Loose coupling: a service and a service consumer are physically independent from each other. It mitigates the impact of possible changes of the service and let the consumer be unaware of the changes.
- Interoperability: the collaboration between a service and its consumer is enabled, regardless of the various possible technologies on which the service and the consumer are based.
- Reusability: an existing service can be reused when requirements of new consumers can be satisfied by functionalities of the service. If a new consumer requires additional functionalities that the service does not includes, the service can be extended according to the consumer's needs whereas the impact of service changes to previously existing consumers should be reduced or eliminated.
- Discoverability: published information of services are saved in a service registry. It may include search and find capabilities to support service reuse.

A service is designed to encompass a certain logic. A service logic can be a task that needs to be accomplished so as to realize a business goal. A service can also encapsulate logics provided by other services. To let other services be aware of an existing service, the service uses a service description to define its own service name and expected return data. Furthermore, using messaging techniques enable the communication among services. As Thomas Erl stated in [Erl, 2005, chap. 3], the three components, i.e. services, descriptions and messages, are the core components of SOA. The keys to distinguish other distributed architectures are the different ways of how services are designed and build. Besides loose coupling, interoperability, reusability and discoverability, services should have service contracts and should be autonomous, abstract, composable and stateless [Erl, 2005, chap. 3].

- Contract[2]: a communication agreement should be defined to ensure interoperability. This can be established by specifying service descriptions and related documents.
- Autonomy: a service is capable of executing its encapsulating logic independently without any interference of other services.
- Abstraction: to the outside world, a service acts like a black box and remains invisible. Details about a service logic is hidden inside the service. The only visible information about a service is exposed via a service contract.
- Composability: a service logic can be classified by different levels of granularity. Composable services are able to be selected and assembled to build composite services.

2 In literatures like [Stal, 2006] "interfaces and contracts" are used for describing technical details of services including access information for consumers. Documents for service descriptions, such as WSDL, XSD schemata and service policies, can be viewed as an interface and a contract of a service. They collectively describe how a service can be accessed technically.

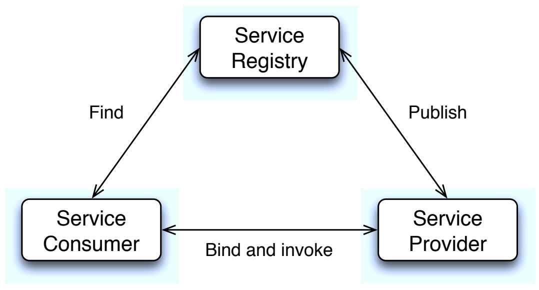


Figure 2.2: Collaborations in SOA

- Statelessness: services should retain stateless information so as to ensure loose coupling among services themselves.

Thomas Erl divides in [Erl, 2007, chap. 5] the above mentioned principles except of interoperability into two categories. Service contract, reusability, autonomy, statelessness and discoverability belong to one category in which the principles are responsible for implementation of specific service design characteristics and shaping concrete service logics. Loose coupling, abstraction and composability belong to another category in which they are applied to regulating application of other principles. In other words, while the principles in the former category try to append new specific physical characteristics to the service design, the principles in the latter category try to keep these new characteristics being implemented in a coordinated and appropriate manner.

In a realistic SOA application, service descriptions are usually saved in a service registry that is accessible to service consumers. If a desirable service is found by a service consumer, the service registry informs the consumer with the interface contract and access address of the service. According to the given information about the service, the consumer can subsequently bind and invoke the service that is provided by a service provider [Endrei et al., 2004]. Figure 2.2 illustrates the collaborations among service, service registry and service provider in SOA.

2.2.2 Web Service Protocol Stack

SOA is technology-agnostic and vendor-neutral, however technology capability is a requisite component of SOA implementation. One common service type in SOA is Web services. The World Wide Web Consortium (W3C) defines Web services as follows [W3C, 2004]:

> *A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically con-*

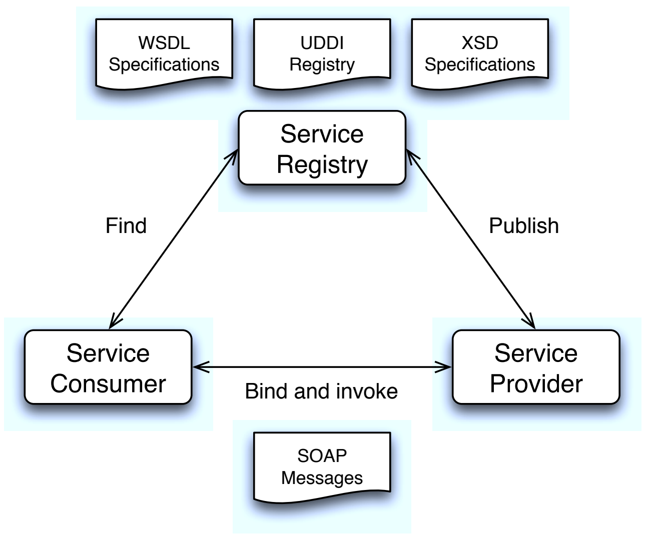


Figure 2.3: Standards in SOA

> *veyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Since a Web service replies on a set of strictly defined standards that consumers and services must comply with, interoperability of SOA is ensured. Web service protocols (refer to as WS-* or Web service protocol stack) that define and support Web services can vary in range due to different security, management and communication concerns [Bean, 2010]. However, four fundamental standards are core to Web service implementation, i.e. Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP), Universal Description Discovery and Integration (UDDI) and XML Schema Definition Language (XSD).

- WSDL: describes the service interface through which a consumer can call a service with expected parameters. A consumer can read the WSDL description to check available operations on provider side. The current version of WSDL is WSDL 2.0.
- XSD: is used as a metadata language that constraints and further describes special datatypes of the service interface. A WSDL may include references of XSD files and embed XSD data internally.
- SOAP: is a messaging framework. It consists of envelop information that includes header, body and optional fault area. A SOAP envelop contains an address of service endpoint for recipient, service protocol and binding information.
- UDDI: is a registry for published services on the Internet. It can be queried via SOAP messages and provides consumers access to WSDL specifications of services so that the

consumers can directly interact with the services by sending SOAP requests. Figure 2.3 depicts the roles of these four standards in SOA.

Since the four technologies above use XML as encoding method, collaborators of a SOA environment are able to produce and consume messages that can be understood by all the participants despite heterogeneous nature of their technology platforms. This is also an advantage of Web services and WS-* standards from which a service system can benefit in an IT ecosystem, since WS-* standards are largely supported by all major technology vendors in some manner [Bean, 2010]. Together with appropriately applying SOA principles, Web services provide an agile and flexible way for system development and integration.

2.2.3 REST and Resource Oriented Architecture

Representational State Transfer (REST) is a set of design criteria that were first introduced by Roy T. Fielding in his dissertation in 2000 [Fielding, 2000]. The methodology of REST focuses on exposing resources through exchanging their states as resource representations between client and server.

A resource in the realm of IT could be any type of data that is worth being created, used or manipulated, e.g. a plain text, a table in a database or a result after executing a software. A representation of a resource conveys the actual information a client needs, but could be in different forms such as a webpage in HTML or a graphic in JPG, etc. In order to communicate with a resource, a client needs to know the Uniform Resource Identifier (URI) of the resource and actions to which a server can respond. These actions are responsible for transferring client state from one to another. In addition to it, links in a requested resource can conduct changes of client state to related resources during new retrievals of resource representations as well.

Although the REST model is not restricted to a certain protocol, but it initially relies upon the core HTTP[3] commands, i.e. POST, GET, PUT, DELETE. In this sense, the mechanism of RESTful services mimics the fundamental operations of the World Wide Web (also known as the Web) that also introduces the concepts of resources, resource representations and URIs[4].

Comparing to the Web, a RESTful Web service request includes a HTTP method, a URI, a HTTP header and if needed, documents that a HTTP body should contain when using methods such as PUT and POST. This request is sent to a target HTTP server. After that, response data are parsed by client for further use. This is very similar to the case that a client enters an URL[5] in a browser, the browser subsequently issues a HTTP request to an target server and get a response back from it.

3 HTTP referred to as Hypertext Transfer Protocol

4 It is worth noting that Roy T. Fielding, who introduces REST in [Fielding, 2000], is also a coauthor of HTTP.

5 URL referred to as Uniform Resource Locator, it is a type of URI.

HTTP methods	CRUD functions	Caveats
POST	CREATE	POST only creates a child resource, i.e. a resource that is subordinate to another one identified by client's request-URI; CREATE is not limited by this kind of subordination.
GET	READ	GET can be conditional or partial so that resources in cache can be reused for better network usage in terms of efficiency; READ is not designed to provide features for better reuse of resources.
PUT	UPDATE	PUT replaces an entire resource; UPDATE only changes a resource
DELETE	DELETE	HTTP DELETE is normally mapped to CRUD DELETE

Table 2.1: Comparison of HTTP methods and CRUD functions

The four basic HTTP commands are also vastly similar to CRUD (Create, Read, Update, and Delete) functions that are widely adopted for persistent storage. For instance, a RESTful service request containing a HTTP GET command is similar to the "READ" action of a database, where the database contains the requested resource. If the resource is found, the requested data can then be sent back to client in forms such as Web page or XML message. However, when comparing HTTP methods with CRUD, differences should also be taken into account (cf. Table 2.1).

Being applied with HTTP and other standards such as URI and XML, a RESTful Web service is compliant with basic parts of the Web and hence is actually a part of the Web with benefits such as scalability and interoperability. In order to expose resources on the Web with the advantages of REST, one architecture called Resource-Oriented Architecture (ROA) can be used. The idea of ROA is to expose resources by utilizing existing web technologies (i.e. HTTP, URI and XML). This simplifies the design of web services because of the interface consistency across the Web. It also improves transparency of use and maintenance. The ROA should have the four following properties according to [Richardson and Ruby, 2007a]:

- Addressability: resources are accessible to clients by means of URIs in the web context. Each resource should have at least one unique URI that is descriptive enough for resource identification.

- Statelessness: resource state on the side of service provider is independent from client states that can be totally different from each other. Furthermore, a resource state remains the same for every client. When referring to a Web service that exposes a certain resource, a server only gets contacts with a client when it sends a request to the server for accessing the resource.

- Connectedness: quality of linking resources with each other can influence whether services are well-connected and fully RESTful. A RESTful Web service responds to client's requests not only by representation of the resource, but also by links to other resources so that a client can change its state from current one to others by following these links.

- The uniform interface: the aforementioned HTTP methods, i.e, GET, PUT, POST, DELETE form the fundamental operations a client can use to make use of a resource. This includes retrieving a representation of a resource, creating, modifying and deleting a resource.

As shown above, ROA relies only on the very basic technologies of the Web. To implement a Web service in way of ROA is likely to be much easier comparing to the SOAP-based WS-* approach that can probably be extended in terms of technology stack in different cases. This also potentially means shorter time and lower cost during development of ROA-guided applications. In addition to that, in ROA resource representations (usually in HTML that are human readable; in XML, RDF or OWL-S that are machine readable as well) should be linked with each other so that a client is able to navigate them along with the links and to search its desirable services on the Web. This also fulfills the functionality of UDDI but with much larger available service volume and less complexity.

2.3 Service Composition Approaches

Service composition is a process of selecting and combining existing atomic or composite services to create novel services. It creates value-added service outcomes that individual services cannot offer. Since Web services have become a popular paradigm for deploying services, the service composition task often refers to as Web Service Composition (WSC) as well.

When Web services are described by WSDL, they are only specified on functional level, e.g. definition of inputs and outputs. To further specify rules for controlling the order in which services are invoked can either be achieved in a static way when message exchanges and service bindings are known a priori or be achieved in a dynamic way when composite services are generated on-demand at execution time [Srivastava and Koehler, 2003].

2.3.1 Orchestration and Choreography

For the static WSC solution, service orderings are predetermined manually at design time. Two different approaches can be distinguished here, i.e. *orchestration* and *choreography* [Beek et al., 2006]. Figure 2.4 illustrates the difference between them.

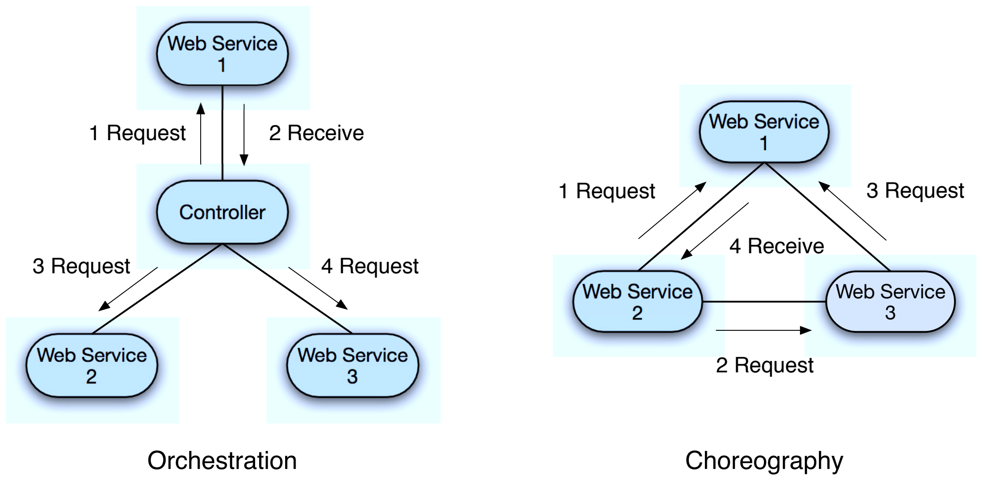


Figure 2.4: Orchestration and Choreography

- Orchestration: Web services involved in WSC are controlled and coordinated by a central controller. Only the central controller that is on the top of the WSC hierarchy has the knowledge and control of the overall business process that invokes and combines the participating services. Each participating service is only in charge of processing its own service requests. An important orchestration language is Web Services Business Process Execution Language (WSBPEL[6]) [Bucchiarone et al., 2007]. It is a process-oriented language that describes service behaviors both on executable and abstract levels. An executable business process defines functional behaviors of a participating service. An abstract business process defines public message exchanges between participants and is not directly executable. The current version of WSBPEL is WSBPEL 2.0.

- Choreography: An overall business process is established through collaborations of participating Web services. Process control knowledge is known to each service that follows its own rules autonomously hence no central authority is needed. Web Services Choreography Description Language (WS-CDL) is a message-oriented specification language that defines behaviors of participants in a choreography from viewpoint of their message exchanges. It has been recommended by the World Wide Web Consortium as a standard for choreographies of Web services [Kavantzas et al., 2004].

Comparing to choreography that describes exchange sequences of messages from a multi-party perspective, orchestration focuses on implementing control flows from a perspective of each individual party. They can complement each other when BPEL is applied to implementing par-

6 WSBPEL is also short for BPEL.

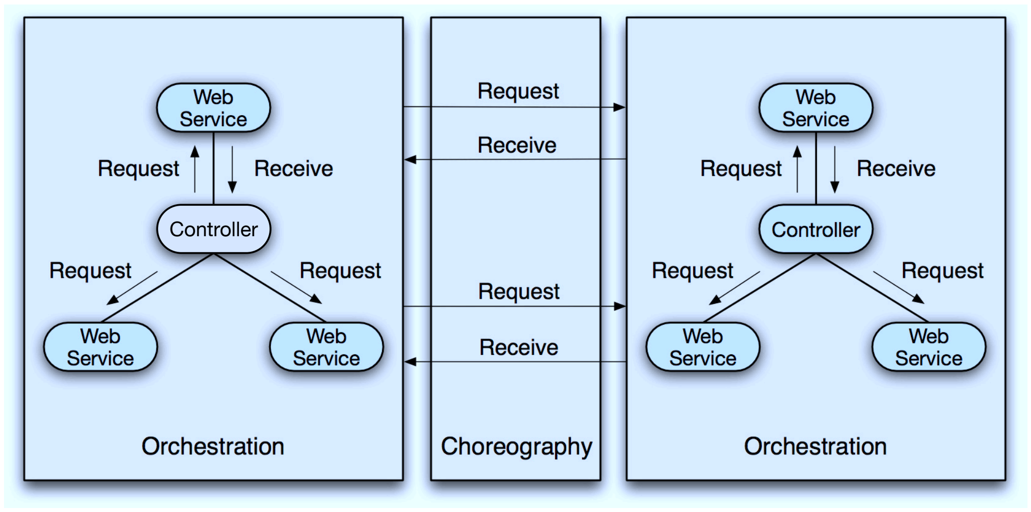


Figure 2.5: Relationship of Orchestration and Choreography

ticipating party's internals and WS-CDL is applied to specifying interactions of all parties from global view. Figure 2.5 depicts the complementary relationship between these two approaches.

As a matter of fact, in some service systems such as in B2B applications, there are impossible to establish a central controller that "orchestrates" system participants to reach a common business goal. On the contrary, the participants co-create a global contract on constraints and conditions of their interactions in a collaborative fashion. Their "choreographic" collaboration ensures a goal is fulfilled based on the contract while leaving implementation details to each participant alone. However, since in orchestration a controller takes the duty of managing the entire business process, Web services can be incorporated without knowing of their involvements on a higher level of a process, hence it is more conforming to the principles of RESTful Web services and ROA in which a Web service is only responsible for exposing or manipulating resources without being aware of taking part in a larger business process.

2.3.2 Semantic-Based Composition

The Semantic Web is envisioned to be a web of data that are machine understandable by introducing semantics to them [Lee et al., 2001]. This enables software agents to find and manipulate information of the web on their own with minimum human interventions. Similarly, Web services can also be given rich semantics to support semi-automatic or automatic service composition during a Web process lifecycle[7] for reaching a business logic.

In order to annotate Web services with semantics, one basic methodology is the application of ontologies [Cardoso and Sheth, 2005]. An ontology[8] defines classes, attributes and relationships among class members of a knowledge domain with their corresponding representative primitives [Liu and Özsu, 2009]. It abstracts a domain knowledge away from implementation details and hence is considered as a *conceptualization* of the domain on semantic level. With a shared ontology, Web services are able to communicate with each other by means of a formal domain specification.

Ontology Web Language for Services (OWL-S)[9] is a Semantic Web language that is designed for enriching Web services with semantics and enabling automation of WSC. An OWL-S ontology consists of three sub-ontologies written in OWL[10] as basic building blocks, i.e. service profile, process model and service grounding. The service profile describes "what a service does", for service discovery and matchmaking; the process model defines "how a service works", for purposes of enabling further matchmaking, composition, and monitoring; the service grounding defines how to access a service with detailed specifications of protocols and message formats, etc.

In OWL-S, a service is modeled as a process that specifies the way in which a client can interact with the service. To this end, a process model in OWL-S provides a specification of a service in terms of input, output, precondition and effect (IOPE). In oder to perform automated WSC, a software agent should be able to reason about logical relationships between IOPEs of available services and a user defined goal and then combine these services to collectively reach the goal without violating constraints of the IOPEs. To tackle the reasoning process, one way is to draw on Artificial Intelligence (AI) technology [Rao and Su, 2004].

7 [Cardoso and Sheth, 2005] stated a semantic Web process lifecycle includes description/annotation, advertisement, discovery, selection, composition and execution of Web services.
8 Another popular definition of ontologies can be found in [Gruber, 1993]: *An ontology is a specification of a conceptualization.*
9 OWL-S is still undergoing standardization at the W3C: `http://www.w3.org/Submission/OWL-S` accessed in April 2012.
10 Web Ontology Language (OWL) is widely used for creating ontologies and is a recommendation of W3C: `http://www.w3.org/TR/owl-features` accessed in April 2012.

2.4 Planning-Based Service Composition

2.4.1 Basic Concepts of Planning

A planning task in AI is to construct a sequence of actions leading from starting states to goal states. A classical planning[11]model can be defined as a 5-tuple system[12]:

$$\Sigma = (S, S_0, G, A, \Gamma)$$

where

- S is a set of possible world states.
- S_o is a set of initial states, $S_o \subset S$.
- G is a set of goal states, $G \subset S$.
- A is a set of actions that are applicable to its planning domain.
- $\Gamma : S \times A \longrightarrow 2^S$, Γ is a state transition function that transforms a set of states to other ones by applying an action.

Each attempt of finding a formal planning solution that satisfies a goal state in Σ is to identify a sequence of actions $A_k = (a_o, a_1, a_2...a_n)$ (where $A_k \subseteq A$) that are applied to the corresponding state transition functions $\Gamma_k = (\tau_0, \tau_1, \tau_2...\tau_n)$ (where $\Gamma_k \subseteq \Gamma$) respectively with $s_1 = \tau_0(s_0, a_0), s_2 = \tau_1(s_1, a_1)..., s_n = \tau_{n-1}(s_{n-1}, a_{n-1})$ (where s_n is a goal state).

The objective G that a planning solution is trying to achieve is not only restricted to one single state. It can also consist of a group of states or might be to lead the transition system to keep away from a particular set of states during execution time of a plan or can optimize utility functions according to certain metrics.

