End-user service composition from a social networks analysis perspective

Abderrahmane Maaradji

Ecole Doctorale EDITE

Thèse présentée pour l'obtention du diplôme de Docteur de Télécom & Management SudParis

Doctorat conjoint Télécom & Management SudParis et Université Pierre et Marie Curie

Par
Abderrahmane MAARADJI

Titre

End-user Service Composition From a Social Networks Analysis Perspective

Soutenue le 2 décembre 2011 devant le jury composé de :

Rapporteur Professeur Jean-Marc PETIT
Rapporteur Professeur Zakaria MAAMAR
Examinateur Professeur Athena VAKALI
Examinateur Professeur Marcelo DIAS DE AMORIM
Examinateur Professeur Salima BENBERNOU
Co-encadrant Monsieur Johann DAIGREMONT
Co-encadrant Docteur Hakim HACID
Directeur de thèse Professeur Noël CRESPI

Thèse n° 2011TELE0028

Abstract

Service composition has risen from the need to make information systems more flexible and open. The Service Oriented Architecture has become the reference architecture model for applications carried by the impetus of Internet (Web). In fact, information systems are able to expose interfaces through the Web which has increased the number of available Web services. Moreover, with the emergence of Web 2.0, service composition has evolved toward Web users with limited technical skills. Those end-users, named Y generation, are participating, creating, sharing and commenting content through the Web. This evolution in service composition is translated by the reference paradigm of Mashup and Mashup editors such as Yahoo Pipes! This paradigm has established the service composition within end users community enabling them to meet their own needs, for instance by creating applications that do not exist. Additionally, Web 2.0 has also brought its social dimension, allowing users to interact, either directly through the online social networks or indirectly by sharing, modifying content, or adding metadata.

In this context, this thesis aims to support the evolving concept of service composition through meaningful contributions. The main contribution of this thesis is indeed the introduction of the social dimension within the process of building a composite service through end users' dedicated environments. In fact, this concept of social dimension considers the activity of composing services (creating a Mashup) as a social activity. This activity reveals social links between users based on their similarity in selecting and combining services. These links could be an interesting dissemination means of expertise, accumulated by users when composing services. In other terms, based on frequent composition patterns, and similarity between users, when a user is editing a Mashup, dynamic recommendations are proposed. These recommendations aim to complete the initial part of Mashup already introduced by the user. This concept has been explored through (i) a step-by-step Mashup completion by recommending a single service at each step, and (ii) a full Mashup completion approaches by recommending the whole sequence of services that could complete the Mashup.

Beyond the integration of the social dimension within the service composition process, this thesis has addressed a particular constraint for this recommendation system which conditions the interactive systems requirements in terms of response time. In this regard, we developed robust algorithms adapted to the specificities of our problem. Whereas a composite service is considered as a sequence of basic service, finding similarities between users comes first to find frequent patterns (subsequences) and then represent them in an advantageous data structure for the recommendation algorithm. The proposed algorithm FESMA, meets exactly those requirements based on the FSTREE structure with interesting results compared to the prior art.

Finally, to implement the proposed algorithms and methods, we developed a

Mashup creation framework, called Social Composer (SoCo). This framework, dedicated to end users, firstly implements abstraction and usability requirements through a workflow-based graphic environment. As well, it implements all the mechanisms needed to deploy composed service starting from an abstract description entered by the user. More importantly, SoCo has been augmented by including the dynamic recommendation functionality, demonstrating by the way the feasibility of this concept.

Keywords: Service composition, Mashup, dynamic service recommendation, frequent sequence mining algorithm

List of original articles

This thesis is based on the following original articles:

1. A. Maaradji, H. Hacid, R. Skraba, A. Vakali and N. Crespi; “Service Composition Full Completion Based on Novel Frequent Sequence Mining”, Bell Labs Technical Journal, (under reviews, to appear in 2012).

2. A. Maaradji, H. Hacid, R. Skraba, and A. Vakali; “Social Web Mashups Full Completion via Frequent Sequence Mining”; The 7th IEEE 2011 World Congress on Services (SERVICES 2011), Washington DC,USA, July 2011.

3. A. Maaradji, H. Hacid, R. Skraba, A. Lateef, J. Daigremont, and N. Crespi; “Social-based Web Services Discovery and Composition for Step-by-step Mashup Completion”, The 9th IEEE International Conference on Web Services (ICWS 2011), Washington DC,USA, July 2011.

4. A. Maaradji, H. Hacid, R. Skraba, A. Lateef, J. Daigremont, N. Crespi; “Social Discovery and Composition of Web Services”; in the EUD4Services Workshop-Empowering End-Users to Develop Service-based Applications; Torre Canne, Italy, June 2011.

5. A. Maaradji, H. Hakim, J. Daigremont, N. Crespi, “Towards a Social Network Based Approach for Services Composition”, in IEEE International Conference on Communications (ICC 2010), Cape Town, South Africa, May 2010.

6. A. Maaradji, H. Hacid, J. Daigremont, N. Crespi, “Social Composer: A Social-Aware Mashup Creation Environment”, ACM Conference on Computer Supported Cooperative Work CSCW 2010 (demo session), Savannah, Georgia, USA, February 2010.

7. A. Maaradji, H. Hacid, J. Daigremont and N. Crespi, “Composition de services web basée sur les réseaux sociaux”, in Conférence Internationale Francophone sur l’Extraction et la Gestion des Connaissances, Tunisia, January 2010.

8. R. Skraba, M. Beauvais, J. Stan, A. Maaradji, J. Daigremont, “Developing Compelling Social-Enabled Applications with Context-Based Social Interaction Analysis”, in International Conference on Advances in Social Network Analysis and Mining, July 2009.

9. A. Maaradji, C. Huang and N. Crespi, “Towards Personalized Services Composition on IMS: A Basic Approach”, International Conference on Advanced Infocomm Technology, Xi’an, China, July 2009.

10. A. Maaradji, A. Gonguet, N. Crespi and J. Daigremont, "La composition de services dans l’IMS, un premier pas vers l’ubiquitous computing", présenté au

22ème Congrès DNAC : La convergence des réseaux IP et le Post-IP, Paris, France, Décembre 2008.

Contents

Chapter 1

Introduction

1.1 Business context and motivation of the thesis

The concept of service composition has evolved in many forms and names over decades. Its most recent remarkable achievement is the "Service Oriented Architecture" (SOA) model which considers the service as a building block for any application design. This model translates the concept by aiming at creating modular, scalable, composable, and interoperable applications. Traditionally, this need raised in a Business-to-Business context, and was commonly known as "Enterprise Application Integration" (EAI).

With the rise and overwhelming success of the World Wide Web (WWW), the concepts of SOA and service composition have been consolidated and drew more interest. Indeed, in the recent few years, there has been an explosion of the number of applications published on the Web. In fact, Web business actors are competing to be first to market by providing applications and providing Application Programming Interfaces (APIs) through the Web. These APIs are to be reused by other applications. Moreover, the rise of SOA concept and service composition are more or less supported by the mitigated success of standardization efforts. Actually, whether to describe services, publish them on the Web, or compose them, a multitude of standards (WSDL, UDDI, BPEL, ...) are proposed and supported.

Besides, Web and Telecom convergence has established new business rules that made the traditional actors in the Telecom world reconsider their strategies in order to stay competitive and take into account the rapid rise of the WWW business actors embodied by Google. In this context, Alcatel-Lucent has launched the new strategy of "Application Enablement"[1] in order to highlight the resources provided by the network. This strategy aims to expose network and infrastructure resources as services to enable the creation of intelligent and innovative applications and solutions.

In this global context, Alcatel-Lucent implements, at business and research levels, its strategy of "Application Enablement". Although our interest is the research aspect, we briefly mention some business initiatives implementing this strategy. Firstly, in order to bring together an ecosystem of service providers, enterprises, and developers to drive the creation of innovative applications, Alcatel-Lucent has launched the "Open API Service"[2] website. This website provides a platform for

[1] http://www.alcatel-lucent.com/applicationenablement
[2] http://openapiservice.com

developers that facilitates development and testing of new applications through innovative APIs bundles coupled with a revenue sharing model that eliminates upfront costs. To strengthen its position, Alcatel-Lucent went off realizing a series of acquisitions of various technology innovations.

Among these acquisitions, ProgrammableWeb[3], the technology industry's universal source for Web APIs used by application developers to build Web, mobile, and other connected applications that serve consumers and the workplace. Another acquisition is OpenPlug[4], a company that offers a widely deployed set of software tools used to create new mobile handsets and the next generation of mobile applications. On the basis of those platforms, and associated with developers community events, many contests and workshops have been performed in order to bring together not only developers but end-users communities as well, in fast growing applications ecosystem. All of these innovative service composition technologies and platforms have been continuously and relentlessly promoted by Alcatel-Lucent to gain acceptance within the developers and end-users communities. Developers' community events, workshops and contests have been the vehicle through which the promotion of those platforms occurred. This ultimately led to a complete fast going ecosystem of the developers to the end-users communities.

At the research level, in order to embed this strategy, the Bell Labs developed many paradigms, such as "Everything As A Service" (EaaS) that basically considers any digital resource as a service. For instance, communicating objects, streaming video, or even a network bandwidth are considered as services and could be composed according to a given logic. In particular, a watchful look is dedicated to service composition from the end-user perspective in term of environment, use cases, and applications that may result. The Bell Labs participates in several European research projects related to this paradigm. As an example, we cite the "Do It Yourself Smart Experience" and the SERVERY projects. Both projects aim at facilitating service composition by end-users. Additionally, SERVERY particularity highlights the services provided by the networks (Telecom). Within this framework, the SocialCom department, which sponsored the current work, has addressed the EaaS paradigm from end-user's social dimension perspective. Indeed, by considering the proliferation of communication means and socialization of the Web, the user is immersed in a digital social environment. Precisely, considering service composition as a social activity takes a special interest. In fact, social networks analysis (SNA) techniques can be adapted to assist end-users in the service composition within a Web social environment. This thesis, conducted in this department, attempts to meet these aspirations by providing theoretical and practical responses beyond the state of the art.