In a static environment that is fully observable for an agent, planning can be done prior to the execution of a plan, this is called *offline planning*. Opposite to it, in a dynamic environment where contingent events occur without pre-knowledge of a planning agent, *online planning* interleaving planning and execution is usually used for revising a planning process toward G. A conceptual illustration of the two models is shown in Figure 2.6.

11 A classical planning problem refers to planning in an environment that is static, fully observable and deterministic. This is contrary to non-classical planning where environments are dynamic, stochastic and only partially observable [Russell and Norvig, 2010, chap. 10].
12 In [Nau et al., 2004a], a classical planning model without considerations of *contingent events* is called *restricted state-transition system.*

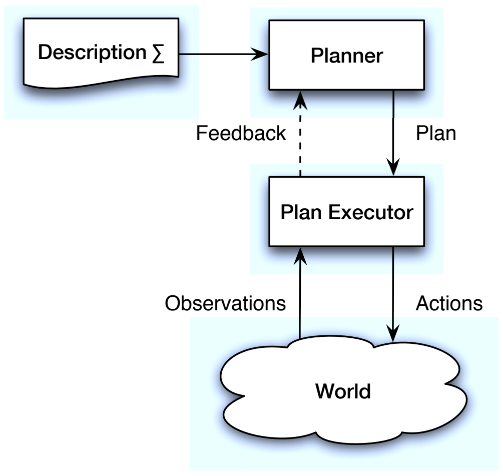


Figure 2.6: Offline and Online Planning Models

The planner takes Σ for representation of the real world as input and generates a plan, i.e. a course of actions, as output. The plan executor that can be a machine or a human executes the plan according to the course of actions. An executed action can either be a sensing action[13] or a world-altering action. The former one only collects information on the current real world such as extracting a piece of information from a website; the latter one changes the world state physically such as making a bank transfer that consequentially lead monetary changes on both sides of credit and debit. In case of *online planning*, after the plan executor perceives the world state, it gives feedback (in dashed line in Figure 2.6) upon execution status of the plan to the planner for possible adjustments of planning process.

Based on distinct configurability in different planning domains, three types of planners can be classified, i.e. domain-specific planners, domain-independent planners, and domain-configurable planners [Nau, 2007]:

- domain-specific planners are implemented specifically for a given planning domain and are usually not compatible to other domains without significant modifications of the planners. Due to their uniquely tuned planning techniques, this kind of planners are usually efficient for solving their own domain specific problems.
- domain-independent planners have no domain-specific knowledge. Planners based on classical planning approaches are usually domain-independent. The performance of the planners are less effective than the domain-specific ones, but the general-purpose nature makes them easy to configure and also adoptable in a much wider application spectrum.

13 In some literatures, sensing action is also called information gathering action.

- domain-configurable planners have domain-independent planning engines but domain-specific descriptions, i.e. a set of domain-specific knowledge such as control rules that are interpretable for planning engines. The domain-specific knowledge saves the planners reaching unnecessary search paths during planning processes and hence increases efficiency of the planners comparing to the domain-independent planners.

2.4.2 Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) is a de-facto standard encoding language for definition of planning domains. It was first introduced as the official language in 1998 during the first International Planning Competition (IPC) [McDermott, 2000] and has been evolved in every IPC hereafter. The latest version of the language is PDDL3.1[14]. The original purpose of PDDL was to unify similar domain description languages prior to the first IPC so as to facilitate domain sharing and empirical evaluation of participating planners [McDermott et al., 1998]. In this section we focus on basic structures of PDDL. The elementary syntax is based on PDDL1.2.

A basic planning problem described by PDDL encompasses a *domain* description and a *problem* description[15]. General domain elements that are present in every planning instance are included in one domain description and specific elements of planning instances can be included in different problem descriptions. The benefit is that several planning descriptions can be connected to one related domain description hence increases the reusability of the domain.

Although PDDL is devised to describe *the physics of a domain* [McDermott et al., 1998] in general, i.e. to express properties of the physical world with designated domain predicates, actions and action effects of domains, etc., basic assumptions suitable for planners that have different capabilities should still be set in each PDDL-written domain description.[16] One of the assumptions embodies on which formal language a description file is based, e.g. the assumption of STRIPS[17] or ADL[18]. A brief comparison of STRIPS and ADL can be found in Table 2.2.

STRIPS follows CWA that is the same to the semantics of data in database, i.e. unmentioned literals in a domain are considered to be false, while ADL's OWA considers unmentioned literals to be unknown. Conditional effects of ADL denote that a specified effect takes place only if the

14 This version was used in IPC-06: `http://ipc.informatik.uni-freiburg.de/PddlExtension` accessed in April, 2012.

15 Although it is not required in PDDL specifications, many planners compulsorily accept the domain and problem descriptions as separated files.

16 Most planners do not support all features of PDDL, but only subsets of them. Hence specifying domains with compatible features in PDDL files will help planners understand semantics of the domains without erroneous interpretation.

17 The Stanford Research Institute Problem Solver (STRIPS) is an automated planner and at the same time refers to the language that the planner uses as representation of planning domains. Classical planning languages are strongly influenced by STRIPS. [Russell and Norvig, 2010, chap. 10]

18 The Action Description Language (ADL) is a further extension of STRIPS with more expressive power.

STRIPS	**ADL**
Closed World Assumption (CWA)	Open World Assumption (OWA)
only positive literals in states	negation ($\neg$) of literals in states allowed
propositional, ground & function-free first-order literals in states	propositional, first-order literals in states
only ground literals in goals	quantifiers, e.g. ($\exists, \forall$) allowed in goals
conjunction ($\wedge$) in goals	conjunction ($\wedge$) and disjunction ($\vee$) in goals
no equality ($=$)	equality allowed
only conjunctions in action effects	conditional action effects allowed
no variable typing	variable typing allowed

Table 2.2: Comparison of STRIPS and ADL

corresponding conditional formulae are true in the state in which the action is executed. This syntax can be used to express many realistic domains in which action effects are situation-dependent. For instance, a purchasing agent trying to grab an item on an online auction website may change its bid price according to the prices given by its competitors during the time of a auction deal. The expressive power of ADL is obviously much higher than STRIPS and still remains restrictive enough for efficient reasoning. Comparing to STRIPS, ADL formalism is a better trade-off in terms of expressiveness and computational complexity of reasoning [Pednault, 1994].

To define the supported subset of PDDL features for a specific planner, the syntax `:requirements` [19] should be placed as part of a domain description as follows:

```
(define (domain domain_name)
  (:requirements [:strips] [:adl] [:typing] [:constants])
  ...
)
```

As the above code snippet illustrates, the declaration of `:requirements` follows directly behind the name of a domain description. `:strips` and `:adl` are the options for defining the formal language of a domain description. The `:typing` tag defines not only typed objects, but can also define a hierarchy of typed objects for a domain. `:constants` indicates that the domain description include constants that can either be typed or untyped.

19 For some planners this syntax is optional while for others it is mandatory to define the `:requirements` as part of a domain even if it is empty.

After defining `:requirements`, a domain description further contains predicates that are properties of domain objects that can either be true or false. The field `:predicates` in a domain description includes predicate names, a optional list of arguments identified by their beginnings with a question mark "?" and their types, if the domain has typed constructs. For instance:

```
(define (domain domain_name)
  ...
  (:predicates (predicate_name_1 ?a1 - type1 ?a2 - type2 ...)
               (predicate_name_2 ?b1 - type1 ?b2 - type2 ...)
  )
  ...
)
```

Along with predicates, another fundamental element of a domain description is actions. Actions characterize behaviors of domain objects. Optional parameters of actions are defined as variables that have their own values for each specific problem instance. The instantiation of the variables differs in groundings of various planning situations. Besides the parameters, actions have its own preconditions and effects that are defined as logical propositions combined together with logical connectives. The logical propositions are constructed from predicates and objects from each problem instance. The following example shows a basic action schema:

```
(define (domain domain_name)
  ...
  (:action action_name
    (:parameters (?p1 - type1 ?p2 - type2 ...))
    (:precondition precondition_formula)
    (:effect effect_formula)
  )
  ...
)
```

The `precondition_formula` in the action schema can only be a conjunction (`and` in PDDL) of atom formulae when in the `:requirements` field `:strips` is declared. The `effect_formula` for the `:strips` option is a conjunction of added (positive) and deleted (`not` in PDDL) atom formulae. For instance:

```
(:action action_name
  (:parameters (?p1 - type1 ?p2 - type2 ...))
  (:precondition (and atom_1 atom_2 ...))
  (:effect (and (not atom_1) (not atom_1) atom_3 atom_4 ...))
)
```

For the `:adl` option in `:requirements`, equality =[20], negation `not`, conjunction `and`, disjunction `or` and quantifier `forall`, `exists` can be applied to `precondition_formula`; the construct `when`

20 Equality = can also be used in `:strips` domains when the `:equality` tag is declared in `:requirements`.

for conditional effects can be used in `effect_formula` additionally.

A problem description contains an initial state and a goal state for a specific problem instance. They are grounded with concrete domain objects and values when applied to a corresponding domain description. The `:init` and `:goal` fields contain definitions of initial state and goal state descriptions. State descriptions in both fields must be grounded atoms unless quantifiers are used for the definitions. A schema of a problem description can be formulated as follows:

```
(define (problem problem_name)
  (:domain domain_name)
  (:objects obj1 obj2 ...)
  (:init atom1 atom2 ...)
  (:goal atom3 atom4 ...)
)
```

2.4.3 Extensions of PDDL

During the evolution of PDDL, there are two significant extensions of the language, i.e. PDDL2.1 and PDDL3.0, that were introduced in IPC-3[21] and IPC-5[22] respectively. They extend the initial version of PDDL in way that the planning language is tailored to be more expressive for modeling certain classes of domains while maintaining backward compatibility to its previous version.

PDDL2.1 includes new features such as numeric fluents[23], durative actions and plan metrics [Fox and Long, 2003]. Numeric fluents are domain functions that is of type $Object^n \rightarrow \mathbb{R}$, where $Object^n$ is a set of objects in a problem instance[24]. Assuming a car drives from location 1 to location 2 will consume 50 units of fuel and increase the drive cost up to 80 units. The numeric fluents can be formulated like this:

```
(define (domain travel)
  (:requirements :typing :fluents :adl)
  (:types location car)
  (:predicates
    (in ?c - car ?l - location)
  )
  (:functions
    (fuel ?c - car)
    (cost ?c - car)
  )
```

21 IPC-3: `http://planning.cis.strath.ac.uk/competition/` accessed in April, 2012.
22 IPC-5: `http://zeus.ing.unibs.it/ipc-5/` accessed in April, 2012.
23 The use of fluents in a domain is indicated in `:requirement` with `:fluents`.
24 In PDDL3.1, additional fluent along with the numeric one is object fluent that is of type $Object^n \rightarrow Object$. `http://ipc.informatik.uni-freiburg.de/PddlExtension`

```
  (:action drive
    :parameters (?l1 ?l2 - location ?c - car)
    :precondition (and (>= (fuel ?c) 50) (in ?c ?l1))
    :effect (and (decrease (fuel ?c) 50) (increase (cost ?c) 80)
                 (not (in ?c l1)) (in ?c l2))
  )
)
```

In order to update the value of a fluent, the action effect should include direct assignment `:assign` or indirect assignments `:increase` or `:decrease`. Furthermore, fluents must be initiated in the `:init` section of a problem description. If a problem instance specifies that a car should drive through various cities in several possible driving routes and one objective is to maximize the remaining fuel of the car, a plan metric can be placed in the problem description for defining in which way possible plans are to be evaluated during a planning process:

```
(define (problem travel-problem)
  (:domain travel)
  (:objects c - car ...)
  (:init (= (fuel c) 0) ...)
  (:goal ...)
  (:metric maximize (fuel c))
)
```

As seen above, the simplest case for a plan metric is to maximize or minimize the metric. Ideally, when initial and goal states remain unchanged, different plans should be produced by a planning system according to different plan metrics. However some planning systems do not use plan metrics for guidance of planning processes and just use them for manuel evaluations of plan quality hereafter.

Durative actions in PDDL are actions that last a certain time duration during an execution. A time duration could be a constant or a variable time interval. A durative action can be considered as a group of two basic PDDL-actions that take place instantaneously. One action represents happenings at beginning of the durative action, the other one represents happenings at end of the action. Each of the actions has its own preconditions and effects (with the constructs `at start` and `at end` respectively) that have the same semantics as normal PDDL actions. Along with the two instantaneous actions, a durative action could also have invariants (with the construct `over all`) that remain unchanged during the time interval of an action execution. For instance, a modified version of the `drive` action mentioned previously can be defined as follows:

```
...
(durative-action drive
  :parameters (?c - car)
              (?from - location)
              (?to - location)
  :duration (= ?duration (drive-time ?from ?to))
  :condition (and (at start (at ?c ?from))
                  (at start (>= (fuel ?c) 50))
                  (over all (trafficable ?from ?to)))
  :effect (and (at start (not (at ?c ?from)))
               (at end (at ?c ?to))
               (at end (decrease (fuel ?c) 50))
               (at end (increase (cost ?c) 80)))
)
...
```

The duration of the `drive` action depends on the fluent `drive-time` that should be initiated in a problem description. The `over all` invariant describes the road between the departure place (defined in `from`) and the destination (defined in `to`) of a vehicle should stay connected and passable during every execution of the action. The `:condition` containing the invariant also specifies a condition with `at start` stating that a car must be at the departure place at the time point of an action execution. An effect of the action takes place immediately with the annotation (`at start` in `:effect`) defining that a car is no longer at the departure place once the action is applied. At the end of an execution interval, the effects of the action are that a car is at the destination, consumed fuel is decreased and drive cost is increased.

Since the introduction of PDDL3 on the deterministic part of IPC-5 [Gerevini et al., 2008], hard and soft constraints are also integrated in PDDL as standard features. Hard constraints (short for constraints) must be kept true in an entire sequence of states produced during an execution of a plan, hence they are also called state-trajectory constraints. Soft constraints (short for preferences) must not necessarily be satisfied in a plan, however it could be included in plan metrics in order to increase the quality of a plan by means of maximizing the number of satisfied preferences. The appearance of constraints and preferences are restricted to action preconditions and goals. It is not allowed to define constraints in conditional effects.

Hard constraints are formulated by temporal modal operators that involves first order formulae and state predicates [Gerevini and Long, 2005]. Basic operators includes `always`, `sometime`, `at-most-once`, `at end` for expressing how conditions must be satisfied in plan states; `within` for defining deadlines; `sometime-before`, `sometime-after`, `always-within` for defining preferences over a sequence of events. For instance, when organizing a road trip by a car, several cities are chosen as candidate places that a car should reach. Among various possibilities of drive routes, Vienna and Rome are the two cities a car must reach. These hard constraints can be defined as follows:

```
...
(:constraints
  (and (always (visited Vienna) (visited Rome)))
)
...
```

Soft constraints is labeled as `preference` meaning that they do not have to be satisfied in order to reach a corresponding goal state or a precondition. However it does improve the quality of a plan when soft constraints are met. When two plans satisfying the same hard constraints but different subsets of soft constraints, plan quality can still be evaluated when the preferences are allocated with their violation costs. This is accomplished through the `is-violated` function that provides a quantitative measure for the violation of preferences. For instance, a car should reach Vienna and Rome. But the car driver desires to visit Vienna much more than Rome, then a relatively bigger violation cost can be associated to the driver's preference of `(visited Vienna)`:

```
...
(:constraints
  (and
    (preference p1 (visited Vienna))
    (preference p2 (visited Rome))
  )
)
(:metric minimize (+ (*10 (is-violated p1)) (*1 (is-violated p2))))
```

The two `is-violated` functions are alias of the preferences with the inputs `p1` and `p2`. Since the functions are assigned with their corresponding weights `10` and `1`, results of the metric are products of the weights and numbers of preference violations. Hence the smaller the number of violations is, the better the quality of a plan.

With fluent expressions inherited from PDDL2.1, PDDL3 can be used to model domain resources in numerical forms and allow domain users to define and customize their desires through constraints and preferences in each problem instance. The ability of generating a sequence of actions that bring a system from its initial state to a goal state and the ability to customize planning goals can be beneficial for service compositions when composition problems are characterized as AI planning problems.

2.4.4 Variants of PDDL

Beyond the canonical PDDL and its standard extensions (cf. Chapter 2.4.2), there are several variants of the planning language in the AI community. NPDDL, PPDDL, RDDL to name a few, each of which is developed to deal with knowledge representation of different situations of complex planning domains.

- NPDDL (or NuPDDL): it is the input language of the planner MBP (cf. Chapter 2.4.5). This language is designed to be able to model non-deterministic and partial observable planning domains [Pistore et al., 2003]. It is compatible with PDDL2.1 including the planning constructs such as functions, conditional effects and qualifiers. With the statement `oneof` and `unknown`, uncertainties in initial state of a problem description can be described[25]. These two keywords can further be used to represent nondeterministic effects in planning actions. The definition of sensory actions beginning with the statement

25 Examples of the language can be found in `http://mbp.fbk.eu/NuPDDL.html` accessed in June, 2012.

`:observation` allows results of observations to be expressed in boolean type. Through NPDDL, different levels of constraints in goals can be specified with the constructs such as Do Reach, Try Reach, Do Maintain, etc. The CWA is applied here as in STRIPS domains.

- PPDDL: this variant of PDDL is primarily designed to describe MDPs where the planning environment is fully observable but uncertainty exists (cf. Chapter 2.4.5) [Younes and Littman, 2004]. Numeric values of probabilistic distribution can be assigned to initial state of a planning problem and actions effects. The arbitrary nesting ability of conditional and probabilistic effects in PPDDL can be utilized to represent planning problems as Dynamic Bayesian Networks (DBNs) when embodied action schemata are fully grounded. Fluents can be used as rewards for action effects and goal state achievements.
- RDDL: it is strongly influenced by (P)PDDL, but significantly differentiated from these languages [Sanner, 2011]. Actions, states and observations are described as variables. The support of observations and exogenous events enables the language to express partially observable domains. A grounded RDDL domain is also a DBN comparing to PPDDL, but RDDL is more expressive due to its rich set of constructs.

2.4.5 Planning Techniques for Composition Tasks

Classical Planning Techniques

The classical planning technique for generating plans is based on searching of state space. A state space graph usually consists of nodes that represent world states and edges that represent actions. The search process can be either a *forward* (*progression*) search or a *backward* (*regression*) search. The forward approach searches from an initial state to possible successor states by applying domain actions. It searches progressively in order to reach a subset of goal states throughout state space. The backward approach searches from a subset of goal states. It applies actions backward until initial states are reached. Only actions that are relevant to the goal or the current state are considered during a state-space search.

State-space searches are often guided by heuristic functions for improving efficiency [Russell and Norvig, 2010, chap. 10]. In the context of a search graph, a heuristic function is a method that estimates the distance between the current node and the node of a goal. The better a heuristic is, the fewer the number of nodes is generated. This also means less states are stored and a solution is found faster for a planning problem. One way of obtaining a heuristic is the relaxation of a planning problem such as using planning graphs [Nau et al., 2004a].

A planning graph is a layered and directed data structure [Blum and Furst, 1997]. It expands itself layer by layer during a search while examining in each graph expansion whether a resulted planning graph satisfies conditions for plan existence. If the examination is positive, a solution extraction using backward search is executed. A planning graph starts in the first layer with all literals that are true in an initial state. Each of these literals represents a node. By connecting with these nodes, the second layer contains all actions whose preconditions are true in the first layer and they must be non-mutex[26]. In addition to the actions, the second layer also contains all true literals in the first layer and literals representing the effects of the actions. If layer two has goal literals and none of them is mutex to the ones in the same layer, a backward search is executed in order to find a possible plan. The same logic of the expansion and examination takes place successively in following layers until a solution path is found in the graph.

One well-known planning system that utilizes both state-space and planning-graph approaches is the Fast Forward (FF) planner [Hoffmann, 2011]. It depends on forward search in state space and is guided by heuristic values that are calculated through the planning-graph technique. A further extension to FF planner is Metric-FF [Hoffmanng, 2002]. It supports more expressive formalisms including numerical variables, constraints and effects that are introduced in PDDL2.1 and optimization of planning. This enables the planning system to implement service compositions in a wider spectrum of scenarios and also allows implementation of quality evaluation on service properties. Another extension of the FF planner is Conformant-FF [Hoffmann and Brafman, 2004]. It has an additional ability to express uncertainty in initial state in the form of a CNF formula. With aid of PDDL translator, Conformant-FF can be applied to WSC tasks on functional level[27].

The state-space search could be problematic when services need to be invoked in parallel, since each state-space search only generates a totally-ordered plans, i.e. a linear sequence of actions. Planners that produce such plans are called total-order planners. On the contrary, partial-order planners produce plans that may leave order of actions undetermined, i.e. any order of the actions is valid, if these actions do not interfere with each other. Such plans are partially-ordered plans.