[3]http://www.programmableweb.com/
[4]http://www.openplug.com/

1.2 Research context and problem statement

Creating services from scratch, to satisfy customers' growing demands, needs too many programmers, costs too much, and takes too long to reach the market. Creating value-added services by reusing existing ones, known as service composition, represents an alternative way to meet this increasing demand by allowing to produce numerous services quickly. Another important goal of service composition is to make possible services customization according to end-user's preferences. In other words, it provides end-users with personalized and user-centric services.

Web services composition has been a key issue in service sciences and heavily investigated from both industrial and academic perspectives [ter Beek 2007][Yuan 2007]. Indeed, three main approaches have been defined in the service composition research community in order to deal with service composition from an end-user perspective: (i) manual, (ii) automatic, or (iii) semi-automatic approaches. In the first approach, the user has to compose services by writing entire programs (without any automated assistance). This program has to embed Web services invocations (calls) according to the composition logic. In this case, end-users are required to have high programming technical skills, which makes this approach very limited. The goal of the second approach is to automatically build composite services in order to match end-user's request which is supposed to be expressed through a user friendly interface or even automatically computed on the basis of information gathered from his context. This approach has to handle indecision problems [Balbiani 2006], that will ultimately need to involve the end-user within the composition task. This leads to the semi-automatic approach, the third and last one, which aims at providing end-users with an enhanced service composition environment. This environment offers support for automated processing of some composition tasks where the end-user operates in a more or less involved manner. Semi-automatic composition, somehow, comes as an alternative approach by focusing on particular issues, for instance the difficulty of selecting a relevant service among the many available ones of the same category.

Initially, Web service composition was reserved for expert users like programmers. However, with the emergence of Web 2.0, it is becoming more important to make the composition process much more end-user friendly. Consequently, the service composition concept has left the frontiers of the enterprise to reach the Web. We are particularly interested in this approach as described in the following.

We should specify at this stage that we here address a particular type of users who have commonly been referred to as the "Net Generation", "Digital Natives", or even "Generation Y", and are claimed to be very different from their predecessors in their familiarity with technologies and the regularity with which they use them [Palfrey 2010], and that we call here end-users. In the Web 2.0 context, one of the interesting properties of end-users is their ability to produce or participate in producing content. In fact, the Web 2.0 has brought a set of different technologies dedicated to end-users with limited technical skills (even in an enterprise context). It became then very easy for such users to publish or annotate resources (User

Generated Content (UGC)), and to stay in touch with their social relatives with the emergence of social networks and collaborative environments.

As mentioned above, the semi-automatic approach has the main advantage of making the user participate in creating composite services, which are called Mashups in the context of Web 2.0. Mashups creation environments, a new paradigm in Web 2.0, enable end-users to easily create Web-based applications and services (Mashups) that address their specific needs and interests [Liu 2007, Yu 2008]. Several IT and Web actors provide Mashup creation Web environments (named also Mashup editors) as easy-to-use service composition tools. The majority of existing Mashup editors (see section 2.6) offer many user-friendly features in order to assist the end-user and fit with his limited skills, as highlighted in [Grammel 2008, Ennals 2007a]. These feature include abstraction, visibility, juxtaposability, and community features. Those existing features are involved in the pre- and/or post-composition process, particularly social features like annotating and ranking services. In other words, those features can be leveraged for the benefit of the service discovery and selection phases without taking into account the current composition context as explained in Section 2.6.4. By contrast, in the composition process itself, those existing features do not currently provide any direct support to end-users. In this specific area, and despite some existing works that provide support to the composition process, we believe there is still a gap to fill in order to consolidate and provide more support to the end-user, especially by benefiting from the social environments in which the end-user operates. In other words, the objective is to prevent having a one way task ,i.e. users tagging for simply searching, but restitute that information and effort to the user in a better way for a better support.

The problem, that we are interested in, is how to facilitate the service composition by the end-user, characterized with limited technical skills, in the context of the explosion of the number of services over the Web. So the challenge is to fill the gap between an environment dedicated to the end-user and the technical complexity of the composition of Web services (standards, scripts, etc.). In order to address this general issue, we started by identifying the obstacles that hinder the development of service composition in the population of Web end-users and then propose solutions to each one of them. These challenges are primarily related to: (i) the level of abstraction of end-user frameworks for service composition, considered as being low and based on end-user programming concept instead of the combination of functionality (services); (ii) and the selection of services rendered difficult by their large number.

Our proposals to answer those issues do not come as monolithic or systemic solutions but have to be considered as contributions to existing works. Indeed, the existing works (Chapter 2) are already providing some partial but necessary solutions as building blocks to solve the problems mentioned above. In fact, many studies have been conducted to establish best practices to follow when designing Mashup creation environments in terms of basic functionalities, abstraction requirements, and GUI design. However, only few studies have focused on facilitating service selection within the large number of exposed services. Furthermore, after

reviewing the state of the art, we found that the proposed approaches do not really take advantage of the Web 2.0 environment in which the user is evolving. Particularly, the social dimension has been neglected although it contains assets of usable information to facilitate service composition. This constitutes the cornerstone of our contribution. Regarding this particular point, and in addition to the problems stated before, the problem we are addressing is:

- What would be the social dimension in this context?
- What wold be the impacts and the benefits the social dimension may bring to the services?
- In a service composition framework dedicated to end-users, how to leverage social interactions in a way to enable and facilitate composite services creation?

1.3 Contributions of the thesis

To assist the user in the task of creating a composite service (editing a scheme of composition), we propose the automatic completion of composition schemes initiated by the user. The basic idea is accompanying the user into the process of choosing services for composition by dynamically recommending the most relevant services to the considered user and the beginning part of his introduced composition schema. The first contribution of this thesis is the consideration of the social dimension in a service composition environment in order to facilitate the editing of the composition schema for end-users. In addition to the interactions that occur in the margins of the composition process, i.e. the interactions between users and the interactions between users and services in terms of tagging and rating, the central idea of our contribution is to consider the activity of creating composite services as a social activity. For each user, this activity represents the model of his/her behavior which can be deduced by analyzing the interactions between him/her and the patterns of composition that he/she has created. The analysis of the behaviors of different users can reveal the implicit interests between them. Users can therefore be represented in a social network of interest. Based on this social network, we built a recommendation system that compute relevant completions to be suggested dynamically to users when creating a composite service.

Insofar as the user creates composed services within an interactive system, the interactive recommendation system's main constraint is the response time needed to suggest completions. Indeed, to satisfy the constraints of interactivity, the recommendation system should respond within a reasonable time. Even if the quality of recommended completions is important, the response time of the recommendation system has been selected to be the main evaluation criteria. This contribution has been implemented in two forms based on two completion approaches: the first approach is the step-by-step completion and the second is the full completion. The second approach has particularly required the development of new datamining algorithm (frequent pattern mining) for analyzing user-composed services interactions.

These algorithms have been compared to and outperformed existing frequent sequence and itemsets algorithms using the reference datasets.

The theoretical contributions that we have described above have been implemented in a real end-user service composition environment. Indeed, it was necessary to develop a composite service creation environment to test the proposed algorithms. The environment developed, called Social Composer (SoCo) takes the form of a website that allows the creation of Mashups (the advanced form of a composite service), and offers basic functionalities of existing Mashup creation environments such as Yahoo Pipes! or Open Mashup Studio. Basically, this environment shares more or less the same basic features that others provide: a Graphical User Interface (GUI), the ability for the end-user to link graphically basic services as an abstract description of the composed service he wants to create. This abstract description entered by the user is then compiled into a machine executable language which invokes services according to the logic described by the user. The main added feature of SoCo is the step-by-step recommendation functionality that implements the theatrical approach.

1.4 Manuscript organization

The next chapter (Chapter 2) presents the literature review, and analyses the different existing approaches and solutions. Chapter 3 highlight the social dimension within the composition process, and introduces the service dynamic recommendation concept that aims at assisting the user during the task of composition. This chapter outlines the concepts and basic models of this proposed contribution. Additionally, it presents the step-by-step solution as the primarly approach. Chapter 4), presents the description of the full Mashup completion, as a second instantiation of the dynamic service recommendation concept. This chapter focuses on the evaluation of the full Mashup completion approach that is based on a novel frequent sequence mining algorithm. Chapter 5 describes the proposed *SoCo* Framework, its basic features, design and implementation. Finally, our conclusion are stated in chapter 6 after recapitulating the contributions and obtained results that offer a starting point for future perspectives of this work.

Chapter 2

State of the art

2.1 Introduction

This chapter reviews and analyzes the concepts and existing works related to the composition of Web services by the end-user, drawing the useful conclusions in order to position our contributions. First, we review the concept of Web services composition through the basics of Web Services and Service Oriented Architecture (SOA), and provide some classifications based on different criteria. Then, we present a detailed review of the composition tools of Web Services by the end-user particularly Mashups. In that regard, we analyze the few existing mechanisms of assistance and support to end-users, and point out the lack of such features. Finally, we conclude by summarizing the main ideas that emerge from the overall analysis of the chapter.

2.2 Web Services

Nowadays, the concept of service is so pervasive that we talk about Science of Services [Spohrer 2007]. This domain combines the understanding of organizations (enterprises or institutions) and humans with business and technological sciences. In this regard, a service is defined as *"Any act or performance that one party can offer to another that is essentially intangible"* [Kotler 1984], in contrast with the physical industry (manufacturing and agriculture). More widely, Zeitham et al. [Zeithaml 1996] state that *"Services are deeds, Processes and Performance"*. By grouping existing definitions in an synthesized one , we propose the following definition:

Définition 1 *"A service is an intangible provision, composable, expressed in a perceptible manner, which, in a predetermined operating condition, is a source of value for the consumer and the supplier (service provider)".*

This concept is much more prevalent in the IT world where people speak of Web Services. Web Services are platform-independent software, available in distributed environments such as Internet. They are mostly used in enterprise context for application integration and streamlining B2B, where they enable developing applications by assembling existing Web services translating the Service Oriented Architecture (SOA) philosophy. Indeed, Web services are the most significant achievement of the SOA in which applications are self descriptive and low-coupled modules. They are defined by a set of standards that allow us to describe software interfaces and access

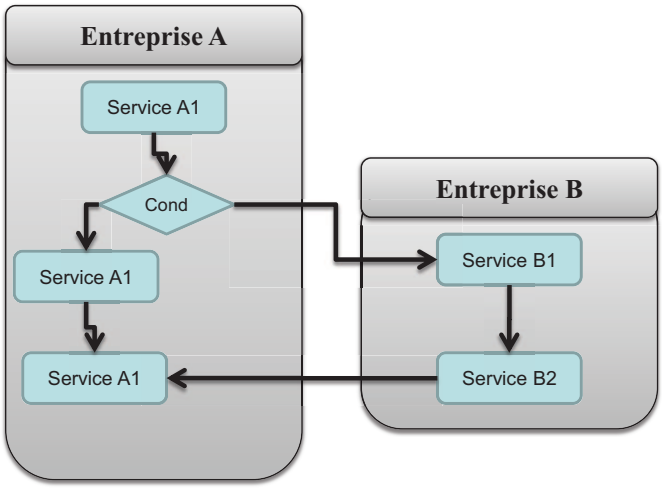

Figure 2.1: Illustration of SOC paradigm through outsourcing example

functions on a network using XML messages [Kirda 2001]. Web Services related underlying standards and technologies (such as WSDL, UDDI) are exposed in section 2.5.1.2. Basically, World Wide Web Consortium (W3C)[1] has defined Web service as the following:

Définition 2 *"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 conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".*

The next section introduces the SOA concept that relies on Web Services as the basic building blocks in a structured architecture.

2.3 Service Oriented Computing

The reference architecture SOA is conceptually derived from the Service Oriented Computing (SOC) paradigm[Papazoglou 2003]. By analogy to operating systems where the concept of "a file" is omnipresent, the SOC paradigm advocates the use of the concept of a service (not just Web Services) as a building block in any information system project. This paradigm found its dedication in integrating enterprise applications due to business needs (merge, acquisition, consolidation, outsourcing ...) which is definitly replicating the concept of business services within the IT world. Further than technological and compatibility constraints, this paradigm goal is to surpass the silo-based information system model towards a systemic (holistic)

[1] http://www.w3.org/TR/ws-gloss/

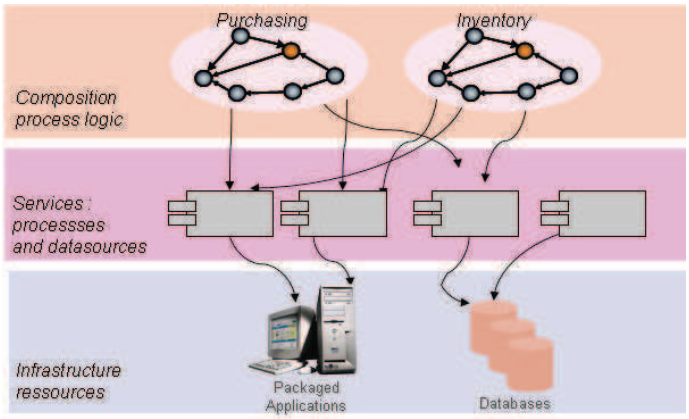

Figure 2.2: A lyered view of SOA Concept

model where inter-system transactions happen seamlessly. Through this paradigm, one (organization) can offer, find, use, and compose services according to its own needs and business requirements. Figure 2.1 shows an example of enterprise "A" outsourcing two services from enterprise "B" thanks to the SOC paradigm. SOC gives rise to several research issues including the composition of services that forms the main topic of this thesis.

2.3.1 Service Oriented Architecture

One of the most successful instances of the SOC paradigm is SOA model. SOA provides a simple model of programming and deployment of applications based on standards that run through the Web infrastructure. This architecture uses Web Services as building blocks. Basically, it defines a software architecture in an open information system, a set of components with well defined roles. This architecture allows presenting the organizations' business processes as services based on a modular architecture where each business basic functionally is represented by a basic service. The components defined within this architecture allow establishing this information system structure and integrating its services into workflows that translate more or less complex business processes. Through a layered view, Figure 2.2 illustrates how business processes are represented through the system based on basic services in an SOA.

The SOA reference architecture was first introduced in [Erl 2005], then was adopted and integrated by many standardization bodies[2] (OASIS, OMG, The Open Group...) relying on the same main principles illustrated in Figure 2.3. As mentioned before, the basic notion is a Web Service which represents a function encapsulated in a component that can be invoked (queried) using a query consisting of one or more parameters and providing one or more answers. Ideally each service must be independent of the others to ensure its reusability and interoperability. Service

[2]http://www.opengroup.org/onlinepubs/7699909399/toc.pdf

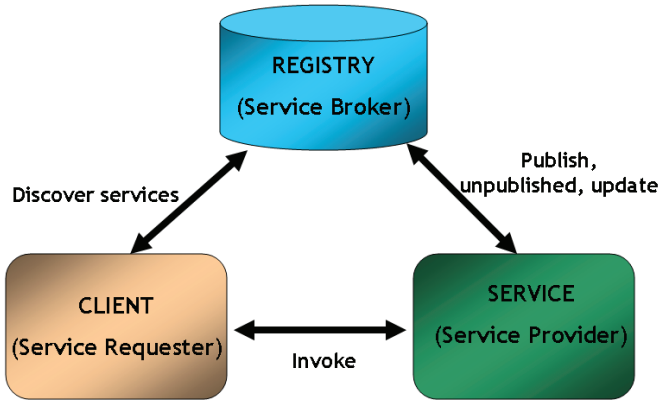

Figure 2.3: SOA basic architecture

Oriented Architecture main actors are:

- *The service provider* (or a third party mandated by him) has in charge the service creation, deployment, description, and then publication through the repository (registry) publication interface.
- *The service repository* which hosts the description of services that have been published by the service provider, and offers the possibility for clients to seek for a specific service among those available and to access service descriptions.
- *The client* (service consumer) should be able to look for services described in the service repository, and select those of interest to him. Based on a service description, a client should be able to invoke this specific service hosted by the service provider.

Then, SOA defines a set of operations and roles as follows (those defined operations and roles are based on standards that are described in Section 2.5.1.2)

- The description of the service consists of enumerating the input parameters of the service as well as the output parameters (type of data). The primary format for describing services is WSDL (Web Services Description Language) standardized by W3C.
- The service publication is to publish in a registry (or repository) services available to clients (service consumers).
- Service discovery includes the ability to search for a service among those that have been published. The primary standard used is UDDI (Universal Description Discovery and Integration), standardized by OASIS.
- The invocation consists in the customer query (connection) and interaction with the service. The main protocol used for the invocation of services is SOAP (Simple Object Access Protocol) presented in 2.5.1.2.

The following section describes and analyzes related existing research from the service composition point of view, the main research topic of our work.

2.4 Service Composition

Creating value-added services by reusing existing ones, which is well known as service composition, has been a key facet for service delivery both for IT and Telecom worlds [Yuan 2007]. Numerous services need to be produced quickly because of the growing demand of customers. Creating services from scratch, to satisfy the increasing demand, needs too much programmers, costs too much resources, and results in too long time to market.

In this context, using the services composition approach can offer a good opportunity to fix those issues. The purpose of services composition is the reuse of existing services to create new ones. This optimizes the development cycle and deployment of innovative services. Another important goal of services composition is to provide the ability to customize services according to end-user's preferences. This approach provides end-users by personalized and user-centric services. With the generalization of the internet, we are witnessing in the recent years the evolution of the service composition paradigm, initially dedicated to a restricted audience of IT specialists for business application integration (software architect, developers,), to a broader audience of Web users.

We can already see the two different dimensions that the composition of services includes: organization (EAI) and end-users dimensions. From the EAI perspective, existing conducted researches aim to overcome mainly the technological constraints by defining a range of standards and protocols for information engineering under the banner of SOA principles. This dimension dominates the majority of research works on services composition. The second dimension represents the challenges in integrating the end-user in the process of composition. This last dimension is relatively growing and taking more space in the area of services composition. This evolution is taking place at the end-user level. It is driven by his needs to customize services with an emphasis of the Internet environment context that encourages users to share, create and comment on content. Indeed, an innovative concept has recently emerged in the Internet, concept that allows end-users to create and share their own services from the composition of other services [Yelmo 2008].