Plan-space planning decomposes goal states into individual subgoals and searches solutions for each subgoal separately. Each node of search space is a partial plan that may consist of partially instantiated actions and constraints[28]. A planning process proceeds through refinements of a plan by introducing more constraints or actions, until in the plan no more flaws such as open goals or conflicting actions exist. The plan-space planning technique can be adapted to solve incomplete information and nondeterministic behaviors that could appear in service composition domains like the solution introduced in [Peer, 2005a], an agent uses an extension of a partial-order planner called VHPOP (which is based on partial-order planning) that integrates replanning feature for alternative plans based on mistakes of previous search attempts.

26 A pair of actions are mutex in the same layer when they have *inconsistent effects*, i.e. an effect of an action negates an effect of the other one; *interference*, i.e. one action deletes a precondition of the other one; *competing needs*, i.e. two actions have mutex preconditions. Two literals are mutex when they have *inconsistent support*, i.e. one literal is the negation of the other one [Blum and Furst, 1997].

27 CONFORMANT-FF: `http://www.loria.fr/~hoffmanj/cff.html` accessed in April, 2012.

28 Such constraints include *precedence constraint*, i.e. action *a* must precede action *b*; *binding constraints*, i.e. equality and inequality; *causal link*, i.e. execution of action *a* causes a precondition of action *b* to be true [Nau et al., 2004a, chap. 5].

Hierarchical Task Network

The Hierarchical Task Network (HTN) Planning approach involves incorporation and exploitation of domain-independent control knowledge. A HTN planner considers a planning goal as a collection of tasks to be performed rather than goals that are declaratively defined in classical planning domains. The planning process of HTN is a process of decomposing these tasks into subtasks recursively until atomic (primitive) tasks can be performed by actions directly. Domain descriptions in HTN include definition of actions for solving primitives tasks and definition of methods, each of which is used to guide decomposition of a compound task into subtasks. If a decomposition with a method turns to be impossible during a planning process, the HTN planner will backtrack to previous state in which other methods can be tried for the decomposition task. A planning process is considered to be successful when a desired compound task is decomposed into a group of primitive tasks without any violation of given conditions.

The idea of HTN planning has similarity of some tasks in real life that already have internal hierarchical structures. For instance, a trip arrangement task could consist of *purchasing a flight ticket*, *booking a hotel room*. The subtask *purchasing a flight ticket* could further divided into subtasks such as *checking availability of flights* and *booking a flight ticket*, etc. In [Nau et al., 2004b] methods of HTN are described as *recursively composable workflows*. The application of the workflows could be accomplished by independent services that are offered by different service sources. The modularity of HTN methods also brings good scalability when applied to service compositions.

SHOP2 (Simple Hierarchical Ordered Planner 2) is a domain-independent and HTN-based planner [Nau et al., 2003]. Its planning approach is a modified version of HTN that is called ordered task decomposition. This technique generates steps of each plan in the same order in which the plan tasks are executed later. Furthermore, it uses domain-specific control knowledge that are defined in the form of HTN-methods. The configurable methods allows the planner not only to be customized for different planning domains but also to improve the performance of the planner with the specific domain knowledge.

Markov Decision Process

The concept of Markov Decision Processes (MDP) for solving a planning problem is to find an optimal policy for assigning an action to each world state. The optimal policy refers to the optimality of each assigned action over a specified period of consideration time. A MDP can be formally defined as a tuple (S, A, T, R, H) where S is a set of states; A is a set of actions; T is a transition function that defines probability distribution over next states by given current states and actions; R is called reward function that specifies the cost of executing each action by given states; H is a period of time over which generated actions in a plan must be optimal.

By introducing the transition funciton T, MDP is able to model actions with uncertainty. When the period of time H is finite, the outcome of a MDP planning task is a group of ordered policies along the time line. If H is infinite, the resulting policy remains the same. A standard method of finding an optimal policy is called *value iteration*. It includes two formulae for evaluation for an optimal policy [Bellman, 1957]. The first one calculates expected values associated with each state as the first step, then by using these values as inputs, optimal actions for each state can be selected among others when the actions has the minimum expected costs according to the second formula. In [Doshi et al., 2005], the workflow composition problem is modeled as a MDP task based on *value iteration* technique. The generated workflow is recoverable when Web service failures occur due to its non-deterministic nature in dynamic environments.

Another standard way of performing MDP is called *policy iteration*. It starts with an initial policy that is derived from rewards on states and computes a new policy according to the maximum expected utility value. Then the algorithm performs *value determinination* and *policy improvements* iteratively until resulted policies converge (stabilize) to the same one [Pashenkova et al., 1996]. The application of *policy iteration* can be found in [Reiff-Marganiec et al., 2009] where a WSC task is firstly decomposed into atomic tasks based on HTN. A set of resulted candidate plans are then evaluated by MDP so as to find the optimal one.

Model Checking

Model checking is a technique for validating finite state machine (FSM) against logical specifications. A FSM is a formal model used to describe a system by a specification language. It consists of states and transitions caused by actions. In order to verify the truth value of a property (planning goal) in a system that is specified by temporal logic formula, a model checker traverses through the FSM guided by the property. Once the property of the system is falsified, the model checker returns a sequence of states (also called counterexample) from the initial state explaining the reason of the falsified claim. Formally a model checking problem can be defined with an logical entailment $M \models \Phi$ where M is a FSM and Φ is a goal of M.

Planning as model checking describes its planning goal as a Linear Time Logic (LTL) formula that represents a system state. The negation of the LTL formula is given to a model checker as input together with a domain description. When the model checker falsifies the claim and returns a sequence of states, a sequence of actions can be derived as a plan [Berardi and Giacomo, 2000].

A formal planning domain as model checking can be defined as $\Sigma = (S, A, \gamma)$ where S is a set of states, A is a set of actions and γ is a non-deterministic state transition function defined as $\gamma : S \times A \to 2^s$. A plan π for a domain Σ is a set of (s, a) where $s \in S$ and $a \in A(a)$. Further requirement includes: $\forall s$ there is at most one action a such that $(s, a) \in \pi$ [Peer, 2005b].

A more efficient way of performing a traversal through state space is by applying data structure such as binary decision diagrams (BDD) so that a model checker treats a group of states rather than one individual state at one step. The first planning system based on model checking and BDDs is MIPS [Edelkamp and Helmert, 2001] that is designed for deterministic domains. Another planner called MBP (Model Based Planner) is able to calculate plans in non-deterministic domains including uncertainty of initial states, action effects and states in which actions are executed [Pistore et al., 2003].

2.5 Knowledge Based Compositions

2.5.1 Recommendation Oriented

The idea of recommendation as composition is to assist stakeholders to complete a composite workflow with recommendations of software components, a workflow or other kind of suggestions for workflow adjustments. The recommendation mainly relies on a knowledge base (KB) that contains plenty of service artifacts related to workflow completion.

Composition as a Service (CaaS) introduced in [Blake et al., 2010] is a service where clients are able to get recommendations about how to finish an incomplete business process or how to realize an abstract workflow with concrete data, services or workflows. The KB of CaaS is responsible for aggregation of concrete service related data that can be queried by the CaaS application. A submitted abstract workflow from a client is matched against predefined graphs in a data structure called *service net*, then each matched graph is used to query the KB for a concrete implementation details of the graph. The combination of the graphs are then send back to clients as recommendations.

In [Kim et al., 2004], workflow editors can also be assisted by a system called Composition Analysis Tool (CAT) through suggestions of next actions needed for completing a workflow. The CAT's KB includes definitions of action representations and their input and output parameters in an ontological manner. The system uses a domain-independent algorithm to analyze the correctness and completeness of user-defined workflows and provides suggestions toward user's desirable properties.

2.5.2 Resource Synthesis Oriented

A resource synthesis task refers to offering service or resource composition advices on different granularity levels through knowledge-based decision support systems. In [Chen et al., 2003] a service composition framework uses ontologies for conceptualizing domain knowledge with semantic service descriptions. These service descriptions are fed to a logical reasoner during service discovery. This framework utilizes domain-specific knowledge in a KB for guiding composition of workflow specifications. The KB contains information about concepts, axioms and rules for purpose of advice provisioning based on initial descriptions of problems.

As an example of concrete resource synthesis, human resource composition based on a description logic (DL) framework is discussed in [Colucci et al., 2005]. The framework takes skill profiles of experts and task descriptions as input and applies inference techniques in DLs to an ontology of skills in order to synthesize a group of people that are competent to finishing a target task. If competence of people cannot fully cover required skills of a task, an answer will be given for describing the missing part of skills.

2.5.3 Action Oriented

This knowledge-based approach emphasizes on representation of the world state and manipulation of the state by means of actions. An example is the PKS system [Bacchus and Petric, 2003]. The system has a group of databases to represent the world state. These databases get updated by actions that modify the world state on knowledge-level instead of changing the world physically.

Four databases involved in the representation contain knowledge of different types. The first database K_f includes ground literals specifying positive and negative facts and formulae that describe function values. The second one K_w contains formulae describing states with boolean values. The truth values of the states can only be known at execution time of sensing actions. K_v extends K_w so that function values can be known at execution time. The last database K_x contains knowledge of disjunctive ground literals. For instance, in a group of ground literals (l_1 | l_2 | l_3), only one of the entries is true. In this way, incomplete knowledge of an agent can be expressed.

Actions in PKS are defined in a similar structure of PDDL with action preconditions and effects. A precondition is a fact that an agent has known in its knowledge base. An effect is an update to the knowledge base through adding new or deleting existing literals. Another way to update the knowledge base in the system is to add domain specific update rules that capture state invariants. Planning goals can be a simple query asking existence of a knowledge in the databases or a complex query consisting of conjunctions and disjunctions of simple queries or quantified query ranging over a set of knowledge literals.

The planning process of PKS is a process of evolving agent's knowledge during the updates caused by actions. The ability of including incomplete knowledge and modeling sensing actions are very useful features in partial observable domains. In [Martínez and Lespérance, 2004], sensing actions and physical actions are used to define information-gathering and world altering services respectively. A web service composition problem in a travel domain is demonstrated as a plan synthesis task for PKS . Users are able to express their goals with desirable constraints and get conditional plans from the planning system.

2.6 Related Planning Systems

The related planning systems capable of encoding planning tasks into knowledge domains while allowing user-defined preferences to be specified are extensively used in the deterministic part of IPC-5 (cf. Chap 2.4.3). The participating planners are evaluated in optimal and satisficing subtracks with planning domain categories such as simple preferences, qualitative and complex preferences, etc.

The evaluation criteria of the optimal subtrack is the number of solved problems and CPU-time. For the satisficing subtrack, planners are evaluated not only in terms of the number of solved problems and CPU-time, but also in terms of plan quality with satisfied preferences and constraints as measurement criteria. We focus on the planners participating in the latter subtrack in this section and discuss the ones that outperform their competitors in IPC-5.

Three planners, i.e. SGPlan5 [Hsu et al., 2006], MIPS-XXL [Jabbar et al., 2006] and HPlan-P [Baier and McIlraith, 2006], are rewarded due to their outstanding performance[29]. With PDDL3.0 as the planning language, they are representative for defining problem classes in different strengths. All these planners are able to combine several planning techniques for finding solutions.

The Metric-FF based SGPlan5 planner uses partial order and planning decomposition techniques. The general idea of the functioning mechanism of the planner is to divide a planning problem into subproblems. Each of the problems corresponds to a subgoal. The planner tries to solve each subproblem from the initial state and combine all found solutions of the subproblems to one solution without any inconsistency. The partitioning of goals leads search space reduction for each subgoal and hence increases the planning performance.

Depending on the complexity of a planning problem, the MIPS-XXL planner could apply two algorithms during a planning search. It first uses an internal cost-optimized best-first heuristic to try to find a solution. Once it fails, the planner applies a breadth-first search algorithm and uses hard disk to extend its available state space for continuing the search process. HPlan-P puts the combination of search techniques even further that users can choose pre-defined heuristics of the planner or define new heuristics by themselves.

It is worth noting that the three planners have a preprocessing procedure for translating PDDL3.0 to another planning representation language. SGPlan5 translates a PDDL3.0 problem into MDF (Multi-valued Domain Formulation) for deriving accurate heuristic guidance. The MIPS-XXL planner compiles PDDL3.0 preferences into PDDL2.1 at first and then begins searching solutions with given preference costs defined in PDDL3.0. HPlan-P transforms PDDL3.0 planning problems to new domain predicates. The satisfaction of preferences and constraints become a job of achieving optional goal conditions in these domain predicates.

29 Evaluation results of IPC-5: `http://zeus.ing.unibs.it/ipc-5/results.html` accessed in Jul. 2012.

2.7 Ruby on Rails

Ruby on Rails (short for RoR or Rails) is a full-stack web application framework written in the programming language Ruby. The term *full-stack* refers to the extend of the functionality this framework offers. This includes supports both on server and client sides such as database abstraction, template rendering, database querying, etc. RoR follows the principle Convention over Configuration (CoC) meaning that RoR uses pre-setup rules in order to reduce manuel configuration of users. Another principle that RoR encourages is DRY (Don't Repeat Yourself) meaning that if a piece of code applied properly, an application will benefit from reduction of code duplication. In this section, a brief introduction about RoR is given.

2.7.1 Architecture of Ruby on Rails

MVC (Model-View-Controller) is an architectural pattern that provides a clear-cut separation of an application, i.e. data access, presentation and control flow. The three aspects make up the core architecture that RoR is based on:

- *model*: in RoR, a model is used for processing data of an application and managing rules of interactions with persistent storage. Each table in database of an application is usually represented by a model.
- *view*: a view provides the logic for displaying the data processed by a model. It is responsible for directly interacting with the user through web browsers or other platforms by means of HTML websites with embedded Ruby codes.
- *controller*: a controller is in charge of processing requests received by a web server. For static websites, it decides which view is to be presented to user; for dynamic websites, it interacts with models for data processing and then invokes a view to web browser.

RoR implements the MVC with its three corresponding components respectively, i.e. Active Record, Action View, Action Controller. The Action View and the Action Controller make up Action Pack[30] that responses a web request in two steps, i.e. firstly performing a CRUD action and then rendering a view or redirecting to another action. An overview of how the MVC works in RoR applications is depicted in figure 2.7.

Active Record provides means for processing persistent data and is also an ORM (Object Relational Mapping) framework. It provides functionalities such as mapping between class and table, CRUD operations to database, data validation, etc. Action view encompasses logic for

30 Action Pack: `http://ap.rubyonrails.org/` accessed in April 2012.

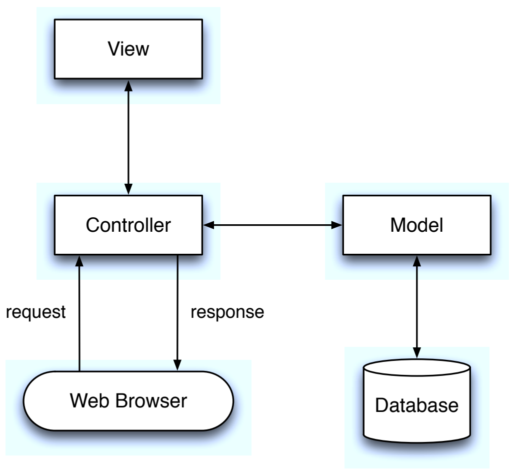


Figure 2.7: Architectural overview of RoR

presenting data from Active Record. This includes templates, layout and helpers[31], etc. Action Controller orchestrates flow of application logic and provides other functionalities such as session handling, caching, etc.

2.7.2 Ruby on Rails with REST

The principles of REST are generally manifested in Rails in terms of representation of resources and transfer of representation state of resources. In concrete these are reflected by the supports of CRUD and routing system. Rails is initially created to support CRUD style operations on resources. The HTTP verbs in REST can be directly mapped into the CRUD actions that are defined in controllers of Rails.

Assuming a system contains information of a group of articles (cf. Table 2.3), Rails uses a model-controller-stack to make a resource RESTful. The article resource refers to an article model and an article controller that includes a set of CRUD actions[32]. In addition to it, named routes appertaining the controller receive requests in the form of URLs and trigger the corresponding actions in the controller if the URLs are valid. Rails automatically generates the named routes associating with a set of standardized CRUD-based actions.

In Rails, different representations of a resource can be made available to clients based on their requests. HTML is the most common representation format when a resource is presented to a

31 A helper is a Ruby file that contains code of functions invocable within RTHML pages
32 Each controller is a Ruby class and each controller action is a method in the class.

HTTP methods	Request-URL	Rails Action	Purpose
GET	/articles	index	show all articles in a list
GET	/articles/new	new	show a page for creating an article
POST	/articles	create	create a new article
GET	/articles/1	show	show the article with id 1
GET	/articles/1/edit	edit	show a page for editing the article with id 1
PUT	/articles/1	update	update the article with id 1
DELETE	/articles/1	destroy	delete the article with id 1

Table 2.3: RESTful resource in Rails

client. Other formats such as XML, JSON, etc. can be defined in Rails with ease as well. They can be defined as default formats of a resource and be placed inside the function `respond_to`. Assuming a controller action `index` exposes a group of articles in a database table as a resource. Its representation of the resource (HTML, XML, JSON) can be rendered depending on the URLs the client requests:

```
def index
  @articles = Article.all
  respond_to do |format|
    format.html
    format.xml  { render :xml => @articles }
    format.json  { render :json => @articles }
  end
end
```

The router of Rails is responsible for identifying incoming URLs and dispatching them to controller's actions if ever defined. In the example above, the router can recognize the URL `/articles` and dispatch this request to the action `index`. It distinguishes the overlapped URL `/articles` (cf. table 2.3) by identifying the different HTTP request methods. Declaring the group of articles as RESTful resources can be done by only defining the routing method `resources` in the Rails file `routes.rb` with the statement `resources :articles`. After the resource declaration, the router generates four named routes (in the example of articles, the four routes are `/articles`, `/articles/new`, `/articles/1`, `/articles/1/edit` pointing to the seven controller actions.

As a matter of fact, the names of the controller actions can be chosen arbitrarily, but the discipline of adopting the convention of Rails, especially the REST convention in Rails, provides best practices for developers and offers users a RESTful interface automatically [Fernandez, 2010].

CHAPTER 3

Service Design

3.1 Objectives

In the light of the theories related to AI planning techniques introduced in the chapter 2, the following chapters in the thesis aims to reach two basic objectives. First, by demonstrating a planning problem in a real-world scenario, composite services should be generated by applying a domain independent planning system in different planning situations. This involves a modeling process of the planning domain and decision making of a proper planning system that is able to generate plans for tackling user-specified problems. A service can therefore be co-created by a domain expert that is in charge of the modeling and a domain user that defines his planning desires. The service should be technically online reachable so that service users can make use of it across the web.

Beyond the scenario, a service system in the form of a web application is proposed to serve as a delivering platform for providing knowledge intensive planning services as well. The knowledge-based planning techniques should be the key enabling factor for the service system. The knowledge needed in the system stem from the expertise of domain experts and users. This includes structures of domains, search-control knowledge, constraints, etc. The service system should be capable of offering functionalities such as creation and maintenance of the knowledge domains provided by domain experts. A domain independent planner allows the service system to remain neutral for tackling various planning tasks in different domains.

3.2 Learning Design Use Case

Learning Design (LD) (or instructional design) is a process in E-Learning where software is used to support sequencing Learning Objects (LOs) in order to ensure learners achieve specified learning outcomes. Researches such as [Castillo et al., 2010; Morales et al., 2008; Garrido et al., 2011], etc. adopted AI Planning techniques for automating the construction of LD. The key purpose of the AI planning in the context of LD is to generate learning routes for students such that each sequence of LOs is a tailored solution to the students with different goals, educational backgrounds and needs, etc. However, previous researches on the LD issue lack of specific control knowledge in planning domains. Users are not able to define preferences on their LOs.

For effectively modeling a learning scenario, *task analysis* [Jonassen et al., 1999b] can be used for ascertaining domain details on both sides of domain experts and domain users. Task analysis is a vital step in LD for an in-depth analysis of learning in order to assist instructional designers to describe what and how learning objectives can be accomplished by learners. A task analysis can be performed in two phases. The task description phase is of identifying, refining and ordering tasks that can together make up a planning domain. In the instructional phase needs and objectives of users are specified. This corresponds to defining goals in a planning problem.

In the following section we focus on modeling a LD scenario and introducing the capability of personal customization on the side of users. After encoding the LOs of the scenario as a knowledge domain in a planning domain file, a student is able to obtain a customized learning route according to the specified knowledge competence and goals stated in his profile. The modeling language we choose is PDDL3.0 by which we can express problems with preferences, i.e. soft goals or soft constraints. As stated in Chapter 2 it has the ability to distinguish importance levels and computational costs of different soft goals and allows planners to achieve only a subset of them in a planning problem when conflicts or computational overload exists. Preferences are weighted in a plan metric assigned with numerical penalty values according to their importance.