Next, we review the research challenges that rise in the service composition topic through the main themes of SOA and the two dimensions mentioned before. Figure 2.4 (from [Papazoglou 2003]) represents an extended architecture of SOA in an information system. It schematically illustrates the different levels of SOA within which research contributions have been made. Proposed technologies (standards and protocols) and formalisms for each challenge are detailed in Section 2.5.

Firstly, service composition involves methods, mechanisms and tools that allow the expression of needs whether at the enterprise level for business specification, or at the end-user level. In this regard, numerous formalisms and tools have been pro-

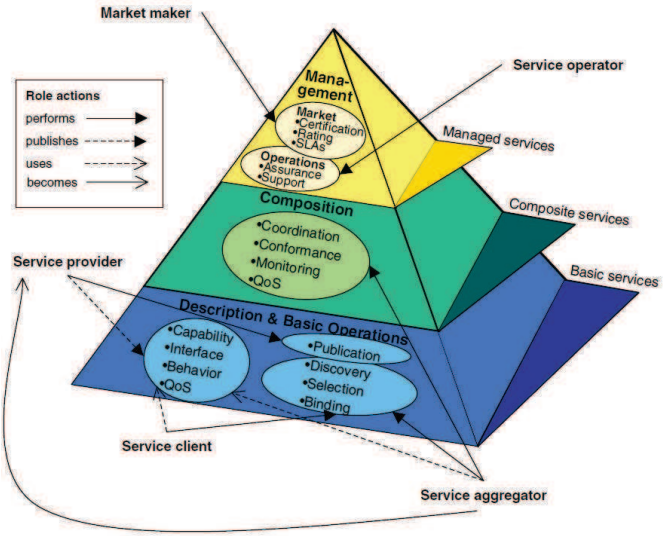

Figure 2.4: The Extended Service Oriented Architecture emphasizing main research topics

posed. Some of these tools are based on formal models. Business Process Execution Language (BPEL) is by far the reference in the field but remains unsuitable for the intended end-user (non-developers). Other tools are listed in Section 2.5.1.2 within their corresponding category.

After the expression of needs, the process of composition consists of selecting the most suitable services, and then scheduling (arranging) them in the most appropriate schema in order to fit the logic of the expressed need. Once the composition schema is defined, the resulting composed service needs to be deployed. The deployment could take the form of a choreography or orchestration of services. After the deployment operation, tools and control measures are implemented to monitor the various performance indicators of the deployed service. This operation is called monitoring. The overall composition process is illustrated in Figure 2.5. In the following, we detail each step from the research point of view.

2.4.1 Main requirements and research topics and challenges in services composition field

This subsection covers the service description, publication and discovery; the composition description and optimization including services interoperability; and finally the composed service deployment and monitoring.

2.4.1.1 The description of services

It is clear that the service description plays an important role in the composition process. A well-described service increases the relevance of its selection as well as the consistency (correctness) of the resulting composition pattern. Indeed, a service is

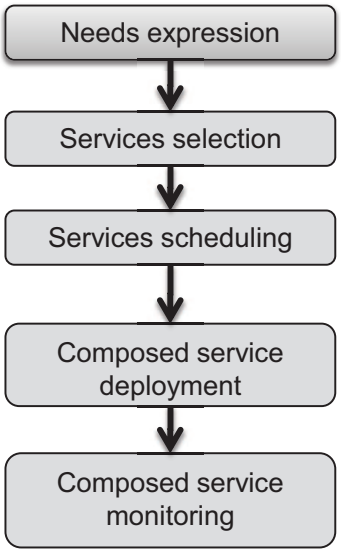

Figure 2.5: Illustration of the Composition Process steps

represented by its description which correspond to the functional and non-functional priorities.

The functional properties, as their name suggests, relates to the functionality delivered by the service. It includes descriptions of the input/output parameters and logic function (business) that the service performs. For example, a service whose logic function is sending an SMS has as input two strings: the number of the recipient and the message body. The description of non-functional properties is an important aspect in the process of composition. Indeed, this part of the description indicates for instance the availability of the service, response time, or even its business model (for example the rates per hours). For instance, for the SMS service, non-functional properties could be the business model (prices, promotions, ...), and quality of service (the maximum delay of delivering a message, ...).

The functional and nonfunctional descriptions generate non-insignificant complexity in the composition process. Thus, several protocols have been proposed where the functional aspect is predominant compared to the non-functional aspect. For example, WSDL (the current reference) is used to express the operations provided by the service. Web Ontology Language for Service(OWL-S) and Web Service Modeling Ontology (WSMO) add a layer of semantic description based on either domain-specific or general ontologies to assist service discovery (see Section 2.5.1.3).

In addition, all proposed protocols and languages (described later) were designed from the perspective of information systems and are intended to experienced users. The semantic description, especially tagging techniques, contributes, not only for better interpretation by machines through reasoning, but for bridging the gap between service description technologies and end-users as well.

2.4.1.2 The publication and discovery of services

Service publication and discovery are two important operations for the composition process and particularly for the selection of most relevant service. The publishing operation essentially raises issues of data and information engineering summarized in databases technologies and access means to populate those databases with services description. Service discovery, based on those databases technologies, have to provide not only access means but should guarantee access to the services descriptions that best fit the selection criteria (the request). This actually constitutes an optimization problem. From the perspective of SOA specifications, UDDI technology with its variants is the reference. Semantic technologies are also an alternative for optimizing the discovery and selection services in the composition process.

2.4.1.3 The Efficiency of the composition process

The heart of the service composition is the selection and scheduling of services to match the description of the service we would like to compose. This description should provide the hints needed to form the composite service schema. Several approaches and technologies are possible. For this purpose, many standards have been proposed to explicitly define the description of a composition pattern namely BPEL4WS, BPML, WSCI. Less explicit tools, based either on textual or graphical interfaces, have been proposed to allow the definition of the composed service logic. For the automatic approach, the logic of composition is formed based on information taken from the user context.

2.4.1.4 Interoperability, execution and monitoring of composite services

Interoperability between services is also a key issue in service composition. Factually, a composite service is represented by a composition pattern which reflects the logic of this service. This logic includes the information flow between services and settings. Two schemes of interoperability are defined in the state of the art of choreography and orchestration [Peltz 2003] (described below) where services communicate with each other through standardized languages. The defined composite service has to be defined according to the interoperability schema. Based on this schema, monitoring tools are designed to gather information about the composite service running state and issues that could occur due to the unpredictability of external partner services or unexpected behavior of composite services [Pistore 2004].

2.5 Taxonomy for services composition

Composition of services in particular, and the Service Oriented Computing in general, have been a fairly productive area of research. Contributions made in that area vary according to addressed issues and approaches. Indeed, and according to the previous section, these contributions may include data engineering techniques for service description languages, protocols for publications and discovery operations,

optimization of services selection and scheduling, and deployment and monitoring of composed services. In reality, these contributions are part of more comprehensive approaches that drive to a more coherent reading of these contributions. We have therefore identified many perspectives of the problems. Thus, it is possible to address the services composition from the system's perspective. This represents either semantic, formal or data engineering techniques and technologies used to address the problem. Another perspective is engendered from the user point of view. Basically, the user has to define the composition logic in a more or less explicit manner based on the provided tools. In this scope, automatic, manual or semi-automatic approaches are identified. The rest of the section details each of these approaches in order to position our contributions among the state of the art.

2.5.1 System perspective

In contrast to the user's perspective, the system perspective provides details about the techniques and mechanisms used to achieve services composition in terms of service publication/discovery, and scheduling and deployment. In this perspective, we identify three non-exclusive approaches: *Formal approach* which provides the tools and formalisms which allow for example the formal validation or verification of a number of predefined properties; *Structural approach* which looks to establish data structures and access methods in formal operational protocols and languages that are often used by other approaches; and finally *Semantic approach* which brings semantics to improve and optimize composition's operations mentioned above. The structural approach.

2.5.1.1 Formal approach

Formal models can be used for automatic or manual modeling of composed services. The formal description techniques allow the use of methods and tools to make the development cycle of services more reliable, faster and cheaper. Formalisms for specifying these services are based on precise and mathematically-based syntaxes and semantics. Developing models will apply methods and tools in three major phases of development lifecycle of the service: (i) the verification, (ii) the automatic or semi-automatic code generation, and (iii) the generation of the test benches.

The objective of the verification phase is to improve the reliability of the process of developing an implementation by ensuring that the formal model on which the implementation is based is valide with respect to a given set of properties. These properties are represented in the form of logical properties or sub-sets of an automata [Clarke 1999] [Gordon 2000].

The formal test phase is a set of executions of specific test sequences on the implementation. Test sequences are obtained from the formal model by trying to cover all aspects of the service compound. Tests can be generated automatically or semi-automatically based on criteria, goals or assumptions. There are many stages and types of tests in the development process of a service: the conformance,

the interoperability, the unit test, the integration tests. The majority of these procedures are standardized or described in some reference software development lifecycle management [Fernandez 1997].

Depending on the degree of modeling, it is possible to generate code for all or part of the application model. The more semantics modeling language is precise, the more code generation is complete. For instance, in the UML formalism, semantics are weak or nonexistent. At best, it will generate the interfaces from the model. Many modeling languages have been standardized and are based on various concepts such as automata, the states/transitions systems, temporal logic, interaction, etc. [Zhao 2006]. Some of them are briefly described in the following.

The Specification and Description Language (SDL) [Broy 1991] is a standard of the International Telecommunication Union (ITU-T), which aims at describing communication protocols. Even if the SDL language is a modeling language that was initially used for communication protocols description, it is more generally used for modeling real-time applications. This is due to the syntax of the language which describes a service using the following:

- Description in the form of state machines;
- Exchange of information via asynchronous messages;
- Use of timers;

The ECharts formalism is defined by ATT laboratories. It has been introduced in 1999 and continues to be supported until today. The ECharts objective is enabling to easily describe modules, services, composite services that are verifiable, maintainable and reusable. The modeling aspect is captured by the fact that ECharts is based on a state machine formalism. In ECharts, transitions may have priorities. This mechanism allows flexibility in the reuse of models from a given machine, new transitions can be added in some states with a higher priority than the existing transitions. These will be executed modifying the behavior of the initial machine [Bond 2006].

In summary, this approach offers formal methods to verify or test software components (services) automatically, and are necessary for the composition of service. However, those formalisms are too specific and require high technical and mathematical skills, and therefore could not be directly used by end-users, but can intervene at underlying layers to ensure some required properties (such as security properties).

2.5.1.2 Structural approach

This approach is more about providing formalisms and tools to describe service interfaces (inputs/outputs) and behavior in order to compose services and create new ones. For instance WSDL provides XML-based syntax for describing services and BPEL provides a framework for orchestrating services. By contrast, formal methods

(Automata, Petri Nets) provide tools to improve the reliability of the process of developing an implementation by ensuring the conformity of the formal model on which the implementation is based to a given set of properties. We present hereafter a number of standards in the area of Web services that allow implementing the SOA concepts [Curbera 2002]. This includes WSDL, SOAP, HTTP, XML and UDDI.

Web Services Description Language (WSDL)[Christensen 2001]: It is a XML-based language that is used to describe the Web service. In other words, it describes: what can a web service do, where it is, how to access it, and in which format. The WSDL provides features for the service naming, the operations naming (input parameters and responses organized in the form of messages). It also contains detailed information about the used communication protocol (often HTTP), information on the technique of data encoding and network address in the form of a URL. It does not contain semantic information about operation, and there is no notion of order on the invocation process. The client can use SOAP to actually call one of the operations listed in the WSDL file.

The Universal Description Discovery and Integration (UDDI): The service directory (also called repository or registry) is the place where the services are registered. The SOA concepts can be instantiated by using the standard UDDI[3] as service directory. The UDDI is structured in three pages (components): White (information by name), Yellow (Information by category) and Green (Service provided by WSDL). It is designed to be interrogated by SOAP messages and to provide access to service description documents (WSDL).

Simple Object Access Protocol (SOAP)[Box 2000]: This is the technical container which encapsulates the XML message according to the SOAP exchange standard. SOAP is itself represented in XML with a header part and a part that corresponds to the application payload (called Body, but also called Payload). SOAP header part is optional and generally used to transfer data authentication or session management. These are aspects that are borne by the underlying protocol. Body Part in turn is in charge of encoding the names of operations and their parameters and returned results. The SOAP is typically deployed over HTTP but can also operate over SMTP or JMS. The SOAP is also defined by an envelope which allows describing the specification of the name space. There are also features for error handling.

Business Process Execution Language (BPEL): The BPEL (actually BPEL4WS) has been introduced by OASIS standardization group as the successor of XLANG and WSFL. The BPEL is an XML representation used as an instantiation of a service-oriented architecture (SOA) concept. Specifically, in the SOA, the enterprise applications are managed from a common platform to enhance dialog between

[3] http://www.uddi.org

applications, and their integration. BPEL organizes the dialog between the different applications of SOA by invoking basic services according to a predefined schema 2.6.

```
namespace pns = "http://example.com/loan-approval/";
namespace lns = "http://example.com/loan-approval/wsdl/";

@type "http://schemas.xmlsoap.org/wsdl/"
import lServicePT = lns:"loanServicePT.wsdl";

@suppressJoinFailure
process pns:loanApprovalProcess {
    partnerLink customer = (lns:loanPartnerLT, loanService, null),
    approver = (lns:loanApprovalLT, null, approver),
    assessor = (lns:riskAssessmentLT, null, assessor);
    try {
        parallel {
            @portType "lns:loanServicePT" @createInstance
            request = receive(customer, request);
            signal(receive-to-assess, [$request.amount < 10000]);
            signal(receive-to-approval, [$request.amount >= 10000]);
        } and {
            join(receive-to-assess);
            @portType "lns:riskAssessmentPT"
            risk = invoke(assessor, check, request);
            signal(assess-to-setMessage, [$risk.level = 'low']);
            signal(assess-to-approval, [$risk.level != 'low']);
        } and {
            join(assess-to-setMessage);
            approval.accept = "yes";
            signal(setMessage-to-reply);
        } and {
            join(receive-to-approval, assess-to-approval);
            @portType "lns:loanApprovalPT"
            approval = invoke(approver, approve, request);
            signal(approval-to-reply);
        } and {
            join(approval-to-reply, setMessage-to-reply);
            @portType "lns:loanServicePT"
            reply(customer, request, approval);
        }
    }
    @faultMessageType "lns:errorMessage"
    catch(lns:loanProcessFault) { |error|
        @portType "lns:loanServicePT" @fault "unableToHandleRequest"
        reply(customer, request, error);
    }
}
```

Figure 2.6: A simple BPEL script

BPEL is a complete and open standard, with a lot of supporting engines. Without serious concurrent, BPEL was quickly accepted by the industry and is now the dominant technology in the field of Web service composition. It takes the form of an XML file readable in the engines of business process management. It organizes the conduct of business processes (workflow). The BPEL file is therefore on matters such as processing data, sending messages or calling a function. There are two types of BPEL processes:

- An abstract, which specifies the exchange of messages between the various parties without specifying the internal behavior of these parties;
- An executable process: that specifies the execution order of activities. Each activity represents a given process (a Web service) involved in the main composition script.

Services Composition using BPEL: The ability to integrate or compose existing services into new services is the most important functionality provided by SOAs. The service composition must be created taking into account the maintenance of services that rely on other services. The SOA offers a homogeneous environment

for the composition in a way that all parts of it are respectively described in the same way and communicating with the same standards for exchanging messages. The composition of services is achieved through a framework which consists of three parts:

- Models of composition and language: The composition of services means the creation of a workflow that defines the order in which the services are invoked, how the data is transmitted and how the logic is implemented. A composition model provides a language in which the composite service workflow has to be written.

- A development environment: This development environment consists of an editor for the language of composition sach as programming languages Integrated Develepment Environment (IDE).

- A runtime environment: A composition of services is executed by creating instances of the composition script and deploying thems in an execution environment (application servers).

There are two distinct ways to conceive a composition of services, i.e. the choreography and orchestration:

- *Choreography:* The choreography describes the collaboration between services to accomplish a given goal. The control logic of a choreography is distributed. Each service knows what to do and which service to contact. Choreography languages allow the description of protocols that the participants have to follow. In [Turner 2005], two main choreography approaches are defined: (1) the global model, which describes a protocol from a global view of the messages exchanged by all parties and (2) the interaction model in which each service describes its temporal and logical dependencies among the exchanged messages, which is similar to defining a kind of interface. WS-CDL (WS Choreography Description Language) adopts the global model, while WSCI and the abstract BPEL process are based on the interaction model.

- *Orchestration:* The orchestration of services allows the definition of the sequence of services according to a predefined schema, and run through "orchestration scripts". These scripts are often represented by business processes or workflows inside or outside entreprises. They describe the interactions between applications by identifying the messages and by connecting the logic and invocation sequences. Orchestration describes the way in which Web services can interact together using messages, including the business logic and execution order. These could include different services from different organizations and the result could be a model of a long-term transactional and multi-stages process.

  An important difference between orchestration and choreography is that the orchestration is centralized, i.e. the process is under control from the business

perspective. However, the choreography provides a comprehensive and collaborative coordination. It describes the role of each participant involved in the application.

In summary, we have seen approaches which address the composition of services in terms of techniques, protocols and standards. The fact remains that these protocols and standards all require technical skills that end users do not have. However, this approach, more practical, results in simpler tools covering all these protocols. For instance it provides tools for service publication and discovery, or tools to sketch service composition BPEL scripts (Figure 2.7), that reduce the level of required technical skills but remains too complex for the end-user.

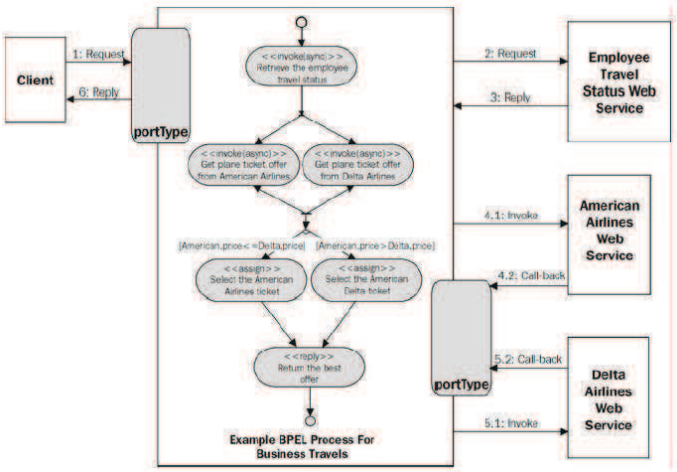

Figure 2.7: A schema simplifying a BPEL script

2.5.1.3 Semantic approach

The Semantic Web should enable greater access to services on the Web. Users and software should be able to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties, and should be able to do so with a high degree of automation. Number of common standards were introduced in the world of Semantic Web services. We describe hereafter two main ones.