3.2.1 Domain Analysis

We choose a domain of web development area as our planning domain for LD task. Web development is a multidisciplinary area ranging from client-side to server-side scripting, from markup to coding and from developing static web pages to highly complex web applications. It is difficult for newcomers to acquaint themselves with the technologies related to this area. We focus in this section on constructing a domain for learners that want to be competent in a subset of web development area with some specific technologies.

The proposed LD task is to teach web development skills to students with different learning backgrounds. The skills can be broadly broken into two major learning objectives, i.e. being able to publish static websites and to publish dynamic web applications (short for webapps). Being able to publish websites is also an essential step for publishing webapps. There are many technologies available for helping people getting competent to publish websites and webapps. To simplify the LD, we restrict them to the most basic ones that include the programming language Ruby and the frameworks based on it. The outline in Table 3.1 briefly describes a set of tasks that should be performed in order to get the competence of skills for reaching the two objectives. It is a vital step to be taken for underpinning further analyzes of the domain and eliciting the knowledge needed for instructional purpose.

Task	**Description**
HTML4	A standard markup language for structuring web contents.
HTML5	The future markup language that will replace HTML4[1].
Haml	An efficient light-weight markup language for describing XHTML of web contents[2].
CSS2	A standard style sheet language for describing presentation style of web documents.
CSS3	The latest standard for CSS that is still under development by W3C just as HTML5
Sass	A scripting language used as extension of CSS3 and provides syntax similar to Haml[3].
Hosting	Making websites accessible on the Web.
DNS	Domain Name System (DNS) translates domain names into IP addresses.
HTTP	The basic communication protocol for delivering web resources.
Javascript	A scripting language that is widely used on the client-side development of dynamic websites.
JQuery	A Javascript library for simplifying scripitng the scripting of HTML[4].
Ruby	A object-oriented scripting language.
Nanoc	A Ruby-based web publishing system for generating static websites[5].
Rails	A Ruby-based full-stack web application framework.
Git	A version-control software that comes in handy on deployment
Heroku	A cloud platform for deploying web applications.
Sinatra	A small and flexible Ruby-based web application framework[6].
MySQL	A widely used database system that offers good supports for Ruby and RoR.
PostgreSQL	Another database system with advanced features and offers support for Ruby and RoR.
website	Putting related knowledge together as a practice in order to publish websites.
Webapp	Developing a web application based on the related knowledge and skills.

Table 3.1: Tasks of Web Development Domain

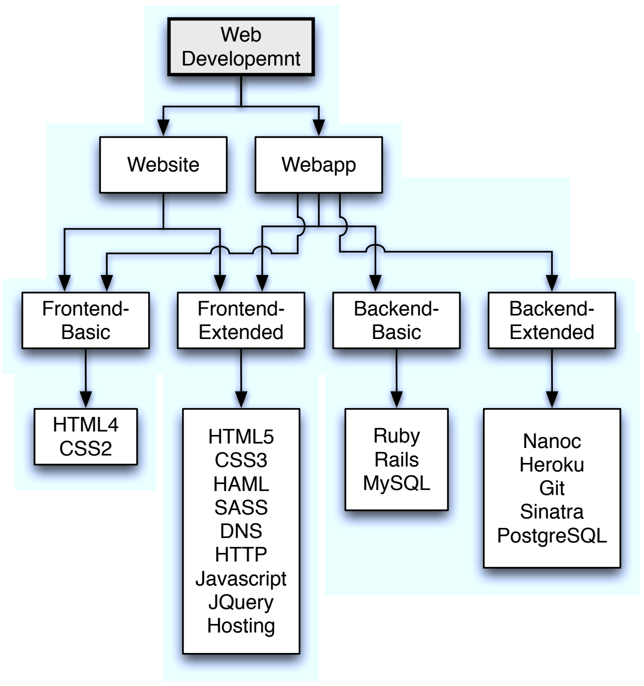


Figure 3.1: Taxonomic Hierarchy of Learning Objects

After investigating the knowledge of contents derived from the tasks in the Table 3.1, we can realize that not all of them must be gotten through by students to be able to publish websites and webapps. However, the more knowledge students learn, the more competent they get in the related objectives. By categorizing them as LOs for our LD use case, a tree structure of the LOs is illustrated in Figure 3.1.

In Figure 3.1, the leaf nodes at the bottom of the hierarchy contain the task units that a student should perform. The essential LOs can be identified in `Frontend-Basic` and `Backend-Basic` categories. Other LOs in `Frontend-Extended` and `Backend-Extended` are optional but necessary for increasing the competence of student's skills in developing websites and webapps. In order to publish them, two additional LOs are designed specifically for assembling the skills together through exercises, i.e. `website` and `webapp`.

1 Until March 2012, HTML5 is still a draft: `http://www.w3.org/TR/html5-diff` accessed in March 2012.
2 HAML: `http://haml.info/` accessed in April 2012.
3 SASS: `http://sass-lang.com/` accessed in April 2012.
4 JQuery: `http://jquery.com/` accessed in April 2012.
5 Nanoc: `http://nanoc.stoneship.org/` accessed in April 2012.
6 Sinatra: `http://www.sinatrarb.com/` accessed in April 2012.

The model of the proposed planning domain for web development is defined as a deterministic FSM so as to be applicable for a planning system. In order to further elicit the knowledge of the domain and find out the relations of the stated tasks, we adopt the Conceptual Graph Analysis (CGA) as the method of task analysis for LD [Jonassen et al., 1999a, chap. 20] and define the domain as a quintuple:

$$\Sigma = (P, P_0, G, T, \Gamma)$$

where

- P is a set of possible student profiles indicating the world states.
- P_o is a student profile representing the initial planning state, $P_o \subset P$.
- G is the learning goal state specified by a student. It is the final state (a new profile) produced by execution of a learning route. $G \subset P$.
- T is the set of tasks in LD stated already in the Table 3.1.
- $\Gamma : P \times A \longrightarrow 2^P$, Γ is a deterministic FSM that changes the state of a student profile to other one by applying a task.

Each student profile models the student's learning data. A planner should generate a learning route leading a student from the initial profile derived from his learning history toward the final profile defined by the student himself. This transition of the profiles is always an improvement of competence levels of skills after following a learning route. We characterize a profile by the following attributes:

- Competence levels: this is specified through a student's previous learning history and is measured numerically for quantitative precision. For the initial profile, it is proposed that the more precise the specification of competence levels from the learning history is, the more accurate a learning route is produced for each individual student.
- Soft skills: it is characterized by language levels that are assigned to a student through previous assessments. This effectively influences the learning materials selected by a planner for the student based on his language ability.
- Learning styles: this refers to the way a student approaches his learning objectives. There are many models for categorizing learning styles. We introduce two of them from Honey and Mumford's model, i.e. *theoretical* and *pragmatic*[7] [Honey and Mumford, 1986].

7 A *theorist* tends to get knowledge through sound theories and structured situations. This type of student learns best when LOs are backed up by theories. A *pragmatist* prefers to apply concepts to practical jobs. This type of student learns best when LOs offer opportunities for trying out new techniques [Honey and Mumford, 1986].

- Learning resources: since a LO may require a student to have specific software installed on his local machine, e.g. a operating system, the student needs to specify the learning resources he currently has. If the learning resource of the student is not the one required by a LO, a planner should find an alternative LO instead.
- Learning time: the amount of time is also calculated from the learning history of a student. Since modeling time is a complex task that involves scheduling techniques, for simplicity reason, we choose floating point number to model discrete time and the learning time is the total amount of time used till now.

A CGA for the LD domain consists of two types of elements, i.e. *states* and *tasks*. States are the prerequisites and learning outcomes of tasks. Every state has its own competence levels of skills and is evaluated numerically as mentioned before. The numeric values not only depend on the initial values specified in a student profile prior to a planning process but also depend on the accumulation of related learning outcomes during the planning process. A list of competence areas can be deduced as follows (cf. Table 3.2):

Competence Area	**Related Tasks**
structure web content[8]	HTML4, HTML5, HAML
format web content[9]	CSS2, CSS3, SASS
frontend programming	Javascript, JQuery
backend programming	Ruby, Nanoc, Sinatra, Rails, webapp
database	MySQL, PostgreSQL, Sinatra, Rails
web application framework[10]	Rails, Sinatra
deployment	Hosting, DNS, HTTP, Heroku
version control	Git, webapp
publish website	website
publish webapp	webapp

Table 3.2: Competence Areas of The Learning Domain

The overlaps of the related tasks in some competence areas indicate that the execution of certain tasks can increase levels of several competence skills at the same time. For instance, "Sinatra" and "Rails" are courses that require additional knowledge of backend programming and database along with the knowledge of the frameworks themselves. Once these tasks are executed, students will improve their competence levels in these areas during the learning. The same rule also applies to the course "webapp", since by learning how to develop and deploy a web application,

8 Short for *structure content* in the rest of the thesis.
9 Short for *format content*.
10 Short for *framework*.

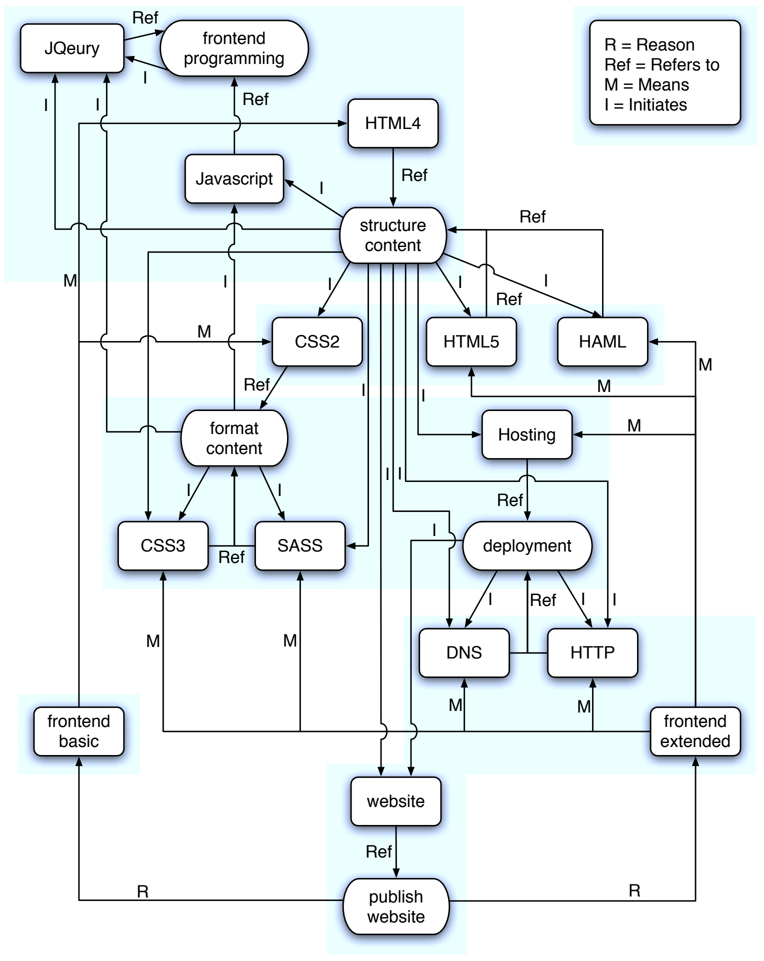


Figure 3.2: Conceptual Graph Analysis for Subdomain Website

students engage their skill improvements not only in the knowledge for publishing webapps, but also in the areas of version control and backend programming.

The learning tasks identified from a CGA represent the LOs that constitute a learning route for each student. They play an important role in the entire LD process. A task could be a tutorial, a lesson, a course or an exercise. A successful execution of a task is then considered to be passing this educational unit with new generated learning outcomes when its prerequisites are satisfied. This means, before the execution, a student must reach the required competence levels and after the execution, task-related competence levels are increased. With the states and tasks in hand, a rudimentary CGA depicts their relations in the case of publishing websites in Figure 3.2.

This conceptual graph has two types of information (stated as nodes) bound together by four types of relationships (stated as arcs). The goal state of gaining competence of publishing web-

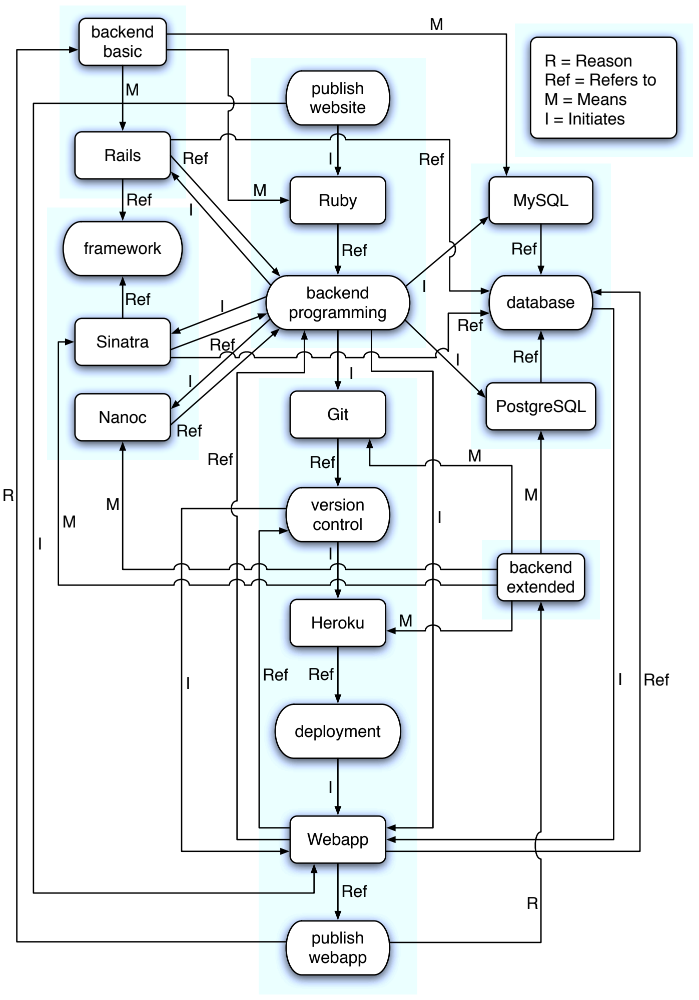


Figure 3.3: Conceptual Graph Analysis for Subdomain Webapp

sites is connected to the subgoals, i.e. accomplishing basic and extended courses of frontend development, by the "R" arcs. The "R" signifies that the reason of existence of the two subgoals is for reaching the major goal state. The goal of being able to publish website is divided into the subgoals of working through the related basic and extended courses of frontend development. Achieving each of the subgoals means (indicated by "M") executing its underlying tasks. In the case of basic frontend courses, this entails the successful passing of HTML4 and CSS2 as categorized in Figure 3.1. After execution of the tasks, their results refer to (abbreviated as "Ref") the increments of the corresponding competence levels and hence transforms the current knowledge state of a student into new one. For instance, when a student completes the course of Javascript, his competence level of frontend-programming will be increased. The "I" arc indicates the current state of a competence level initiates the tasks of which the prerequisites are satisfied.

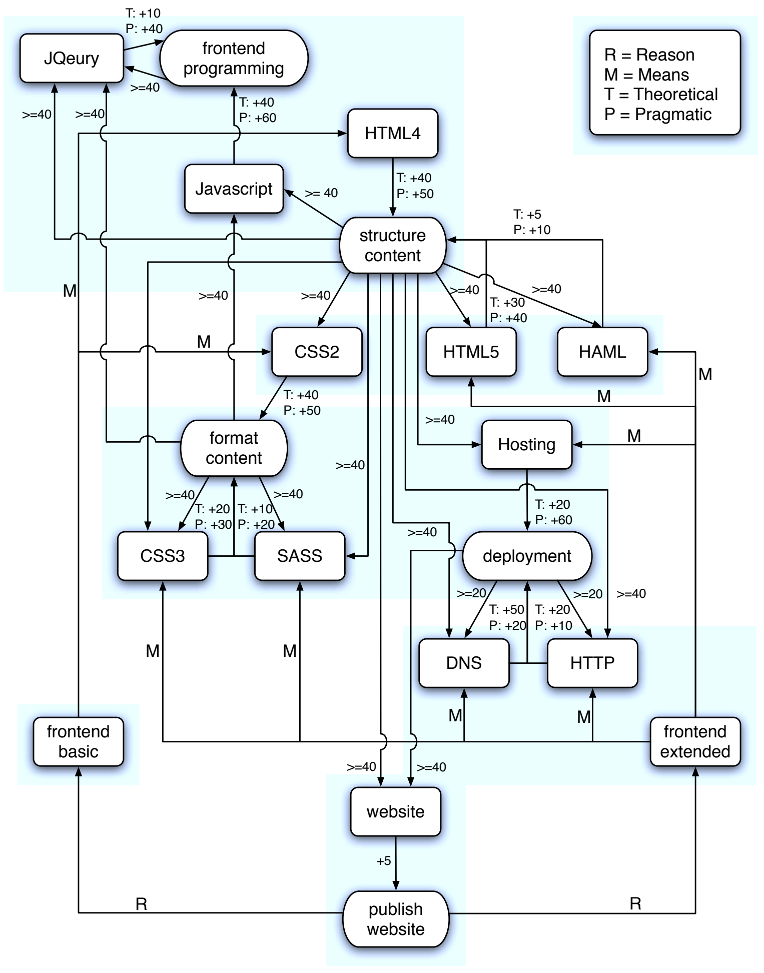


Figure 3.4: Quantized Conceptual Graph Analysis for Subdomain Website

Using the same technique of CGA, tacit knowledge of relations among the LOs for publishing webapp can be elicited through Figure 3.3. The goal "publish webapp" is the reason of the two subgoals "backend basic" and "backend extended". For the former one, e.g. it means the completion of its related courses "Ruby", "Rails", "MySQL". It is worth noting that after executing some tasks in the two CGA graphs, the corresponding competence levels that were previously the learning prerequisites are increased as learning outcomes. For instance, the participation of the course "Rails" requires knowledge of "backend programming". After finishing the course, a student increases the competence level in "backend programming".

Since each student approaches his learning goals in different learning styles, i.e. theoretical and pragmatic, different learning outcomes can be produced. For simplicity, we assume completing a course that emphasizes on concepts and theories yields more competence for a theoretical student, completing a practical course yields more competence for a pragmatic student. We

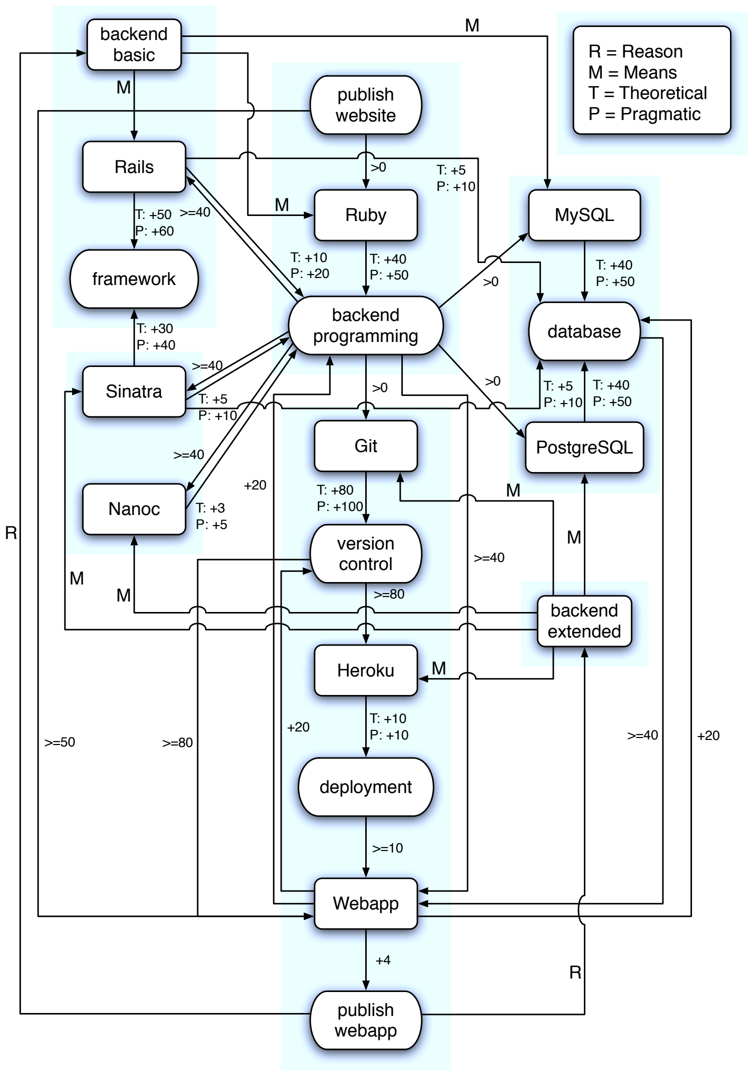


Figure 3.5: Quantized Conceptual Graph Analysis for Subdomain Webapp

further quantize the relations "Initiates" and "Refers to" among the states and tasks based on the previous two CGAs in Figure 3.4 and Figure 3.5. The numerical values denote the competence levels in percentage.

3.2.2 Planner Selection

The reason of selecting a planning system prior to creation of the LD domain in PDDL is that most of the planners cannot support all constructs of PDDL and many of the planners interpret some of the constructing elements in slightly different ways or even interpret them incorrectly. In order to avoid potential problems and obstacles with regard to the interpretation of the planning language, a planner should be selected at first. It should support standardized PDDL with features allowing customization of learning routes for users. Besides this, it should have fluents that can represent numeric changes of competence levels as investigated in the previous section.

The required features of the planning language that support modeling of fluents and preferences restrict the choice of planning systems to the ones that are PDDL3.0 (or higher) compliant. The planners SGPlan5, MIPS-XXL and HPlan-P discussed in Chapter 2.6 obviously become the candidates that could solve the LD problem.

Regarding to the source code of the planners, MIPS-XXL[11] is compilable under Ubuntu 10 and Mac OS X 10.6.8 (as it is tested during the time of the thesis), while SGPlan5[12] is only executable on Ubuntu. The source code of HPlan-P[13] is written in C. However it is not up to date hence not compilable under the previous mentioned operating systems without modification of related libraries.

For the planning task in the LD domain, we choose SGPlan5 as our primary planner since it fully supports all features of PDDL3.0. Its outstanding performance that has been proven in IPC-5 is also a key factor for the decision we made. From the analysis of the LD domain we can realize its PDDL representation requires features like preconditions with numeric fluents and conditional effects. We examined SGPlan5 and MIPS-XXL in terms of the ability of interpreting simple domains with these features. SGPlan5 can generate plans according to the domains and problems we defined. MIPS-XXL in many cases cannot interpret the domains and leave plan outputs blank without any error indications.

3.2.3 Creating the Domain

Creating the planning domain in PDDL3.0 involves formal interpretation of two types of information exposed in the domain analysis (cf. section 3.2.1). Since each competence level is evaluated numerically and varies over the execution time of a learning route, it should be interpreted as *fluent* in the PDDL domain. The counterpart of a learning task is self-explanatory and should be defined as an *action* in PDDL. For structuring an action, we use fluents as preconditions for quantitative qualification of students and conditional effects as results of an action for different learning styles. The BNF (Backus Normal Form) for an action [Fox and Long, 2003] in the domain is given below:

```
<action-def>  ::= (:action <action-symbol> :parameters () <action-def body>)
```

11 MIPS-XXL: `http://sjabbar.com/mips-xxl-planner`, accessed in Jul. 2012.
12 SGPlan5: `http://wah.cse.cuhk.edu.hk/wah/programs/SGPlan/sgplan5.html`, accessed in Jul. 2012.
13 HPlan-P: `http://www.cs.toronto.edu/tlplan/hplanp.shtml`, accessed in Jul. 2012.

```
<action-symbol> ::= <name>
<action-def body> ::= [:precondition <CD>] [:effect <EF>]
```

Condition descriptions `<CD>` include fluent `<f-exp>` and negative predicate expressions:

```
<CD> ::= (and <f-exp>* (not <predicate>))
<f-exp> ::= (<binary-op> <function-head> <number>)
<binary-op> ::= >=
<binary-op> ::= >
<number> ::= Positive numeric literal in integer
<predicate> ::= (passed <const>)
<const> ::= Constant names of courses
```

Effects `<EF>` include functional expression updates `<p-effect>` in conditional effects `<c-effect>` and positive predicates:

```
<EF> ::= (and <c-effect>* <predicate>)
<c-effect> ::= (forall (<variable> ) (when <predicate> (and <p-effect>*)))
<p-effect> ::= (<assign-op> <function-head> <number>)
<assign-op> ::= increase
<predicate> ::= (passed <const>)
<const> ::= Constant names of courses
```

For instance, the task "Javascript" has the prerequisites that the competence levels of "structure content" and "format content" are greater than or equal to 40%. After passing the course, "frontend programming" is increased by 40% for the theoretical learning style and 60% for the pragmatic one:

```
(:action javascript
  :parameters ()
  :precondition (and (>= (structure-content) 40)
                     (>= (format-content) 40)
                     (not (passed JS)))
  :effect (and (forall (?p - profile)
                  (when (theoretical ?p)
                    (and (increase (frontend-programming) 40)
                         (increase (total-time) 15))))
               (forall (?p - profile)
                  (when (pragmatic ?p)
                    (and (increase (frontend-programming) 60)
                         (increase (total-time) 10))))
               (passed JS)))
```

The predicate `(not (passed JS))` in the action's precondition and the `(passed JS)` in the effect are used to prevent the action from being executed once again when a student passes the course. It also avoids itself from being inserted into a learning route when `(passed JS)` is in a student's profile indicating that the student has already passed the course in his learning history.

The conditional effects increase the values of fluents `(frontend-programming)` and `(total-time)` according to the learning style defined in student's profiles (cf. Figure 3.4). Since the course "javascript" is proposed to contain practical learning materials, the learning outcomes are therefore favourable to the pragmatic students in terms of gaining more competence in less learning time comparing to the theoretical students.

Another important type of constructs in the domain is the actions "website" and "webapp". The preconditions of the two actions are similar to courses such as "Javascript", however the calculation of the corresponding competence levels of "website" and "webapp" is slightly different. For instance:

```
(:action website
  :parameters ()
  :precondition (and (>= (structure-content) 40)
                     (>= (deployment) 40) (not (passed WEBSITE)))
  :effect (and
            (forall (?c - frontend-basic)
              (when (passed ?c)
                (increase (publish-website) 25))
            (forall (?c - frontend-extended)
              (when (passed ?c)
                (increase (publish-website) 5))
            (increase (publish-website) 3)
            (passed WEBSITE)))
```

For every completed course in the "frontend-basic" and "frontend-extended" categories, the competence level of "website" is increased by 25% and 5% respectively. The reason is the more courses related to publishing website is completed, the more competent a student will get. Besides it, another reason is that finishing a course of "frontend-basic" is weighted much more than a course of "frontend-extended".

The language skills of students are modeled directly by the predicates `(technical-english ?l - level)` and `(technical-german ?l - level)` where the object `level` contains the constants `LOW`, `MEDIUM` and `HIGH` (c.f. Appendix A). With the predicates in hand, the same learning material written in different languages can be modeled as follows:

```
(:action html4
  :parameters ()
  :precondition (and
                  (or (technical-english LOW)
                      (technical-english MEDIUM)
                      (technical-english HIGH))
                      (= (publish-website) 0) (not (passed HTML4)))
  :effect (and
            (forall (?p - profile)
              (when (theoretical ?p)
                (and (increase (structure-content) 40)
                     (increase (total-time) 4))))
            (forall (?p - profile)
              (when (pragmatic ?p)
                (and (increase (structure-content) 50)
                     (increase (total-time) 5))))
            (passed HTML4)))
```

The semantic inside the construct `or` indicates the action `html4` requires the level of technical english is at least `LOW` for a student. When the same learning material is available in the German language and requires high level of the language skill, the predicate `(technical-german HIGH)` is inserted in the precondition of the action `html4-de`:

```
(:action html4-de
  :parameters ()
  :precondition (and
                  (technical-german HIGH)
                  (= (publish-website) 0) (not (passed HTML4)))
  :effect (and
            (forall (?p - profile)
              (when (theoretical ?p)
                (and (increase (structure-content) 40)
                     (increase (total-time) 4))))
            (forall (?p - profile)
              (when (pragmatic ?p)
                (and (increase (structure-content) 50)
                     (increase (total-time) 5))))
            (passed HTML4)))
```

Akin to the language skills, the learning resources such as the essential software installed on a local computer of a student is defined as predicates as well. For instance, `(operating-system ?s - system)` has three grounded predicates with the constants `WIN`, `MAC` AND `LINUX`. These are used for the planner to distinguish the LO "Git" with distinct contents for the three different operating systems.

3.2.4 Testing the Domain

The structure for a planning problem falls into three parts in a problem file, i.e. `init`, `goal` and `metric-spec`. The BNF for a complete problem specification in PDDL is given as follows:

```
<problem> ::= (define (problem <name>) (:domain <name>)
               <object declaration> <init> <goal> <metric-spec>)
<object declaration> ::= (:objects (profile (name)))
<init> ::= (:init <init-el>* )
<init-el> ::= <predicates>
<init-el> ::= (= <function-head> <number>)
<goal> ::= (:goal <GD>)
<GD> ::= (and <f-exp>* <predicates>* (preference [name] <f-exp>)*)
```

```
<f-exp> ::= (<binary-op> <function-head> <number>)
<binary-op> ::= >=
<binary-op> ::= >
<binary-op> ::= =
<binary-op> ::= <=
<binary-op> ::= <
<number> ::= Positive numeric literal in integer
<predicate> ::= (passed <const>)
<const> ::= Constant names of courses
```

Based on this structure, the domain for LD in Appendix A is tested by three fictive students with the following characteristics in the profiles:

- Peter: theoretical learning style; with low English level and high German level; has Mac installed on computer; has no learning experience related to website and webapp before.

- David: pragmatic learning style, with high English level and low German level, has Windows installed on computer; has finished some courses related to publishing websites.

- Caleb: pragmatic learning style, with high English level and low German level, has Linux installed on computer; has no learning experience related to website and webapp before.

Peter wants to reach the competence level of structuring and formatting web contents to 60% and prefers to finish the "Sass" course. His profile in the problem file (also cf. Appendix B.1) and the learning route generated by SGPlan5 are shown as follows:

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
 Peter - profile)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 0)
  (= (format-content) 0)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 0)
  (theoretical Peter)
  (technical-english LOW)
  (technical-german HIGH)
  (operating-system MAC))
(:goal
  (and (>= (structure-content) 60)
       (>= (format-content) 60)
       (preference p1 (passed SASS))))
(:metric minimize (*100 (is-violated p1))))
```

Listing 3.1: Profile of Peter

```
0.001: (HTML4-DE) [1]
1.002: (CSS2) [1]
2.003: (SASS) [1]
3.004: (HTML5) [1]
4.005: (CSS3) [1]
```

Listing 3.2: Learning Route for Peter

David is already 50% competent in structuring and formatting web contents. He wants to be at least 80% competent in the version control, meanwhile he prefers to be at least 60% competent in structuring web contents in less than 40 hours. These two preferences are weighted by the same penalty values indicating the same importance of reaching these two soft goals. David's profile and his learning route are shown below (cf. Appendix B.2):

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
  David - profile)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 50)
  (passed HTML4)
  (= (format-content) 50)
  (passed CSS2)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 10)
  (pragmatic David)
  (technical-english HIGH)
  (technical-german LOW)
  (operating-system WIN))
(:goal
  (and (>= (version-control) 80)
       (preference p1
          (>= (structure-content) 60))
       (preference p2
          (<= (total-time) 40))))
(:metric minimize (+ (*1 (is-violated p1))
                     (*1 (is-violated p2)))))
```

Listing 3.3: Profile of David

```
0.001: (HOSTING) [1]
1.002: (WEBSITE) [1]
2.003: (RUBY) [1]
3.004: (GIT-WIN) [1]
4.005: (SINATRA) [1]
5.006: (HAML) [1]
```

Listing 3.4: Learning Route for David

Caleb wants to be proficient in publishing websites with a competence level of 90%. Besides that, he wants to be at least 60% competent in publishing webapps and 100% proficient in webapp deployments. His profile and personal learning route are as follows (cf. Appendix B.3):

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
 Caleb - profile)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 0)
  (= (format-content) 0)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 0)
  (theoretical Caleb)
  (technical-english HIGH)
  (technical-german LOW)
  (operating-system LINUX))
(:goal
  (and (>= (publish-website) 90) (>= (
      publish-webapp) 60) (preference p1 (>=
       (deployment) 100))))
(:metric minimize (is-violated p1)))
```

Listing 3.5: Profile of Caleb