Web Ontology Language for Service(OWL-S) (formerly DAML-S) [Martin 2004] is a services ontology that provides a solution to these functionalities. The overall structure of the OWL-S ontology is composed by three main parts: (i) a service profile describes what the service requires from users and what it gives them; (ii) a service model specifies how the service works; and (iii) a service grounding gives information on how to use the service[Milanovic 2004]. The process model is a service model subclass that describes a service in terms of inputs, outputs, preconditions, postconditions, and, if necessary, its own subprocesses. In the process model, we

can describe composite processes and their dependencies and interactions. OWL-S also defines three model of processes: atomic, which have no sub-processes; simple, which are not directly invokable and are used as an abstraction element for either atomic or composite processes; and composite, which consist of sub-processes.

With respect to BPEL tools, OWL-S efforts are still focused on research issues and few implementations are currently available. However we can cite the OWL-S Editor [Elenius 2005] developed by the SRI International as a Protege[4] plugin, which provides a graphical environment for editing an OWL-S service resource, the control flow graph of a process, and "run" (test) a defined process. The OWL-S IDE project [5] is also concerned with the development of OWL-S services. The OWL-S IDE is a plug-in for Eclipse, which attempts to integrate the semantic markup with the programming environment. Developers can write their Java code in Eclipse, and run a Java2OWLS tool to generate an OWL-S "skeleton" directly from the Java sources. The idea of integrating SWSs more closely with the programming environment used to develop the service implementations is a powerful feature. However it will often be more useful to generate the semantic markup before the Java (or other) code, as the semantic descriptions can be seen as a higher level of abstraction of the programming modules. The OWL-S IDE does not provide any graphical visualization of services or processes.

Another OWL-S editor is provided by the University of Malta [Scicluna 2004]. It is a stand-alone program providing WSDL import as well as a graphical editor and visualization for control flow and data flow. Not being integrated with an ontology editor, it shares some of the drawbacks of the OWL-S IDE, without gaining the advantage of programming-language integration.

Web Service Modeling Ontology (WSMO) The Web Services Modeling Ontology (WSMO) [Roman 2005] shares with OWL-S the vision that ontologies are essential to supporting mechanisms like automatic discovery, inter-operation, and composition of Web Services. Similarly to OWL-S, WSMO is an ontology for describing various aspects related to semantic web services. Moreover the WSMO effort defines an expressive Web-oriented language, WSML [Lausen 2005], which provides a uniform syntax for sub-dialects ranging from description logic to first-order logic. Like OWL-S, WSMO Web services specifications is based on the service capability which consists of inputs, outputs, preconditions, and results. Unlike OWL-S, WSMO does not provide a notation for building the composite processes in terms of control flow and data flow. Instead, it focuses on specification of internal and external choreography and orchestration using an approach based on abstract state machines (with guarded transitions).

The service basis of WSMO is defined in the same way as the one of OWL-S. This task is achieved by a mediator, which is a key concept in WSMO. In WSMO's approach, mediators perform tasks such as translation between ontologies, or between

[4] http://protege.stanford.edu/

[5] http://projects.semwebcentral.org/projects/owl-s-ide/, formerly known as CODE

the messages produced by one Web service and those expected by another.WSMO includes a taxonomy of possible mediators that helps to classify the different tasks mediators are supposed to solve. The definition of mediators in WSMO calls attention to some important translation tasks associated with web services. Not surprisingly, these same translation tasks are needed in support of interactions with OWL-S described Web Services. Some OWL-S based systems [Paolucci 2004] also make use of mediator components. However, rather than requiring the existence of a distinguished type of entity in the Web Services infrastructure, OWL-S takes the view that mediators are services and as such these mediation services can use the mechanisms provided by OWL-S for discovery, invocation and composition. Other distinguishing characteristics include WSMO's emphasis on the production of a reference implementation of an execution environment, WSMX, and on the specification of mediators (i.e., mapping programs that solve the interoperation problems between Web Services).

WSMO instances can be created with WSMO Studio [Dimitrov 2007] which is a real complete and an open source Semantic Web Service and Semantic Business Process modeling environment. It provides support for WSMO editing with integrated WSML Reasoner, WSML text editor and validator, Choreography designer, SAWSDL editor for adding semantic annotations to WSDL documents, execution engine and many other features. Moreover it also provides a Semantic Business Process Modeling according to the Business Process Modeling Ontology, a semantically extended version of BPEL, called BPEL4SWS [Filipowska 2007].

As a conclusion the Semantic approach adds an extra layer on top of the structural approach (section 2.5.1.2) by integrating the semantic properties within the operations of description / discovery, and composition of services. With these properties it is possible to link services together semantically. For instance, it is possible to propose a schema of composition from a natural language request (see natural composer in section 2.6.2.2). This is a major step forward from the end-user's perspective. While this approach is valid for very simple patterns of composition, it is unfortunately not advanced as to allow expressing the logic of composition for complex cases.

2.5.1.4 Horizontal Vs Vertical compositions

Several recent research efforts have dealt with the Web service composition problem trying to divide it into two or more sub-problems, introducing vertical/horizontal service compositions and abstract/concrete services concepts. In [Hassine 2006], authors argue that automatically composing Web services involves two main processes of composition, the vertical and horizontal compositions. Vertical composition, aims at finding the "best" combination of abstract Web services, namely, the abstract workflow to achieve the main objective, while satisfying all restrictions interdependently. Abstract services refer to each of the sub-tasks (abstract functionality) when joined together represent the main objective of the composite services. Each abstract service can be executed by many equivalent Web services called concrete

services. Consequently, the horizontal composition goal is to find the "best" concrete Web service among a set of functionally-equivalent services available on the Web. Those functionally equivalent services represent a web service community (concept introduced in [Maamar 2007]). The choice of a concrete Web service is done based on functional (eg, on inputs) and / or non-functional attributes (eg, related to QoS).

The main advantage of distinguishing between these two processes of composition is to simplify the Web service composition problem to reduce the computational complexity. It provides an easier way to consider user intervention, so the user is able to modify / adapt the abstract workflow where necessary [Greenshpan 2009].

Most of the described work considers Web services from more a system perspective. These last years, end-users have become the center of different technologies. Web services could not resist to this phenomena as we explain in the following.

2.5.2 User perspective

With the emergence of the Web 2.0 and the related technologies, composing services has left the traditional frontiers of enterprises. SOA concepts need to shift to this new area in order to take into account end-users which represent a new opportunity of evolution for these concepts. Actually, with the growing number of services available through the Web, the introduction of the end-users in the loop is taking more and more importance. In fact, the end-user needs to use a certain kind of composition in different situations especially that Web 2.0 has brought a set of technologies making it easy to create or collaborate on new services or use others services for e.g. Mashups.

This new perspective brings a totally new "breathing space" for research in the area of services composition. In this section, we discuss of existing research from the user's point of view. It will show in particular the limitations of conventional methods (called manual) because it requires significant skills in languages, formalisms and protocols related to the composition of services reserved to experienced users (developers). In addition, this section highlights the limitations of the automatic approach that decouples the composition from users. This approach is facing complex problems that are hard to resolve (even undecidable in some cases [Balbiani 2006]). The hybrid approach, called semi-automatic, involves the user in the composition process and represents an interesting alternative. Eventually, it provides tools for the simplification and abstraction of the different tools and techniques of composition, and also provides functionalities to support the end-user.

2.5.2.1 Manual Web services composition

The first approach is based on composing manually multiple services by the user. This operation must be entirely and manually performed by the end-user. Formal languages like SDL can be used. Alternatively, textual editors and GUI-based tools that are based on technical protocols and formalisms like BPEL-based IDEs can also be used. It is not necessary to mention that both alternatives require a high

level of technical knowledge and experience which are lacked by the user. Because the majority of end-users are not programmers, this approach is highly criticized for requiring an unrealistic technical level on the end-users, which makes dramatically its use limits.

2.5.2.2 Automatic Web services composition

The second approach is the automatic services composition. This approach aims at automatically building composite services that are in response to a user context or request. Except of the request, the end-user does't provide any more information to the composition process. We cite below some works that falls with the automatic approach's category and summarizes the overall landscape of contributions made in that area. The most common technique used in this approach is based on the so-called goal-driven service composition, and particularly the inputs/outputs matching. In other terms, from a final goal definition (set by the user and/or his context), this technique uses the matching between output and input interfaces data types in order to define the most likely pair of services that can be composed together. Step by step, this operation should succeed to building the composition pattern.

In [Zhang 2003], authors propose a method based on semantic matching between the input parameters (respectively pre-condition properties) of a service with the output parameters (respectively the post-condition properties) of its predecessor. In a similar way [Lécué 2008] introduces a framework for service composition based on functional aspects, in which services are chained according to their functional description. The suggested framework uses the Causal Link Matrix (CLM) formalism in order to facilitate the computation of the final service composition as a semantic graph.

On another side, context-aware service composition is considered as another way to automatically compose services. Authors in [Zhovtobryukh 2006] argue that incorporating context awareness into web service composition mechanisms increases relevance and the robustness of produced compositions. Zhovtobryukh proposes a Petri net based approach to enhance core composition mechanisms. Just like final state automata, other formal modeling tools [Milanovic 2004] are used to perform automatic service composition.

Full automation of the composition process is not without inconveniences. Pratically, in the absence of the user involvement validation, the automated operation offers few guarantees about the relevance of the selected and composed services, and can even lead to an end product that does not match the initial goal.Moreover, automation includes a significant complexity that can lead to situations of indecision (in a formal-based approach). Indeed, [Balbiani 2006] shows that checking an e-service composition model is undecidable in some cases. The authors argue that undecidability is due to unbounded FIFO queues. The transaction sequential consistency problem provides another perspective for understanding the queue effect, where independent transactions are allowed to commute.

2.5.2.3 Semi-automatic Web services composition

The third approach is the semi-automatic services composition which aims to provide end-users with an enhanced service creation environment. This environment offers support for automated processing of the composition where the end-user operates in a more or less manner. This approach gains more interest as the automated service composition approach presents serious limitations. The semi-automatic composition comes to resolve the situation by involving the end-user in the composition process by addressing particular issues, for instance the difficulty of selecting a relevant service among the many available.The semi-automatic composition has taken several forms that evolved over time. A current evolution of semi-automatic composition is what is now commonly called Mashups [Liu 2007]. This latter evolution incarnates the emergence of web2.0 and more specifically its User Generated Content(UGC) aspect.

More generally, based on exiting related works, we can see the emergence of a multitude of methods for semi-automatic composition that are identified and axplained hereafter. Generally, from the user perspective, semi-automatic service composition includes composition frameworks with graphical or textual interfaces, semantic-based tools like tagging technics, or even social features like sharing or rating services (both basic and composed). Those characteristics are detailed in the next section (section 2.6). However, beyond the simple and direct user involvement (participation) through selecting and scheduling services and still from the user perspective, we have identified three major ways of considering the user in the composition process. In fact some systems focus on single end-user by tracking his own interests or preferences and leverage them for his need. An alternative way is to consider the user as part of a community. consequently, the system tracks the interest of this community in order to build a list of preferences used to help in the composition process. A third emerging way, introduced here in this work, is a social network oriented approach which is based on leveraging the social aspect of how end-users operates in the service composition environment.Those three approaches are detailed in the following.

User-centric approach The first kind aims at building a profile of the user or involve him in the indecision stages by providing tools and interfaces to facilitate the service composition process. In this approach we can find numerous user-driven composition tools like in [Lord 2005] where semantic service discovery facilities are provided based on user preferences. A similar approach is presented in [Law 2007] where the author introduces a system called Koala (currently Co-Script). This system, materialized by a “side bar” in the Firefox browser, learns from the user behavior when processing a web page, and transform sthis behavior into a series of actions. the system objectives are (i) to parametrize the following abstract actions and makes them executable; and (ii) to allow end-users to share their composed actions. For the last case, the script can be modified by other users or adapted to their profiles. Even if this approach goal is to provide the end-user facilities and

support tools for service composition, we may notice that it is not taking advantages from the whole information available about how users use services in semi-automatic service composition environment.

Community-centric approach The second kind relies on the knowledge produced in communities or in specific-domains. A community can be involved in the process of composition through two ways. The first one is tagging or annotating basic and composite services in order to provide advanced descriptions of services (semantics, classification). This first way allows the improvement of the discovery and selection operations which could be leveraged to support users forward and downstream the composition process itself (what we name “a priori or posteriori support”); The second way of community involvement is to extract the generated knowledge in a community or a specific domain in order to define a set of rules considered as "best pratices" in this community or specific domain. These rules are used to build recommendation systems to assist users in the composition process.

In this regard, authors [Chen 2003] provide an interesting introduction of domain-knowledge for services composition. First, they explain the observed lack in the services composition structural approach (UDDI, WSDL, SOAP). In fact, this latter does not address the issue of coordination and scheduling of services. Several industry standards such as BPEL and WSFL offer solutions for “a priori” composition. According to the authors, this is unsuited to a domain-specific approach (targeting a specific area, for instance scientific computing).

The other approach addressing this problem is the semantic approach based on ontologies witch enables a “sophisticated” service discovery. Some researches describe the possibility of using this technique of discovery (semantic matching) to manage the services composition. The lack of this method is the indeterminism that may arise during the selection phase, which is based on the semantic description of service functionalities. The authors stress the fact that the e-science domain implies a certain dynamic processes that the structural and semantic approaches can not cover. Hence, the domain-specific knowledge is established to support the services dynamic selection and configuration. Methods such as CommonKADS and OilEd have been introduced to interpret the domain-specific knowledge provided by experts in a list of rules and actions. This list allows building a service recommendation system. Those recommendations can be provided to a software agent or the end-user through the development environment. This will help to pro-actively improve the services selection and composition processes.

Coming back to [Chen 2003], the author proposed a prototype for the specific domain of engineering design search and optimization (EDSO) for modeling, analysis and optimization of aerodynamic object. This prototype helps less or more experienced users to build (compose) a suite of EDSO algorithms represented in Web service form to meet their specific needs. Another study [DiBernardo 2008] suggested the same approach for services composition by upgrading this process using domain specific-knowledge (in this case it refers to a life-science domain).

2.6 Mashups Editors: An End-user Services Composition Environment

Nowadays, we're witnessing the proliferation of Web services and APIs exposed through the Web [Yu 2009b]. Services composition tools propose environment to take advantage of this proliferation by allowing users to compose services for their own interest. Beside that, Web2.0 is "cultivating" and promoting a population of creative users who generate a significant amount of content. However as we have mentioned before, end-users have no required skills to manipulate Web services. Thus, services composition platforms and tools aim at providing features and facilities to help end-users in these operations. These efforts led to the emergence of the so-called Mashups. As an introduction, a Mashups is defined as a Web application created by reusing existing Web resources considered here as services. The framework and environment used to create a Mashup is a named Mashup editor (called also Mashup creation environment or Mashup maker). This section presents existing Mashup frameworks and conducted research studies with a special focus on features related to support for the end-user.

2.6.1 Mashup and Mashup creation environment

An application that combines content from more than one source into an integrated experience or service is called a Mashup. The process of "mashup creation" can be obviously done at the level of a web programming language (e.g. php, java) by developers, or more easily done in frameworks (e.g. Mashup Editors) by end-users. Mashup is a more informal service composition. Service developers often have strong preferences with regards to their service creation environment. For end-users, a more user-friendly environment is more attractive, but will of course imply less options[Yu 2007]. Because they are very intuitive, emerging service creation tools focus on how to enable the end-user himself to create Mashups. For instance we find in the Internet world Yahoo Pipes [6], Microsoft Popfly [7], MashMaker [Ennals 2007b], [Ennals 2007a], MARGMASH [Díaz 2007] and MARMITE [Wong 2007], and in the telecom world eZweb [Soriano 2007]. This section introduces a brief description of each environment, whereas a entire section is the chapter 5 is allocated to describe the common internal architecture of a Mashup creation environment.

2.6.2 Overview of major Mashup creation environment

Mashups creation platforms supports the user in integrating and orchestrating services for his final composite application and provides an abstract layer that hides the complexity of the underlying process model (e.g. BPEL). The growing visual programming paradigm (graphical) of Mashups is the most common way to meet those requirements. Other ways are the description of the processes via a natural

[6] Yahoo Pipes, http://pipes.yahoo.com/pipes/
[7] Microsoft Popfly, It was discontinued on August 24, 2009

readable rule language occasionally called Controlled Natural Language (CNL) or the implementation of a timeline that describes the user interaction on the basis of their chronological appearance. In order to come to a comprehensive solution for the modeling process several other aspects like event-handling, dependencies between user interaction or message flows have to be considered.

2.6.2.1 Graphical editor

The Graphical mashups editor tool allows an end-user to create simple mashups by using the graphical user interface for drawing the workflow describing the logic of the compiste service. The end-user can simply drag/drop boxes representing the available building blocks (representing Web services), and connect them to indicate the flow dependencies.

Yahoo pipes! Yahoo Pipes is a Web application that consists of a graphical tool that provides end-users with the service composition capabilities (Mashup). Figure 2.8 is a screenshot of the Yahoo Pipes tool. The left side of the figure is the service database, and the right side is the composite service created by the end-user. The composite service is defined by a set of chained input/output boxes which represent service interfaces, and wires which represent input/output connection between these interfaces.

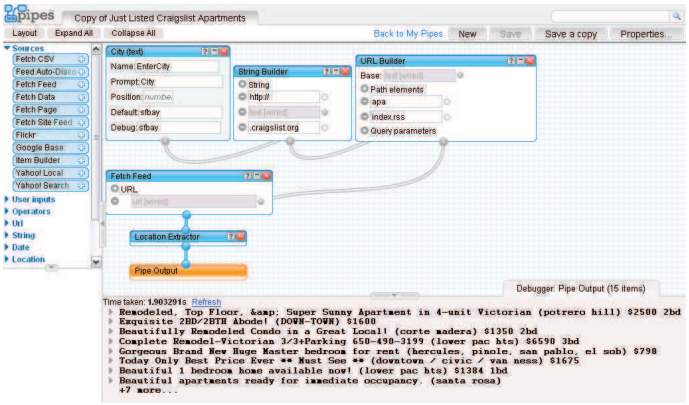

Figure 2.8: HousingMaps programmed in Yahoo! Pipes

MashMaker is a Firefox plug-in which enables the end-user to create his own Mashup from existing web sites. The most important innovation here is the data extraction from web pages that contain unstructured data. Figure 2.9 shows a "Facebook" web page in which Mashmaker component extracts automatically all addresses, names and phone numbers. Thereafter, if the user wants to display these addresses in a Map, he just has to drag/drop it in a mapping service (such as Yahoo Maps or Google Maps).