```
0.001: (HTML4-VIDEO) [1]
1.002: (HTML5) [1]
2.003: (CSS2) [1]
3.004: (JAVASCRIPT) [1]
4.005: (SASS) [1]
5.006: (HAML) [1]
6.007: (CSS3) [1]
7.008: (HOSTING) [1]
8.009: (HTTP) [1]
9.010: (DNS) [1]
10.011: (WEBSITE) [1]
11.012: (RUBY) [1]
12.013: (MYSQL) [1]
13.014: (RAILS) [1]
14.015: (GIT-LINUX) [1]
15.016: (HEROKU) [1]
16.017: (WEBAPP) [1]
```

Listing 3.6: Learning Route for Caleb

CHAPTER 4

System Analysis

4.1 Requirement Analysis

The proposed prototype system is in the first place a service system responsible for storing and administrating knowledge domains as resources. The system provides users (including the providers of knowledge domains) the ability of accessing resources of others. The provider of a knowledge domain is considered as a *domain expert* due to the expertise the provider possesses in the domain. Moreover, a *domain user* can explore knowledge domains of interest in the system and utilize them by means of specifying the user's own planning goals and getting personal plans from the system hereafter.

The knowledge domains in the system repository are described by PDDL. A PDDL action is self-descriptive with its action prerequisites and outcomes. Since actions are backbones of a planning domain, the readability of actions not only allows domain providers to extend their knowledge domains with relatively small endeavors, but also allows normal users that do not have the corresponding domain expertise to understand the semantics behind the domains and define appropriate planning goals on their own.

The service system itself offers a platform as a service where domain experts can make their PDDL domains useful on the web. They should be able to perform administrative tasks for their published resources in the system. As for utilizing the resources provided by others, a domain expert acts in this case as a domain user and has the same abilities to perform operations on resources as a normal user can do.

For domain users, a knowledge domain published by the system is reusable in the sense that it can be applied to different planning goals for the different users. The public available knowledge domains are representations of resources related to their domains. Hence, from a user's perspective, the representation of domain knowledge and the generated plan based on them can be viewed as two major services provided by a domain expert and the prototype system.

For plan generation, a planning system is needed to be integrated into the service system. Service compositions in this context refer to two aspects. The first aspect is combining and sequencing PDDL-actions of a knowledge domain toward to a user's goal. Each action is considered as a service unit that updates the state of the domain once executed. Another aspect is the ability of composing new knowledge in the form of a plan to help a domain user to achieve his goals.

The system requirements are briefly summarized below.

- Domain experts are able to publish and maintain their knowledge domains that are accessible for domain users.
- Published knowledge domains are stored in the file system of the prototype system and readable for the planner. The domains represented by PDDL can be read by users as resources and applied to planning tasks.
- Planning goals can be defined in problem files by domain users. The users are able to publish and maintain their problem files for the purpose of diverse service compositions.
- A planner is integrated in the system for enabling the plan generation.
- The service system is able to administrate the existing resources and authorize system stakeholders with different privileges based on an authentication policy.
- Graphical user interface should be presented to system stakeholders that participate in the value co-creation of the system.

4.2 System Use Cases

The service system involves five actors including human users and a planning system, i.e. administrator, domain expert, domain user, anonymous user and SGPlan5. A human user must authenticate himself before managing resources in the system. An Administrator can (but not must) be generalized as domain experts and domain users since he also has the authority of producing knowledge domains and making use of them as the others. Both a human user and the planner SGPlan5 involve the generation of a plan, since a planner cannot produce a plan without the user's approval. An UML use case diagram gives an overview of the system (cf. Figure 4.1). We break down this use case in the overview by concrete descriptions as follows.

In the system only administrators are authorized to manage user resources after an authentication process. The prototype system allows administrators to delete users and their associated resources (cf. Figure 4.2). After a user logins as an administrator, the system offers a listing

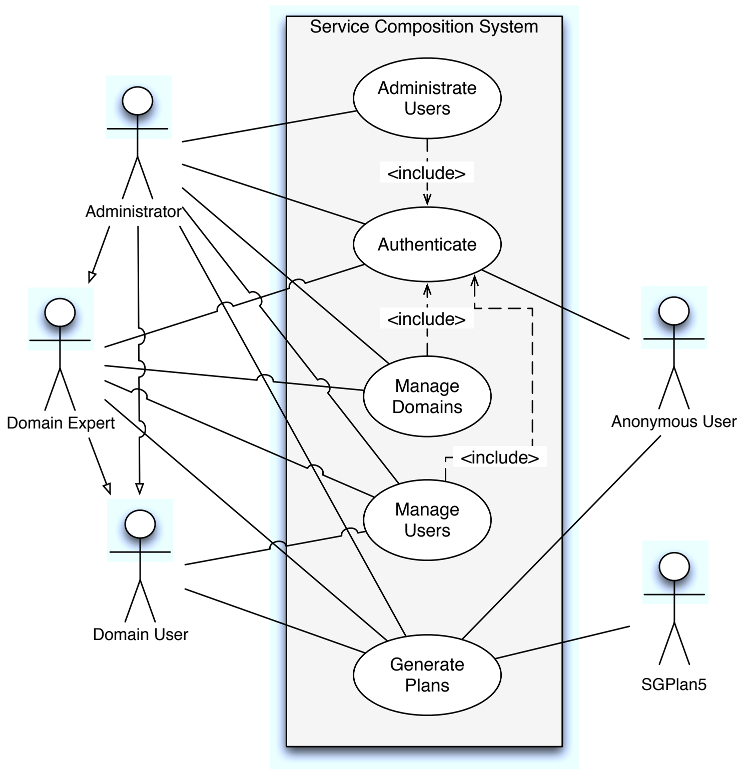


Figure 4.1: Use Case - System Overview

of all registered users that the admin can browse in his session. This listing should differentiate the one of other users that a delete link should appear after each user's name (that is again hyperlinked to each user's profile).

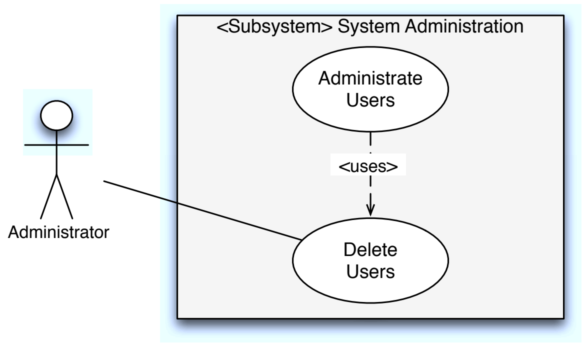


Figure 4.2: Use Case - Administration

The use case "authenticate" involves creating user accounts, signing in and signing out these accounts. Anonymous users must authenticate themselves at first in order to manipulate certain resources that are restricted to be accessed for unregistered users (cf. Figure 4.3). For completing a signup process, a user must enter his account name in a signup form along with a valid email address and a password. The password must be typed in twice in order to avoid possible typos.

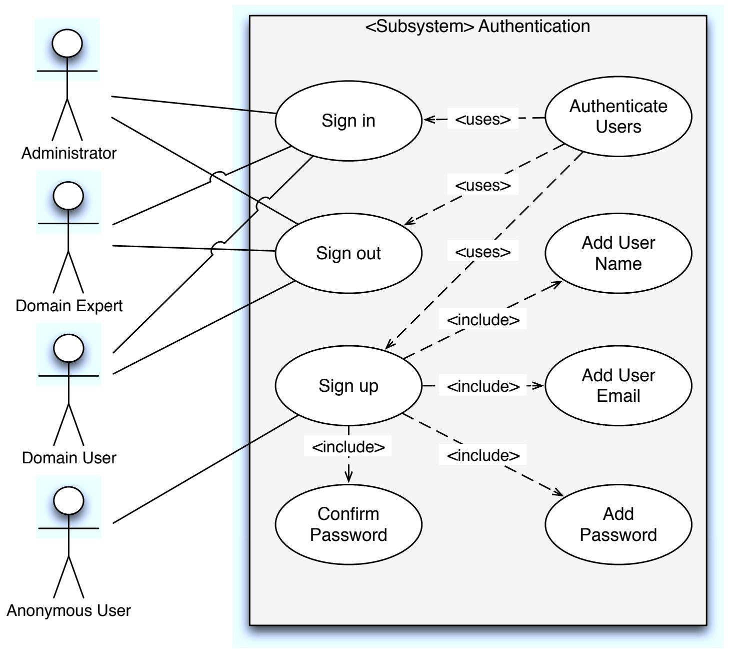


Figure 4.3: Use Case - Authentication

Managing resources in the system is a relatively complex task comparing to other system features. The resources can be mainly grouped into knowledge and user resources. The knowledge resources are represented as knowledge domains that are stored persistently in the file system of the prototype. They can be created, edited, rendered and deleted by domain owners. For creating a domain, a user must pass an authentication process mentioned before. Adding a domain name and a description for each domain are essential steps required by the system. A knowledge domain must be constructed offline by a domain expert before uploading to the system. After uploading the domain, the domain owner can edit the domain information that he created before. If an existing domain file needs to be updated, it must at first be deleted by its domain owner and a new one can then be uploaded afterwards.

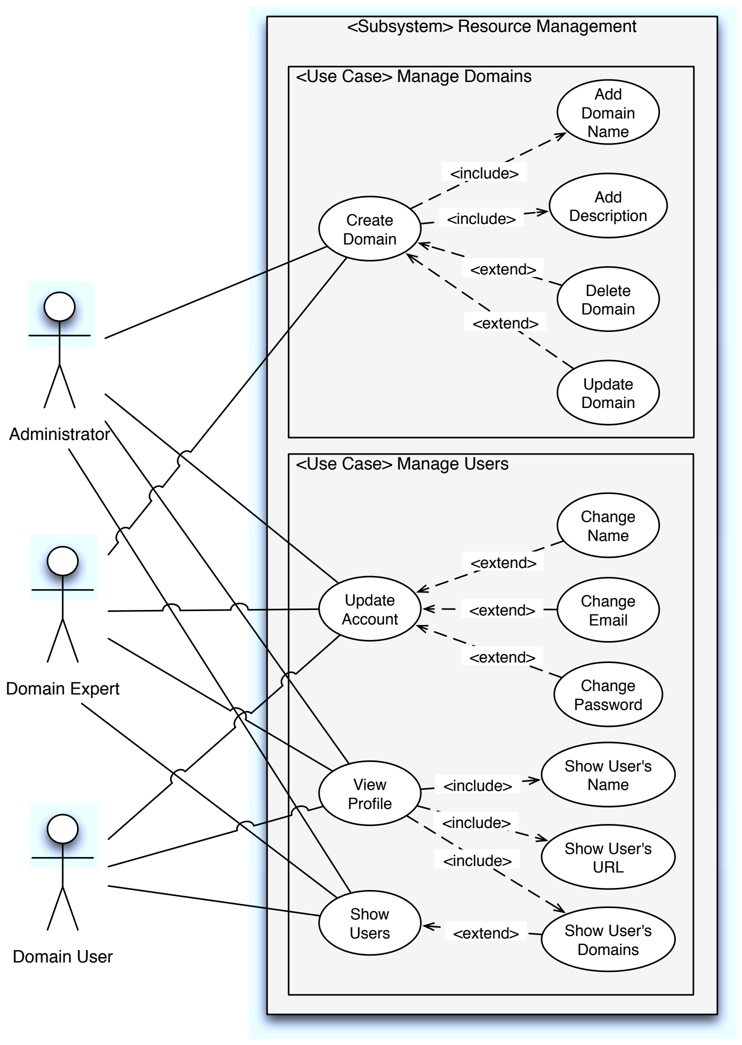


Figure 4.4: Use Case - Resource Management

The user resources refer to the information of user' accounts and the information about the associated relations to knowledge domains. Account information can only be modified by authenticated account owners. This includes updating of account name, email address and password. Besides that, signed-in users can view all registered users of the system and each domain belonging to the users (cf. Figure 4.4).

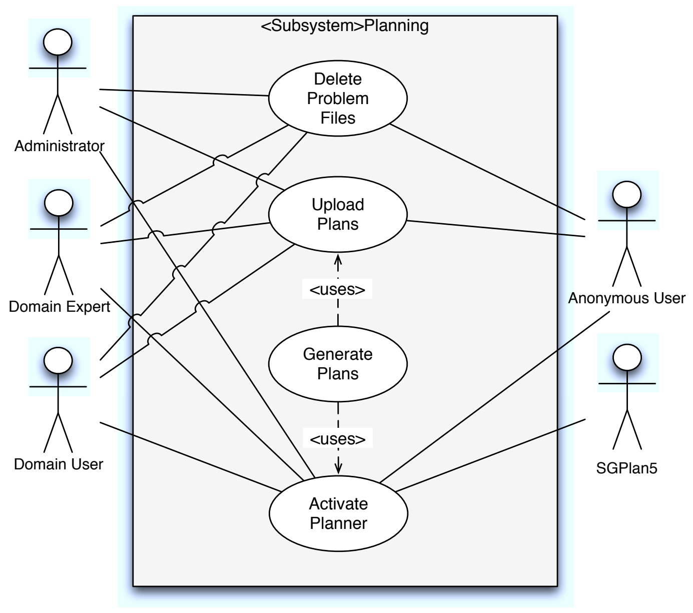


Figure 4.5: Use Case - Plan Generation

Plan generation involves uploading problem files and the activation of the planner. The prototype system allows every stakeholder to publish his planning problems and obtain plans through the system (cf. Figure 4.5). If a problem file already exists in a knowledge domain, the user must delete it in order to upload a new one.

Combined with the use cases described before, the basic process for plan generation can be briefly outlined. First, an anonymous user registers himself on the system registration page by creating an account with his user name, valid email address and confirmed password. After the registration, he creates a knowledge domain by giving a domain name, a domain description on a separate page dedicated to domain creation. Besides that, he must upload a domain and a problem file from his local file system to complete the creation process. The file upload is performed through a web form that should be placed right below the input fields for the domain name and description. When the domain is successfully created, the user goes to the domain page and activate the SGPlan5 planner by pressing an activation button on the page. A planning result is then displayed by the system that catches the output of the planner and puts it forward to a plan page.

CHAPTER 5

Implementation

5.1 Architecture

For realization of the proposed service system we first draw several architectural conditions that the system development is based on. Since the system is primarily used as a repository of knowledge domain resources and for the purpose of composing knowledge intensive services, we follow REST design philosophy and choose RoR v. 3.0.9 as the development framework that the service system is build on. The "convention over configuration" design paradigm of RoR can provide simplified assumptions including support for the unified HTTP interface. As stated in Chapter 2.7, RoR primarily supports exposing resources from database in RESTful style.

For implementation of data models of resource representations, MySQL is selected as an open-source option for persistent storage. MySQL databases can be connected to a RoR application by installing the gem `mysql2` in the project's Gemfile[1] and configured in "database.yml" of the project folder.

The intrinsic MVC principle of RoR divides the service system into Action View, Action Controller and Active Record that cooperate closely with each other. With the planner SGPlan5 integrated, Figure 5.1 depicts the architectural structure of the system.

1 A gem is basically a Ruby-based package. A Gemfile is a file includes dependencies of gems.

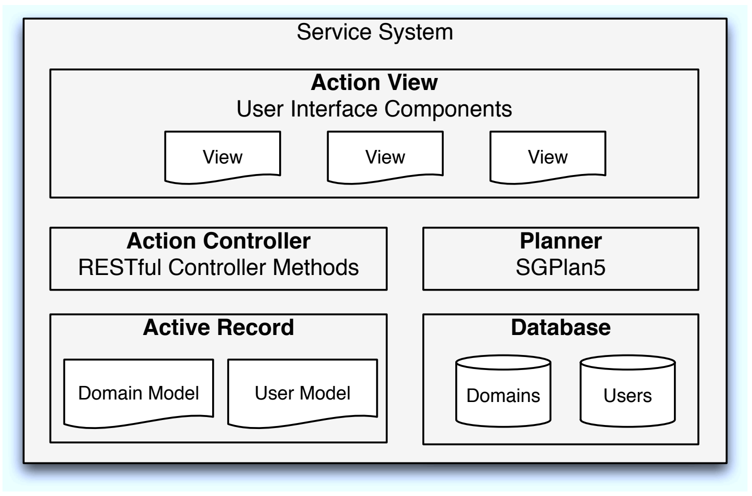


Figure 5.1: The Service System Architecture

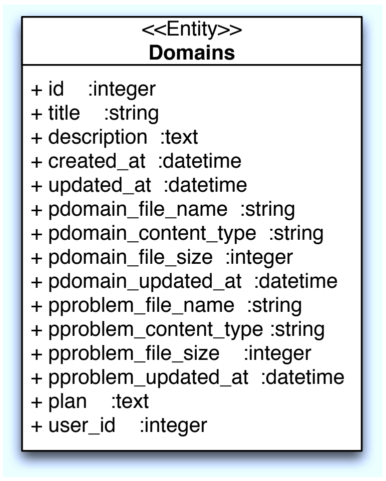
5.2 System Implementation

5.2.1 Domain Resource

The data set related to the knowledge domains in Chapter 4 is a type of read/write resource. Registered users can contribute and get its representations in the system. Each representation conveys several pieces of information associated with a knowledge domain:

- Title
- Description
- Planning domain
- Planning problem
- Generated plan

By using the RoR's scaffolding feature, the services of creating and exposing RESTful domain resource can be created automatically. The data model of a domain resource is illustrated in Figure 5.2. The `created_at` and `updated_at` are created automatically by Rails during the migration of the domain model. The prefix `pdomain_` denotes the attributes of the planning domain file and the prefix `pproblem_` denotes the attributes of the planning problem file. These attributes are used to give concrete informations about user's uploaded planning files. `user_id` is used to connect the user resource introduced later in the following section. In the project's



<<Entity>>
Domains
+ id :integer
+ title :string
+ description :text
+ created_at :datetime
+ updated_at :datetime
+ pdomain_file_name :string
+ pdomain_content_type :string
+ pdomain_file_size :integer
+ pdomain_updated_at :datetime
+ pproblem_file_name :string
+ pproblem_content_type :string
+ pproblem_file_size :integer
+ pproblem_updated_at :datetime
+ plan :text
+ user_id :integer

Figure 5.2: The Domain Data Model

Gemfile, a default root URL can be specified explicitly for listing all knowledge domains on the system's homepage:

```
root :to => 'domains#index'
```

A generated plan is the planning result generated by SGPlan5. We consider the planning result as a one-off resource since every user can uploads his own planning problem and get a plan for himself. According to ROA, a plan can therefore be defined as a separated resource with its own uniform HTTP actions in the controller. For the reason of constraining the scope of the system implementation, we use an overloaded PUT action in `DomainsController` for the implementation of this one-shot behavior of plan generation and assign it an URL for ensuring the addressability of the planning result. The naming convention of exposing a domain resource is of the following form:

```
http://localhost:3000/domains/<domain id>
```

`localhost:3000` is the default hostname of the prototype system on a local computer. The `<domain id>` indicates the knowledge domain's ID that is unique for each domain. The RESTful routes for a domain resource can be summarized in Table 5.1.

The service system uses the Ruby gem "paperclip"[2] to enable the file attachment feature that allows users to upload domain and problem files. In the form helper "_form.html.erb" of the domain view, `:html => { :multipart => true }` has to be specified for defining a MIME-multipart[3] form that enables file uploads. Two check boxes are inserted to the submission form

2 Paperclip: `https://github.com/thoughtbot/paperclip/` accessed in Jul. 2012.
3 MIME short for Multipurpose Internet Mail Extensions

HTTP Request	URL	Action
GET	/domains	index
GET	/domains/<domain id>	show
GET	/domains/new	new
GET	/domains/<domain id>/edit	edit
POST	/domains	create
PUT	/domains/<domain id>	update
DELETE	/domains/<domain id>	destroy

Table 5.1: RESTful Routes of The Domain Resource

for fetching the values that determine whether the user clicks the check boxes for deletion of the domain and problem files. For instance:

```
<p>
<% unless @domain.new_record? || !@domain.pdomain? %>
  <%= link_to "Uploaded domain", @domain.pdomain.url %> | click to delete
  <%= f.check_box :pdomain_delete, :label => 'Delete PDomain' %>
<% end %>
</p>
<p>
<% if @domain.new_record? || !@domain.pdomain? %>
  Planning Domain:
  <p>
    <%= f.file_field :pdomain %>
  </p>
<% end %>
</p>
```

Listing 5.1: Form Helper for The Upload Feature

The code snippet above defines that the URL of the domain file and the checkbox for deletion of the file should be presented to users when a domain has already been uploaded. Otherwise, an upload field comes out instead to allow users to upload a file from a local file system. The method `pdomain_delete` goes to the domain's model where validations of all the form fields take place. It is worth noting that additional statements `has_attached_file :pdomain` and `has_attached_file :pproblem` have to be declared in the domain model for announcing the attached attributes:

```
class Domain < ActiveRecord::Base
  attr_accessible :title, :description, :pdomain_delete, :pdomain, :
      pproblem_delete, :pproblem, :plan

  belongs_to :user

  validates :title, :presence => true, :length => { :maximum => 50 }
  validates :description, :presence => true, :length => { :maximum => 500 }
  validates :user_id, :presence => true

  default_scope :order => 'domains.created_at DESC'

  has_attached_file :pdomain
  has_attached_file :pproblem

  before_save :destroy_pdomain?
  before_save :destroy_pproblem?


  def pdomain_delete
    @pdomain_delete ||= "0"
  end

  def pdomain_delete=(value)
    @pdomain_delete = value
  end

  def pproblem_delete
    @pproblem_delete ||= "0"
  end

  def pproblem_delete=(value)
    @pproblem_delete = value
  end
private
  def destroy_pdomain?
    self.pdomain.clear if @pdomain_delete == "1"
  end
  def destroy_pproblem?
    self.pproblem.clear if @pproblem_delete == "1"
  end
end
```

Listing 5.2: The Domain Model

The form fields for the domain title and description should not be empty. Before a POST action makes effects in the backend database through the model, the methods `before_save :destroy_pdomain?` and `before_save :destroy_pproblem?` check up if the check boxes are marked by a user for file deletion. If the check boxes are marked, the existing attachment files will be deleted.

In the domain's controller "domains_controller.rb", an action named "plan" is defined for fetching requests of plan generation sent from the HTML form. Besides it, it is also responsible for activating the planner and adding a planning result as an attribute to the domain's model. In the route file "routes.rb", the rule `match '/domains/:id/plan', :to => 'domains#plan'` ensures a plan's URL with the corresponding domain's ID is directed correctly to the plan action

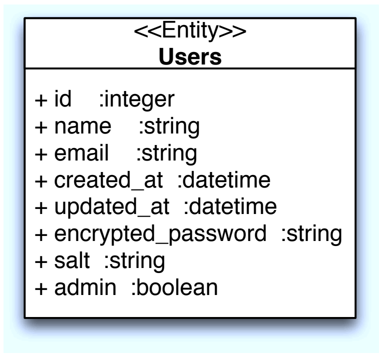


Figure 5.3: The User Data Model

of the domain's controller.

The SGPlan5 planner is integrated inside the definition of the plan action in the domain's controller with the command `%x[sgplan522 -o #{@dpath} -f #{@ppath} -out ~/output]` [4]. The argument `-out ~/output` indicates a resulted plan is stored in the "output" file of the home directory of the local operating system[5].

5.2.2 User Resource

A user resource mainly contains a user name and a user password. Since the system needs administrators to manage users, additional column in the model is used for differentiating user's privileges. The user's data model is illustrated in the Figure 5.3. The `admin` field is defined as a boolean value toggling the administrator status of a user on and off. We assign each user's password a unique string called `salt`. This makes passwords difficult to be cracked in the case of SQL injection attack.

The routes of a user resource is akin to the ones of a domain resource stated in Table 5.1. In the route file, an additional signup link is defined by `match '/signup', :to => 'users#new'` for allowing a user to create an account in the service system. The interface of account creation is defined in "new.html.erb" of the user's view. A user's password must be typed in twice for conforming the correctness. The validation of the two input passwords can be accomplished by using the hash argument `:confirmation => true` in the `validates` method of the user's model.

4 The planner has to be made systemwide available by adding its path into the environment variable of the local operating system.
5 The planning result can also be output directly to the standard output when omitting this argument.

5.2.3 Session Resource

In the prototype system everyone can upload a problem file and activate the planner for deriving a plan. Furthermore, every user that has an account in the system is allowed to create domains. However, it is prohibited to manipulate domains that do not belong to the correct users. To this end, an authentication mechanism is needed to protect the owner's knowledge domains without affecting the normal use of other system features. An authentication process identifies a user's ID and determines whether the ownership of a knowledge domain is correct. This further involves a signin and signout process for a user's session.

A user's signin page can be considered to be a "new" action through which a session is created. A new session for a user corresponds to the "create" action. Destroying (the "destroy" action) an existing session can be accomplished when a user signs out. These two actions correspond to Rails' standard controller actions and hence can be implemented as a RESTful resource as well (cf. Table 5.2).

HTTP Request	**URL**	**Action**
GET	/usersignin	new
POST	/usersessions	create
DELETE	/usersignout	destroy

Table 5.2: RESTful Routes of The Session Resource

Comparing to a domain or user resource, the difference of a session resource is that it is not a resource saved in database, but uses a cookie located on a user's computer. Hence there is no session model mapping incoming controller requests to database updates. In the route file, a session resource is defined to response the requests only with "new", "create" and "destroy". The user's signin and signout pages are matched to the "new" and "destroy" actions in the sessions's controller.

```
resources :sessions, :only => [:new, :create, :destroy]
match '/usersignin', :to => 'sessions#new'
match '/usersignout', :to => 'sessions#destroy'
```

The structure of the submission form for signup is defined as follows:

```
<%= form_for(:session, :url => sessions_path) do |f| %>
...
<% end %>
```

Once the form is posted, it yields a nested hash `{ :session => { :email => "<user_email>", :password => "<user_password>" } }` that can be used in the "create" action of the session's controller for examining the correctness of a user's email and password.

5.2.4 Connecting Resources

The session resource in the previous section gives users the ability to sign in and sign out with an existing account. After signin, a user is directed to the user's show view that is designed to be the profile of the user. For ensuring the connectedness of our system, each page rendered for a signed-in user should provide hyperlinks that can lead the user from the current state to the next state. For an unsigned-in user, the system should always give him the opportunity to sign in or sign up. To this end, a header helper "_header.html.erb" is placed in the system's layout "application.html.erb" with the command `<%= render 'layouts/header' %>`. It serves as a navigation menu that connects the session and user resources tightly together. In this way a user can easily follow a path through the entire system by sending requests of links and forms of resource representations. The header has the following contents:

```
<nav>
<ul>
  <li><%= link_to "Knowledge Domains", root_path %></li>
  <% if signed_in? %>
  <li><%= link_to "Users", users_path %></li>
  <li><%= link_to "Profile", current_user %></li>
  <li><%= link_to "Settings", edit_user_path(current_user) %></li>
  <li><%= link_to "Sign out", usersignout_path, :method => :delete %></li>
  <% else %>
  <li><%= link_to "Sign in", usersignin_path %></li>
  <li><%= link_to "Sign up", usersignup_path %></li>
  <% end %>
</ul>
</nav>
```

The methods `signed_in?` and `current_user` are defined in the session's helper. If `signed_in?` is true, the navigation menu will present the current signed-in user a list of registered users in the system through the hyperlink "Users". The "Profile" link contains the personal information about a user, i.e. user name, profile's URL, number of uploaded domains of the user and the domain titles that link to the actual domain pages. The "Settings" page allows the user modify user name and password. `current_user` returns the user object that is stored in the cookie. Hence `<%= link_to "Settings", edit_user_path(current_user)%>` links "Settings" to the URL of the signed-in user's edit page.

The connection between a user and his domain resources is accomplished using the associations `belongs_to` and `has_many`. The former one is specified in the domain's model "domain.rb" (also cf. the Listing 5.2) and the latter one is defined in the user's model "user.rb".

```
class Domain < ActiveRecord::Base
  attr_accessible :title, :description, :pdomain_delete, :pdomain,
                  :pproblem_delete, :pproblem, :plan

  belongs_to :user
  ...
```

```
end
```

```
class User < ActiveRecord::Base
  attr_accessor :password
  attr_accessible :name, :email, :password, :password_confirmation

  has_many :domains, :dependent => :destroy
  ...
end
```

The hash `:dependent => :destroy` denotes the dependency of the user and domain resources, i.e. when a user is deleted, the user's belonging domains are deleted as well. The declaration of the association yields automatically two new methods. We use the new method `@user.domains` to return an array of domains created by a user, where the instant variable `@user` is the incoming request on this user resource in "users_controller.rb". The array is used in the user's profile to list up the links of the user's knowledge domains.

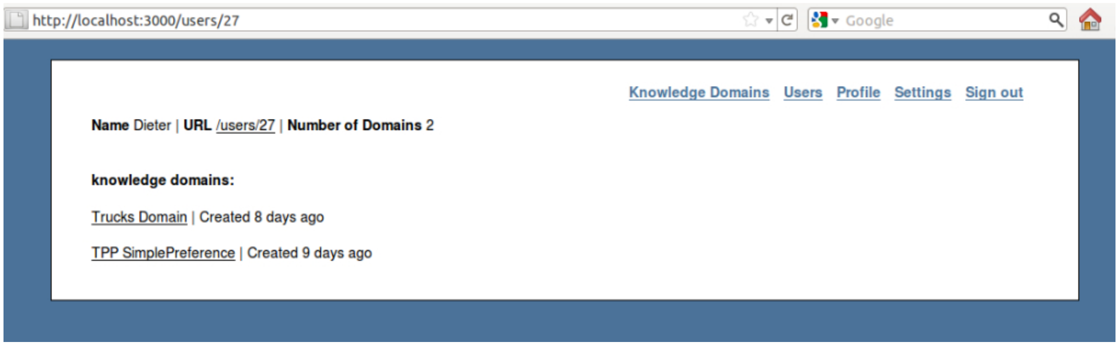
```
class UsersController < ApplicationController
  before_filter :authenticate, :only => [:index, :edit, :update]
  before_filter :admin_user, :only => :destroy
  ...
   def show
    @user = User.find(params[:id])
    @domains = @user.domains
    @title = @user.name
  end
  ...
end
```

Another new method yielded by the association can be found in "domains_controller.rb" where only a registered and signed-in user is allowed to create a new knowledge domain:

```
class DomainsController < ApplicationController
  ...
  def create
    @domain = current_user.domains.build(params[:domain])
    ...
  end
  ...
end
```

5.3 Results

The result of binding the user, session and domain resources together is demonstrated in Figure 5.4. A user's profile with the name "Dieter" is located with the URL `/users/27`. The domain list denotes that the user created two domains that are linked through their domain titles.



Name Dieter | **URL** /users/27 | **Number of Domains** 2

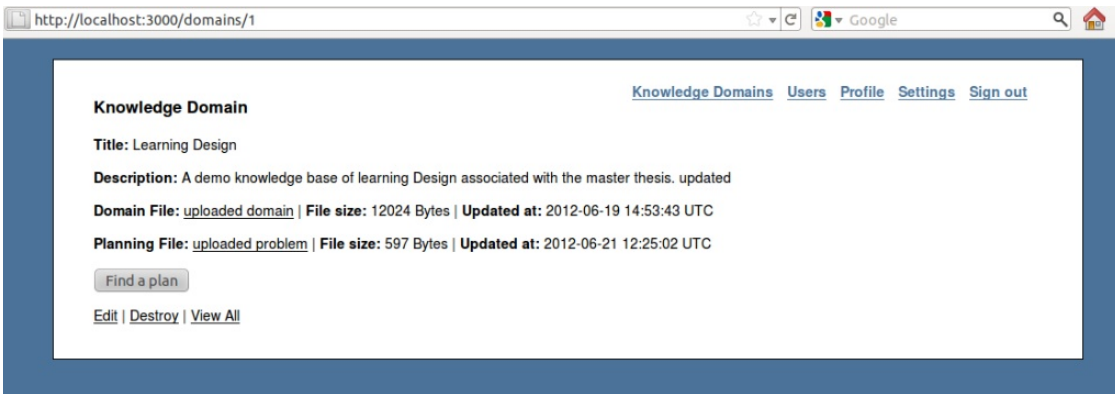
knowledge domains:

Trucks Domain | Created 8 days ago

TPP SimplePreference | Created 9 days ago

Figure 5.4: A User Profile

To publish the knowledge domain for LD introduced in Chapter 3.2, we first create a new account and then upload the domain and problem files to the prototype system. A screenshot of the domain view after completing an upload is found in Figure 5.5



Knowledge Domain

Title: Learning Design

Description: A demo knowledge base of learning Design associated with the master thesis. updated

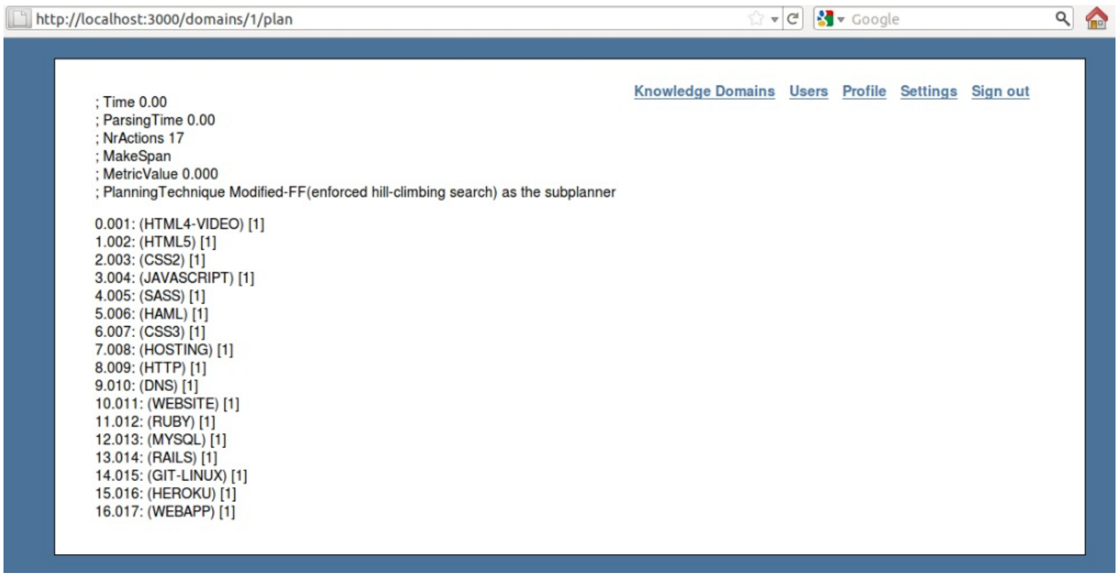
Domain File: uploaded domain | **File size:** 12024 Bytes | **Updated at:** 2012-06-19 14:53:43 UTC

Planning File: uploaded problem | **File size:** 597 Bytes | **Updated at:** 2012-06-21 12:25:02 UTC

Edit | Destroy | View All

Figure 5.5: The Show View of LD

A plan from the LD domain can be generated after a user presses the "Find a plan" button (cf. Figure 5.6). The content of a plan is shown under the plan URL with its domain ID included.



```
; Time 0.00
; ParsingTime 0.00
; NrActions 17
; MakeSpan
; MetricValue 0.000
; PlanningTechnique Modified-FF(enforced hill-climbing search) as the subplanner

0.001: (HTML4-VIDEO) [1]
1.002: (HTML5) [1]
2.003: (CSS2) [1]
3.004: (JAVASCRIPT) [1]
4.005: (SASS) [1]
5.006: (HAML) [1]
6.007: (CSS3) [1]
7.008: (HOSTING) [1]
8.009: (HTTP) [1]
9.010: (DNS) [1]
10.011: (WEBSITE) [1]
11.012: (RUBY) [1]
12.013: (MYSQL) [1]
13.014: (RAILS) [1]
14.015: (GIT-LINUX) [1]
15.016: (HEROKU) [1]
16.017: (WEBAPP) [1]
```

Figure 5.6: A Plan of LD

CHAPTER 6

Evaluation

In this chapter we recapitulate the problems we want to tackle in the thesis and the methodologies that guide the entire realization of the objectives. We focus on appraising the results that have been achieved to evaluate the work we put endeavor into and explain the obstacles we encountered during the research of the topic.

6.1 Functional Assessment

6.1.1 Knowledge Domain

A major objective of the thesis is to create a knowledge domain applicable to AI techniques. The purpose of it is to produce new knowledge based on the existing knowledge domain and the information given by domain users. The value of resulted new knowledge mainly depends on the expertise the knowledge domain conveys and inference ability the applied planning system has. Which domain is complex enough so that it is hard for human users to deduce manually and how to formalize the domain with an appropriate modeling language are the challenges we encountered at the beginning.

The example of the LD domain has proven to be a sophisticated area both on the logical and quantitative levels (cf. chapter 3.2). Even though the LOs we chose are small in amount (twenty one LOs in all), the relations among the subjects are fairly complex (cf. figure 3.4 and figure 3.5) and make human users difficult to find learning routes by hand. For modeling the LD domain, the de facto standard language PDDL3.0 is selected for the reason of wide support in the AI planning community. The language fulfills the need of personal customization that supposed to be provided by the prototype system. It is capable of describing the deterministic behaviors of our domain with relatively rich semantics comparing to its previous versions PDDL1.* and PDDL2.*.

As for the domain modeling, it refers to the job of acquisition and structuring of the information related to the LD issue and the implementation of the acquired knowledge in the form of a knowledge domain. We specifically chose a domain from web application development and performed a CGA for this area to elicit the knowledge a learner should possess in order to achieve certain competence levels of domain expertise. This process is done structurally and is also understandable with the aid of conceptual graphs. For assessing the knowledge skills precisely, the competence levels are quantized in percentage. The expertise of publishing websites and webapps is converted to PDDL with constructs such as numerical fluents and conditional effects. It is appropriate to model competence levels with numerical fluents since a student's skill varies during the time of learning. Conditional effects provides flexibility for domain experts to model learning outcomes based on each specific situation of students.

The domain is tested by three student's profiles as test cases attached in appendix (cf. Appendix B). SGPlan5 can generate for each of the profiles a correct learning route (cf. Listings 3.2, 3.4, 3.6) through which the students can increase their competence levels up to the desired ones described in the profiles. Each of the students is able to express learning goals both as hard and soft constraints. The correctness of the generated plans are measured in terms of fulfillment of the constraints. Recalling the fact that a plan is considered to be correct when hard constraints are satisfied. A full satisfaction of soft constraints should be engaged by a planner but does not have to be.

The hard constraints specified in the three test cases are fully satisfied by the planner. The soft constraints are defined by the students as personal preferences. The plans in Listing 3.2 and 3.6 fully reach the soft goals in problem files. In Listing 3.4, the preference of reaching the competence level of "structuring contents" in 40 hours is not fulfilled, since there is no possibility for the planner to select any existing LOs in order to satisfy this goal. However, the generated plan is still acceptable due to the suboptimal solution that includes the LO "HAML" for reaching the preference and reduces the learning time as much as possible.

6.1.2 Prototype

Another major objective in the thesis is to create a service system where domain knowledge created by users can be published and utilized by means of a web application. Each user registered in the system should be able to create and share his domain specific knowledge in the KB of the system. The public available knowledge domains can be used by every user that wants to grab new knowledge of interest through planning outcomes. Since the KB is accessible on the Web and the Web is make up of resources connected by hyperlinks (cf. Chapter 2.2.3), hence we consider each of the knowledge domains in the KB as a resource and the prototype system is considered as a container that connects and maintains the resources online.

Due to the similarity between the prototype and the nature of the Web, we adopt ROA for the guidance of the system implementation. We identified two major resources, i.e. knowledge domains and users, and align the methods for resource manipulations to the RESTful actions that are HTTP-conformed. To facilitate the system development, the web application framework RoR is chosen. It natively supports the REST (cf. Chapter 2.7.2) and its convention over configuration principle helped us focusing on building the system functionalities without putting efforts on various configurations.

The resource management in the system involves creation, update, deletion of the domain and user resources. The session resource management enables further association of the resources and authentication of the domain ownership (cf. Chapter 5.2.4) that is specified during the analysis of system requirements (cf. Chapter 4). Users are able to publish their own domains and find useful ones of others through the prototype platform. Profiles contain the information about users and their associated domain resources. SGPlan5 has been successfully integrated to the prototype for plan generation. The system requirements have been therefore fulfilled by the implemented system features.

6.2 Nonfunctional Assessment

Besides the functional assessment, we also evaluate the prototype on a nonfunctional level where addressability, statelessness, connectedness and the uniform interface are taken into the account. These properties mentioned in Chapter 2.2.3 determine the level of alignment of ROA for the prototype.

- Addressability: the prototype is addressable since it exposes the resources we identified, i.e. knowledge domains, user's profiles, domain files, problem files and generated plans, through individual URIs that are unambiguously accessible from a user perspective. Domain users are able to "copy & paste" the URIs to "transfer" the "representational state" to others that can access the resources across the web.
- Statelessness: in the prototype system we can statelessly view contents of domains, create, update problem files and generate plans. However, since the session resource is used to realize the authentication feature of the system, the whole system slightly violates the principle of statelessness [Richardson and Ruby, 2007b], despite the fact that sessions in the prototype is a RESTful resource by itself. Beside that, the authentication token is locally stored in the cookies on the client side, i.e. only the user's identity is stored in session, hence the core of the prototype is still RESTful and scalable.
- Connectedness: the header helper implemented in the Chapter 5.2.4 enables the system to guide a user's path by serving links and forms across the entire system states. In other words, users are able to put services of the system in different states just by following the hypertext representations or filling out the forms. Resource representations in the system are well linked to each other.

- The uniform interface: the prototype uses the basic operations of HTTP to deal with the resources with the only exception of the use of the overloaded POST action for plan generation. A possible solution for it is to consider generated plans as a subordinate resource to the domains and handle it separately with the standard HTTP verbs. Even so, we stick to use the "shortcut" to avoid excessive work and the prototype system remains simple. As for a strict adherence to the REST tenets, it depends on the concrete situations where the suitability of adopting REST to each specific web application needs to be considered. In our case, the prototype is only in charge of holding the domain and user resources, hence it is sufficient to attach generated plans as attributes to their domains.

6.3 Obstacles

As mentioned in Chapter 3.2.2, many planning systems only interpret a subset of the PDDL constructs correctly. For this reason, before constructing a domain in PDDL, the domain modeler should first study the manuals or the specifications of the planners. However, another problem could appear, since the related materials refer mostly to the functioning theories and barely introduce the modeling details and their supporting PDDL-elements. A work-around regarding to this problem is to use as many simple constructs as possible in order to avoid unpredictable misinterpretations. Otherwise the domain modeler has to attempt to test a PDDL-encoded domain against a planner. This can be a fairly time-consuming job. In this way, it is less guaranteed the domain can be interpreted correctly since many planners lack concrete and understandable error reporting capability.

The standard versions of PDDL are designed for describing deterministic and fully observable domains that is very restrictive in terms of applicability, since many of our realistic environments are nondeterministic and partially observable. In addition to the restrictive areas of applying PDDL, domain modeling tasks require a lot of knowledge engineering efforts. Different types of knowledge domains may require their own approaches for effectively acquiring appropriate knowledge or expertise.

CHAPTER 7

Summary and future work

In this thesis we addressed a concrete application case of service composition problems. A learning design domain is systematically analyzed in the phase of knowledge acquisition and formalized as a PDDL domain as planning input. This planning domain is written in PDDL3.0 with the constructs including numeric fluents and conditional effects. Domain users are able to specify their learning goals with constraints and preferences that definitely give more flexibility for customizations on the user side. We leveraged the domain independent planning system SG-PLan5 as the planner that enables the entire service composition process and produces tailored learning routes for each individual according to his specified personal desires.

The learning design use case depicts the importance of the selection of proper methods for different kinds of knowledge engineering tasks. The effective modeling of knowledge domains is the first step for eliciting the necessary expertise needed for a useful knowledge-based planning process. Another vital factor that influences the usefulness of a planning-based service composition is the selection of planning languages. Sufficient expressive power and inference efficiency are determinant for the language decision. PDDL3.0 is well supported by various planning systems. The ability of expressing preferences in goals are advantageous in some domain areas. However, this STRIPS-based language is still not expressive enough due to its restricted set of planning constructs. Despite this fact, we can still describe certain domains properly when they are simplified and abstracted correctly as a tradeoff.

To foster sharing of knowledge domains and their application for producing composite services, we developed a service system prototype where domains can be published and used in a knowledge base and plans can be generated according to user's planning problems. The implementation incorporates the design of the system in resource oriented architecture with the use of the modern web application development framework Ruby on Rails. The RESTful nature of Ruby on Rails facilitate the entire resource management of the system. The system together with the knowledge resources stored in it is scalable due to its conformed operations of standard HTTP actions.

In the implemented system, domain experts or knowledge engineers cares about identification, acquisition, modeling, publishing and maintenance of their domain knowledge. Normal users can find and read these domains. By specifying their planning needs in problem files, they can obtain plans through activating the integrated planner SGPlan5 after the planning files are uploaded to the system. This prototype system exemplifies an attempt of merging knowledge-based planning techniques and service composition problems. It encourages utilization of planning systems by providing online exploitation of knowledge resources for solving diverse planning goals. Besides the development results we discussed and summarized, it still remains features to be appended in the foreseeable future. We highlight them as follows:

- Since the system only adopts SGPLan5 as the planner for service composition, the knowledge domains are fairly constrained in terms of expressive power for representing realistic cases. The integration of other planning systems enables more knowledge domains to be shared and applied in the service system. Each of them can be written by a suitable planning language accepted by these planners. Even for the same domain, a domain user should be able to select different planners for benchmarking planning results.
- A search function for identifying user's interested domains can be added to the system. As knowledge domains accumulate in the system, there is no way to find a domain just by scrolling down or clicking paging indices of the website of the knowledge base. Semantic technology could be useful in a deep site-search, since recommendations for user's queries can be given according to an ontology relating query contents to coherent domains.
- Collaborative functions can promote the knowledge construction and sharing more effectively. Different users can work on building a knowledge domain as a group and an online version control function can keep the same domain save and consistent.
- Improvement of the user interface can be made to allow domains to be constructed in a graphical editor and a real-time validation feature can operate in background of a construction process for syntax check. This could accelerate the time of domain building. The construction process can be less error-prone.
- On the side of domain experts, social network features can help domain providers to know each other and exchange their expertise in either a privately hidden or publicly viewable way. This further fosters the growth of the experts community and the expansion of the entire knowledge base for a long term.

List of Figures

List of Tables

Acronyms

ADL	Action Description Language	20
AI	Artificial Intelligence	2
$B2B$	Business-to-Business	2
BDD	Binary Decision Diagram	30
BNF	Backus Normal Form	47
$BPEL4WS$	Business Process Execution Language for Web Services	2
$CaaS$	Composition as a Service	35
CAT	Composition Analysis Tool	35
CGA	Conceptual Graph Analysis	41
CNF	Conjunctive Normal Form	27
CoC	Convention over Configuration	30
$CRUD$	create, read, update, or delete	13
CWA	Closed World Assumption	21
DBN	Dynamic Bayesian Network	35
DL	Description Logic	36
DNS	Domain Name System	40
EAI	Enterprise Application Integration	2
FSM	Finite State Machine	29
HTN	Hierarchical Task Network Planning	28
ICT	Information and Communications Technology	1

Bibliography

[Bacchus and Petric, 2003] F. Bacchus and R. Petric. Reasoning with conditional plans in the presence of incomplete knowledge. In *Proceedings of the ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Information*, pages 96–102. Università di Trento, 2003.

[Baida et al., 2004] Ziv Baida, Jaap Gordijn, and Borys Omelayenko. A shared service terminology for online service provisioning, 2004.

[Baier and McIlraith, 2006] Jorge A. Baier and Sheila McIlraith. Planning with temporally extended goals using heuristic search. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS-06)*, 2006.

[Bean, 2010] James Bean. *SOA and Web Services Interface Design*. Elsevier Inc, 2010.

[Beek et al., 2006] Maurice H. Beek, Antonio Bucciarone, and Stefania Gnesi. A survey on service composition approaches: From industrial standards to formal methods, 2006.

[Bellman, 1957] R. Bellman. *Dynamic Programming*. Dover Publications, 1957.

[Berardi and Giacomo, 2000] Daniela Berardi and Giuseppe De Giacomo. Planning via model checking: Some experimental results, 2000.

[Bitran and Lojo, 1993] Gabriel R. Bitran and Maureen P. Lojo. A framework for analyzing service operations, March 1993.

[Blake et al., 2010] M. Brian Blake, Wei Tan, and Florian Rosenberg. Composition as a service, 2010.

[Blum and Furst, 1997] A. Blum and M. Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 90:281–300, 1997.

[Bucchiarone et al., 2007] Antonio Bucchiarone, Maurice ter Beek, and Stefania Gnesi. Web service composition approaches: From industrial standards to formal methods, 2007.

[Cardoso and Sheth, 2005] Jorge Cardoso and Amit Sheth. Introduction to semantic web services and web process composition. In *SWSWPC 2004*, pages 1–13. Springer, 2005.

[Cardoso et al., 2009] Jorge Cardoso, Konrad Voigt, and Matthias Winkler. Service engineering for the internet of services, 2009.

[Castillo et al., 2010] L. Castillo, L. Morales, A. Gonzalez-Ferrer, J. Fdez-Olivares, D. Borrajo, and E. Onaindia. Automatic generation of temporal planning domains for e-learning. *Journal of Scheduling*, 13(4):347–362, 2010.

[Chen et al., 2003] L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston, and P.R Smart. Towards a knowledge-based approach to semantic service composition, 2003.

[Colucci et al., 2005] S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, G.Piscitelli, and S. Coppi. Knowledge based approach to semantic composition of teams in an organization, 2005.

[Doshi et al., 2005] Prashant Doshi, Richard Goodwin, Rama Akkiraju, and Kunal Verma. Dynamic workflow composition using markov decision processes. *International Journal of Web Services Research*, 2005.

[Edelkamp and Helmert, 2001] Stefan Edelkamp and Malte Helmert. Mips the model-checking integrated planning system. *AI Magazine*, 22(3), 2001.

[Endrei et al., 2004] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Patterns: Service-Oriented Architecture and Web Services*. International Business Machines Corporation, April 2004.

[Erl, 2005] Thomas Erl. *Service-Oriented Architecture Concepts Technology and Design*. Prentice Hall PTR, August 2005.

[Erl, 2007] Thomas Erl. *SOA Principles of Service Design*. Prentice Hall, July 2007.

[Fensel et al., 2011] Prof. Dieter Fensel, Dr. Federico Michele, Facca Ioan Toma, and Dr. Elena Simperl. Semantic web services, 2011.

[Fernandez, 2010] Obie Fernandez. *THE RAILS 3 WAY*. Addison-Wesley, 2010.

[Fielding, 2000] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[Fox and Long, 2003] Maria Fox and Derek Long. Pddl2.1 : An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, December 2003.

[Garrido et al., 2011] Antonio Garrido, Lluvia Morales, and Ivan Serina. Applying case-based planning to personalized e-learning, 2011.

[Gerevini and Long, 2005] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. Technical report, Department of Electronics for Automation, University of Brescia, 2005.

[Gerevini et al., 2008] Alfonso E. Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173:619–668, 2008.

[Gruber, 1993] Tom Gruber. A translation approach to portable ontologies. In *Knowledge Acquisition*, volume 5, pages :199–220, 1993.

[Hidaka, 2006] Kazuyoshi Hidaka. Trends in services sciences in japan and abroad, 2006.

[Hoffmann and Brafman, 2004] J. Hoffmann and R. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2004.

[Hoffmann, 2011] Jörg Hoffmann. Ff: The fast-forward planning system, 2011.

[Hoffmanng, 2002] Jörg Hoffmanng. Extending ff to numerical state variables, 2002.

[Honey and Mumford, 1986] Peter Honey and Alan Mumford. The manual of learning styles. Peter Honey Peter Honey Associates, 1986.

[Hsu et al., 2006] Chih-Wei Hsu, Benjamin W. Wah, Ruoyun Huang, and Yixin Chen. Handling soft constraints and goals preferences in sgplan, 2006.

[Jabbar et al., 2006] S. Jabbar, M. Nazih, and S. Edelkamp. Large-scale optimal pddl3 planning with mips-xxl. In *Proceedings of the ICAPS Booklet on the Fifth International Planning Competition*, 2006.

[Jonassen et al., 1999a] David H. Jonassen, Martin Tessmer, and Wallace H. Hannum. *Task Analysis Methods for Instructional Design*, chapter 20, page 199. LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS 1999 Mahwah, New Jersey London, 1999a.

[Jonassen et al., 1999b] David H. Jonassen, Martin Tessmer, and Wallace H. Hannum. Task analysis methods for instructional design. LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS 1999 Mahwah, New Jersey London, 1999b.

[Kavantzas et al., 2004] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, and Yves Lafon. *Web Services Choreography Description Language Version 1.0*. World Wide Web Consortium, w3c working draft 17 december 2004 edition, 2004.

[Kenessey, 2004] Zoltan Kenessey. The primary, secondary, tertiary and quaternary sectors of the economy. U.S. Federal Reserve Board, 2004.

[Kim et al., 2004] Jihie Kim, Yolanda Gil, and Marc Spraragen. Demonstration: A knowledge-based approach to interactive workflow composition, 2004.

[Lee et al., 2001] Tim Berners Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American Magazine*, May 2001.

[Liu and Özsu, 2009] Ling Liu and M. Tamer Özsu. Encyclopedia of database systems, 2009.

[Marinagi et al., 2005] Catherine C. Marinagi, Themis Panayiotopoulos, and Constantine D. Spyropoulos. Ai planning and intelligent agents, 2005.

[Martínez and Lespérance, 2004] Erick Martínez and Yves Lespérance. Web service composition as a planning task: Experiments using knowledge-based planning, 2004.

[McDermott, 2000] Drew McDermott. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2), 2000.

[McDermott et al., 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL – The Planning Domain Definition Language – Version 1.2*. The AIPS-98 Planning Competition Committee, October 1998.

[Medjahed and Bouguettaya, 2011] Brahim Medjahed and Athman Bouguettaya. *Service Composition for the Semantic Web*. Springer, 2011.

[Morales et al., 2008] Lluvia Morales, Luis Castillo, Juan Fernandez-Olivares, and Arturo Gonzalez-Ferrer. Automatic Generation of User Adapted Learning Designs: An AI-Planning Proposal. In *AH '08 Proceedings of the 5th international conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 324 – 328. Springer-Verlag Berlin, Heidelberg, 2008.

[Nau et al., 2003] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *Artificial Intelligence Research*, pages 379–404, 2003.

[Nau et al., 2004a] Dana Nau, Malik Ghallab, and Paolo Traverso. *Automated Planning Theory and Practice*. Morgan Kaufmann Publishers, May 2004a.

[Nau et al., 2004b] Dana Nau, Evren Sirin, Bijan Parsia, Dan Wu, and James Hendler. HTN Planning for Web Service Composition Using SHOP2, June 2004b.

[Nau, 2007] Dana S. Nau. Current Trends in Automated Planning. *AI Magazine - AAAI*, 28(4), 2007.

[Pashenkova et al., 1996] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem, April 1996.

[Pednault, 1994] Edwin P. D. Pednault. Adl and the state-transition model of action. *Journal of Logic and Computation*, 4:467, 1994.

[Peer, 2005a] Joachim Peer. A pop-based replanning agent for automatic web service composition. In *ESWC'05 Proceedings of the Second European conference on The Semantic Web: research and Applications*. Springer-Verlag Berlin, Heidelberg, 2005a.

[Peer, 2005b] Joachim Peer. Web Service Composition as AI Planning – a Survey, March 2005b.

[Pistore et al., 2003] Marco Pistore, Piergiorgio Bertoli, Alessandro Cimatti, and Ugo Dal Lago. Extending PDDL to nondeterminism, limited sensing and iterative conditional plans. In *ICAPS Workshop on PDDL, Informal Proceedings*, 2003.

[Rajshekhar, 2008] A.P. Rajshekhar. *Building Dynamic Web 2.0 Websites with Ruby on Rails*. Packt Publishing Ltd., 2008.

[Rao and Su, 2004] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, 2004.

[Reiff-Marganiec et al., 2009] Stephan Reiff-Marganiec, Kun Chen, and Jiuyun Xu. Markov-HTN Planning Approach to Enhance Flexibility of Automatic Web Services Composition. In *IEEE International Conference on Web Services*, pages 9–16, 2009.

[Richardson and Ruby, 2007a] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'REILLY, 2007a.

[Richardson and Ruby, 2007b] Leonard Richardson and Sam Ruby. *RESTful Web Services*, chapter 4, page 89. O'REILLY, 2007b.

[Roxburgh and Manyika, 2011] Charles Roxburgh and James Manyika. The great transformer: The impact of internet on economic growth, October 2011.

[Russell and Norvig, 2010] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, 2010.

[Sampson and Froehle, 2006] Scott E. Sampson and Craig M. Froehle. Foundations and implications of a proposed unified services theory, 2006.

[Sanner, 2011] Scott Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description, 2011.

[Spohrer et al., 2007] Jim Spohrer, Paul P. Maglio, John Bailey, and Daniel Gruhl. Steps toward a science of service systems, 2007.

[Spohrer et al., 2008] Jim Spohrer, Stephen L. Vargo, Nathan Caswell, and Paul P. Maglio. The service system is the basic abstraction of service science. In *Proceedings of the 41st Hawaii International Conference on System Sciences*, 2008.

[Srivastava and Koehler, 2003] Biplav Srivastava and Jana Koehler. Web service composition - current solutions and open problems, 2003.

[Stal, 2006] Michael Stal. Using architectural patterns and blueprints for service-oriented architecture, March/April 2006.

[Teboul, 2006] James Teboul. SERVICE IS FRONT STAGE We are all in services ... more or less!, 2006.

[W3C, 2004] W3C. Web services glossary, February 2004. URL `http://www.w3.org/TR/ws-gloss/`.

[Younes and Littman, 2004] Hakan L. S. Younes and Michael L. Littman. PPDDL1.0: An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects, 2004.

APPENDIX A

The Knowledge Domain

```
(define (domain Online-Learning)
(:requirements :typing :adl :fluents)
(:types profile level system course software language - object
    frontend-basic frontend-extended backend-basic backend-extended - course

)
(:constants HTML4 HTML5 HAML CSS2 CSS3 SASS DNS HOSTING HTTP RUBY NANOC MYSQL
      POSTGRESQL RAILS HEROKU JS JQUERY GIT SINATRA WEBSITE WEBAPP - course
      HTML4 CSS2 - frontend-basic
      HTML5 CSS3 HAML SASS DNS HTTP JS JQUERY HOSTING - frontend-extended
      RUBY RAILS MYSQL - backend-basic
      NANOC HEROKU GIT SINATRA POSTGRESQL - backend-extended
      LOW MEDIUM HIGH - level
      WIN MAC LINUX - system
)
(:predicates
  (theoretical ?p - profile)
  (pragmatic ?p - profile)
  (passed ?c - course)
  (technical-english ?l - level)
  (technical-german ?l - level)
  (operating-system ?s - system)
)

(:functions
  (publish-website)
  (structure-content)
  (format-content)
  (publish-webapp)
  (frontend-programming)
  (backend-programming)
  (database)
  (framework)
  (deployment)
  (version-control)
)

(:action html4
  :parameters ()
  :precondition (and (or (technical-english LOW) (technical-english MEDIUM)
             (technical-english HIGH)) (= (publish-website) 0) (not (passed
                HTML4)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (structure-content) 40) (
              increase (total-time) 4))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (structure-content) 50) (
              increase (total-time) 5))))
        (passed HTML4)
      )
)

(:action html4-video
  :parameters ()
  :precondition (and (technical-english HIGH) (= (publish-website) 0) (not (
      passed HTML4)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (structure-content) 40) (
              increase (total-time) 0.8))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (structure-content) 50) (
              increase (total-time) 1))))
        (passed HTML4)
      )
)

(:action html4-de
  :parameters ()
  :precondition (and (technical-german HIGH) (= (publish-website) 0) (not (
      passed HTML4)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (structure-content) 40) (
              increase (total-time) 4))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (structure-content) 50) (
              increase (total-time) 5))))
        (passed HTML4)
      )
)

(:action html5
  :parameters ()
  :precondition (and (>= (structure-content) 40) (not (passed HTML5)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (structure-content) 30) (
              increase (total-time) 3))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (structure-content) 40) (
              increase (total-time) 4))))
        (passed HTML5)
      )
)

(:action haml
  :parameters ()
  :precondition (and (>= (structure-content) 40) (> (format-content) 0) (not
      (passed HAML)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (structure-content) 5) (
              increase (total-time) 2))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (structure-content) 10) (
              increase (total-time) 3))))
        (passed HAML)
      )
)

(:action css2
  :parameters ()
  :precondition (and (>= (structure-content) 40) (not (passed CSS2)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (format-content) 40) (
              increase (total-time) 4))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (format-content) 50) (increase
              (total-time) 5))))
        (passed CSS2)
      )
)

(:action css3
  :parameters ()
  :precondition (and (>= (structure-content) 70) (>= (format-content) 40) (
      not (passed CSS3)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (format-content) 20) (
              increase (total-time) 3))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (format-content) 30) (increase
              (total-time) 4))))
        (passed CSS3)
      )
)

(:action sass
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (format-content) 40) (
      not (passed SASS)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (format-content) 10) (
              increase (total-time) 2))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (format-content) 20) (increase
              (total-time) 3))))
        (passed SASS)
      )
)

(:action hosting
  :parameters ()
  :precondition (and (>= (structure-content) 40) (not (passed HOSTING)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (deployment) 20) (increase (
              total-time) 1))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (deployment) 60) (increase (
              total-time) 2))))
        (passed HOSTING)
      )
)

(:action dns
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (deployment) 20) (not (
      passed DNS)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (deployment) 50) (increase (
              total-time) 2))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (deployment) 20) (increase (
              total-time) 1.5))))
        (passed DNS)
      )
)

(:action http
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (deployment) 20) (not (
      passed HTTP)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (deployment) 20) (increase (
              total-time) 1.5))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (deployment) 10) (increase (
              total-time) 2))))
        (passed HTTP))
)

(:action website
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (deployment) 40) (not (
      passed WEBSITE)))
  :effect (and
        (forall (?c - frontend-basic)
          (when (passed ?c) (increase (publish-website) 25))
        )
        (forall (?c - frontend-extended)
          (when (passed ?c) (increase (publish-website) 5))
        )
        (increase (publish-website) 5)
        (increase (total-time) 3)
        (passed WEBSITE)
      )
)

(:action javascript
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (format-content) 40) (
      not (passed JS)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (frontend-programming) 40) (
              increase (total-time) 15))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (frontend-programming) 60) (
              increase (total-time) 10))))
        (passed JS))
)

(:action jquery
  :parameters ()
  :precondition (and (>= (structure-content) 40) (>= (format-content) 40)
             (>= (frontend-programming) 40) (not (passed JQUERY)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (frontend-programming) 10) (
              increase (total-time) 15))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (frontend-programming) 40) (
              increase (total-time) 10))))
        (passed JQUERY))
)

(:action ruby
  :parameters ()
  :precondition (and (> (publish-website) 0) (not (passed RUBY)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (backend-programming) 40) (
              increase (total-time) 25))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (backend-programming) 50) (
              increase (total-time) 20))))
        (passed RUBY))
)

(:action nanoc
  :parameters ()
  :precondition (and (>= (backend-programming) 40) (not (passed NANOC)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (backend-programming) 3) (
              increase (total-time) 1))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (backend-programming) 5) (
              increase (total-time) 2))))
        (passed NANOC))
)

(:action sinatra
  :parameters ()
  :precondition (and (>= (backend-programming) 40) (not (passed SINATRA)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p)
            (and (increase (framework) 30) (increase (backend-programming) 5)
                 (increase (database) 5) (increase (total-time) 30))))
        (forall (?p - profile)
          (when (pragmatic ?p)
            (and (increase (framework) 40) (increase (backend-programming)
                10)
               (increase (database) 10) (increase (total-time) 20))))
        (passed SINATRA)
      )
)

(:action rails
  :parameters ()
  :precondition (and (>= (backend-programming) 40) (not (passed RAILS)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p)
            (and (increase (framework) 50) (increase (backend-programming)
                10)
               (increase (database) 5) (increase (total-time) 35))))
        (forall (?p - profile)
          (when (pragmatic ?p)
            (and (increase (framework) 60) (increase (backend-programming)
                20)
               (increase (database) 10) (increase (total-time) 40))))
        (passed RAILS)
      )
)

(:action mysql
  :parameters ()
  :precondition (and (= (publish-webapp) 0) (> (backend-programming) 0) (not
      (passed MYSQL)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (database) 40) (increase (
              total-time) 8))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (database) 50) (increase (total
              -time) 15))))
        (passed MYSQL)
      )
)

(:action postgresql
  :parameters ()
  :precondition (and (= (publish-webapp) 0) (> (backend-programming) 0) (not
      (passed POSTGRESQL)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (database) 40) (increase (
              total-time) 10))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (database) 50) (increase (total
              -time) 13))))
        (passed POSTGRESQL)
      )
)

(:action git-win
  :parameters ()
  :precondition (and (operating-system WIN) (> (backend-programming) 0) (not
      (passed GIT)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (version-control) 80) (
              increase (total-time) 1))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (version-control) 100) (
              increase (total-time) 2))))
        (passed GIT)
      )
)

(:action git-linux
  :parameters ()
  :precondition (and (operating-system LINUX) (> (backend-programming) 0) (
      not (passed GIT)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (version-control) 80) (
              increase (total-time) 1))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (version-control) 100) (
              increase (total-time) 2))))
        (passed GIT)
      )
)

(:action git-mac
  :parameters ()
  :precondition (and (operating-system MAC) (> (backend-programming) 0) (not
      (passed GIT)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (version-control) 80) (
              increase (total-time) 1))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (version-control) 100) (
              increase (total-time) 2))))
        (passed GIT)
      )
)

(:action heroku
  :parameters ()
  :precondition (and (>= (version-control) 80) (not (passed HEROKU)))
  :effect (and
        (forall (?p - profile)
          (when (theoretical ?p) (and (increase (deployment) 10) (increase (
              total-time) 0.5))))
        (forall (?p - profile)
          (when (pragmatic ?p) (and (increase (deployment) 10) (increase (
              total-time) 1))))
        (passed HEROKU)
      )
)

(:action webapp
  :parameters ()
  :precondition (and (>= (publish-website) 50) (>= (version-control) 80) (>=
      (database) 40)
             (>= (deployment) 10) (>= (backend-programming) 40) (not (passed
                 WEBAPP)))
  :effect (and
        (forall (?c - frontend-basic)
          (when (passed ?c) (increase (publish-webapp) 1))
        )
        (forall (?c - frontend-extended)
          (when (passed ?c) (increase (publish-webapp) 1))
        )
        (forall (?c - backend-basic)
          (when (passed ?c) (increase (publish-webapp) 20))
        )
        (forall (?c - backend-extended)
          (when (passed ?c) (increase (publish-webapp) 5))
        )
        (increase (publish-webapp) 4)
        (increase (backend-programming) 20) (increase (database) 20)
        (increase (version-control) 20) (increase (total-time) 30)
        (increase (total-time) 15)
        (passed WEBAPP)
      )
)
```

```
;; NOTE: when condition can not contain fluent and predicate in the same
    clause.
;; use "forall" to define special scale of skills from courses.

)
```

APPENDIX B

LD Test Suites

B.1 Test Profile 1

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
  Peter - profile
)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 0)
  (= (format-content) 0)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 0)
  (theoretical Peter)
  (technical-english LOW)
  (technical-german HIGH)
  (operating-system MAC)
)
(:goal
  (and (>= (structure-content) 60)
       (>= (format-content) 60)
       (preference p1 (passed SASS)))
)
(:metric minimize (*100 (is-violated p1)))
)
```

B.2 Test Profile 2

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
  David - profile
)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 50)
  (passed HTML4)
  (= (format-content) 50)
  (passed CSS2)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 10)
  (pragmatic David)
  (technical-english HIGH)
  (technical-german LOW)
  (operating-system WIN)
)
(:goal
  (and (>= (version-control) 80)
     (>= (framework) 40)
  (preference p1 (>= (structure-content) 60))
  (preference p2 (<= (total-time) 60))
  )
)
(:metric minimize (+ (*1 (is-violated p1)) (*1 (is-violated p2))))
)
```

B.3 Test Profile 3

```
(define (problem Learning-Path)
(:domain Online-Learning)
(:objects
  Caleb - profile
)
(:init
  (= (publish-website) 0)
  (= (publish-webapp) 0)
  (= (structure-content) 0)
  (= (format-content) 0)
  (= (frontend-programming) 0)
  (= (backend-programming) 0)
  (= (database) 0)
  (= (framework) 0)
  (= (deployment) 0)
  (= (version-control) 0)
  (= (total-time) 0)
  (theoretical Caleb)
  (technical-english HIGH)
  (technical-german LOW)
  (operating-system LINUX)
)
(:goal
  (and (>= (publish-website) 90) (>= (publish-webapp) 60) (preference p1 (>=
      (deployment) 100)))
)

(:metric minimize (is-violated p1))
)
```

APPENDIX C
Prototype Installation

This installation guide is dedicated to getting the prototype system up and running on a Mac OX system.

To install Ruby, Rails, Git and RVM in one step, the easiest way is to download and install RailsInstaller for OS X:

```
http://railsinstaller.org/#osx
```

After the installation, check out if Git, Ruby and Rails are installed correctly with the following commands on terminal:

```
$ git version
$ ruby -v
$ rails -v
```

If installed correctly, install MySQL in the 64bit version according to the Mac version on your local machine:

```
http://dev.mysql.com/downloads/mysql/5.1.html#macosx-dmg
```

In the link above, choose the MySQL package installer package in `.dmg` and following the installation guide in the following link:

```
http://dev.mysql.com/doc/refman/5.1/en/macosx-installation-prefpane.html
```

Then install the MySQL/Ruby connector with the following command (administrator privilege needed):

```
$ sudo gem install mysql
```

Turn on the MySQL server on preference panel and type `mysql` on a terminal to go to the mysql terminal. Type in the following commands to create a database for the prototype system and set it up to the current database:

```
mysql> CREATE DATABASE wsc;
mysql> USE wsc
```

To download the source code of the prototype system, first choose a folder on your Mac to place the files. Navigate to the folder and clone the source code from GitHub using the following command:

```
$ git clone git@github.com:jinzeli/wsc_using_ai.git
```

Alternatively, you can go to the project website directly and press the "Downloads" button to get the source code in zip or tar.gz formats:

```
https://github.com/jinzeli/wsc_using_ai
```

After the download, navigate to the project folder `wsc_using_ai` on terminal and type in:

```
$ bundle install
```

This will install all needed gems (packages) for the prototype. After that, finally run the prototype on terminal with:

```
$ rails server
```

The prototype application is under the following URL accessible:

```
http://localhost:3000
```

NOTE: To run the prototype on a Apache server, follow the guide below, skip the steps for installing MacPorts, Ruby, Rails, MySQL; go to the section Apache directly:

```
http://geryit.com/blog/installing-mysql-with-rails-on-mac-os-x-snow-leopard/
```