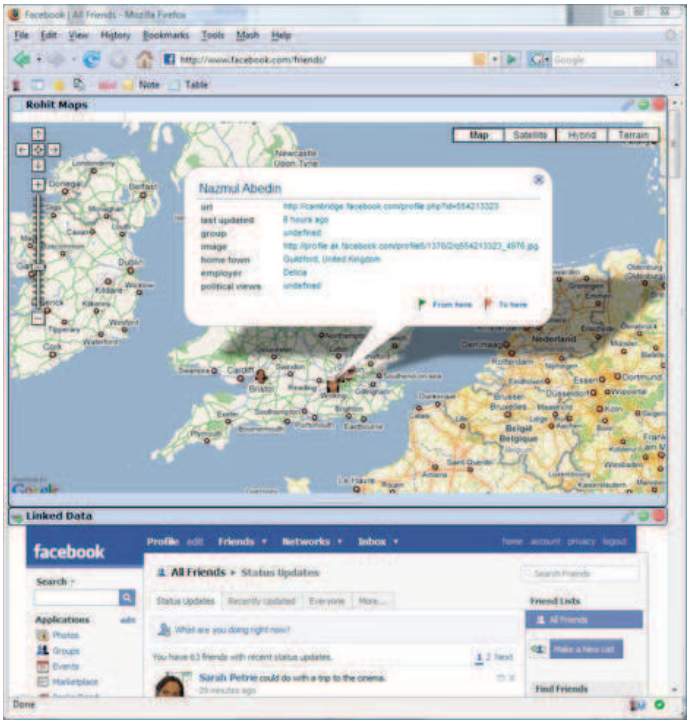

Figure 2.9: MashMaker example (over Facebook Webpage)

MARMITE is another framework which enables the end-user to create their own Mashup with an incremental execution; users can execute composite service step by step and see the intermediate results (see Figure 2.10). It is implemented as a Firefox plug-in too. Such as in Yahoo Pipes!, Marmite composite services are a set of boxes (called operators) chained with wires. However, some services can have alternative associated displays such as a map or a video player. Users can link the output of a given service with the input of an intended successor service. MARMITE authors have tested their framework on a sample of six persons [Wong 2007], where two of them are experienced programmers, and two others have experience with spdreadsheet but not with programming while the remaining two others have no experience with neither programming nor spreadsheet. As a result, three out of six did not succeed to build a composite service and those who have succeeded are those who have knowledgeable in development and one of those who have spreadsheet experience.

Open Mashups Studio The Open Mashups Studio [8] is a Mashup creation environment introduced by Orange Labs. It is based on Open Mashups Modeling (OMM). OMM is a domain specific language dedicated to applications based on component assembly. It uses a data flow paradigm to connect components and a very simple type system to represent exchanged data. As figure 2.11 shows, Open

[8] http://www.open-mashups.org/

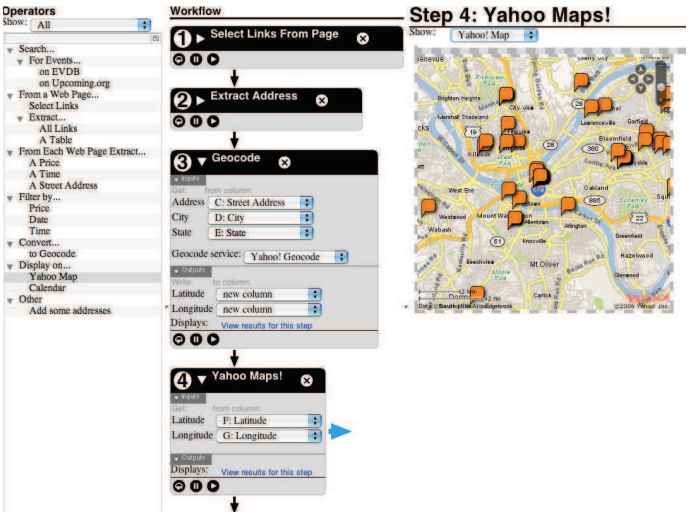

Figure 2.10: HousingMaps programmed in MARMITE

Mashups Studio is a Firefox plug-in and provides a similar environment as Yahoo pipes or Marmite. In addition, Open Mashups Studio users can specify the Mashup interface.

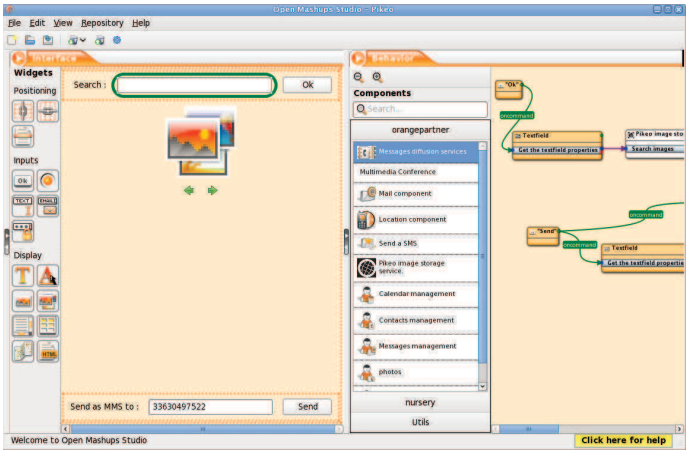

Figure 2.11: Open Mashups Studio Screenshot

2.6.2.2 Natural language Editor

The introduction of the semantic Web paradigm in service-oriented architectures enables explicit representation and reasoning about services, via a semantically rich description of their operations. Natural Language Composition focuses on the development of interactive service composition tools which use a textual user interface based on a natural language. For instance, [Bosca 2005] introduces an approach towards service selection and composition based upon the interpretation of user

requests expressed through an informal human-computer interaction interface that employs a controlled (restricted) natural language.

Natural Language Composer First introduced in [Shiaa 2008] then furthered in the SERVERY[9] project, The Natural Language Composer is used to create composite services based on the interpretation of a service request done using a restricted natural language. This interpretation is obviously constrained by the number of service components that are annotated for natural language usage. An example of sentence that can be interpreted is "Send by SMS Paris weather translated in English" which will result in on-the-fly creation of a service that will sequence three basic services: retrieval of the weather forecast from Paris, translation of a given text in English and finally SMS sending. Four main steps are done to make the system capable of interpreting such sentences and then being able to generate a service that can be executed:

1. Based on Natural Language Annotation of services in the system, the parsing of sentences is generally recursive in order to analyse then find a possible candidate among the list of existing annotated services;

2. Construction of the interpretation graph (in an intermediate formalism);

3. Based on interpretation graph, the system generates the orchestration script in order to create a sequence of service calls and the arguments appropriately assigned;

4. Deploying the script into a given execution technology.

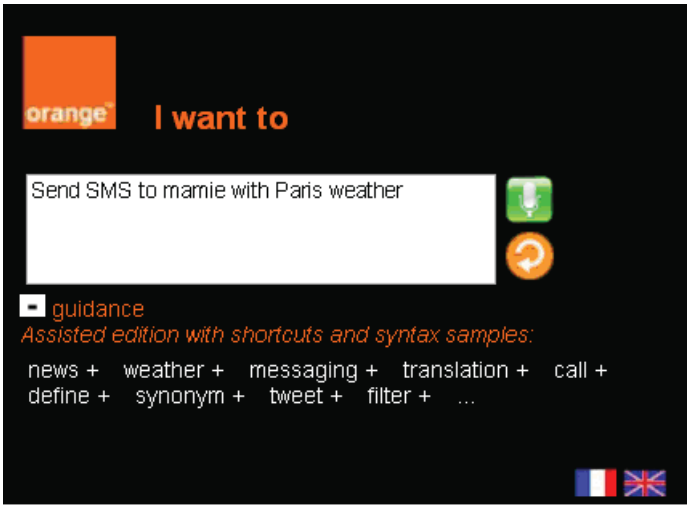

Figure 2.12: Interface of the Natural Language Composer

[9] http://projects.celtic-initiative.org/servery/

Ubiquity (from Mozilla labs [Erlewine]), an add-on for Mozilla Firefox [10], is an experimental interface based on natural language input. It is a collection of quick and easy natural-language-derived commands that act as Mashups of web services, thus allowing users to get information and relate it to current and other webpages. Users requests are based on restricted natural language command which can be extended by the community (see figure 2.13). Basically, Ubiquity commands are small chunks of javascript (as an intermediate scripting language) which can be interfaced with Web services.

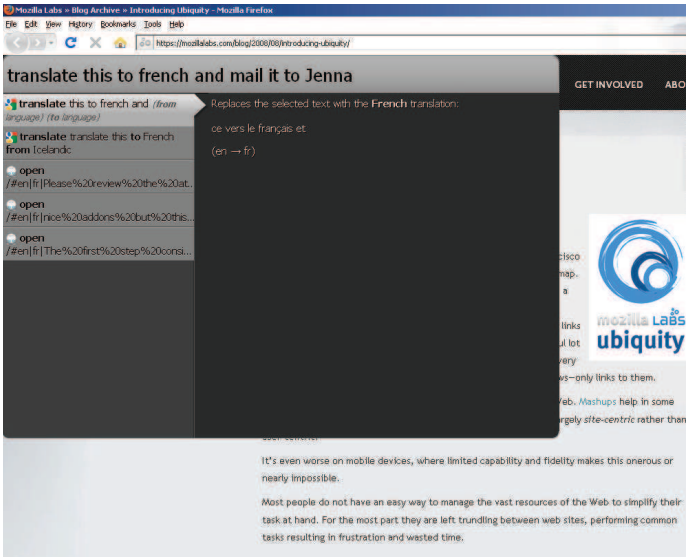

Figure 2.13: Exemple of user's request using Ubiquity

2.6.3 General properties analysis

In [Yu 2008], authors present an overview of tools and environments for creating Mashup to identify research issues. The authors point and explain the difference between Mashup development and classic component-based application development. The Mashup targets specific situational needs (typically a use case). To perform this analysis, the authors selected some Mashup creation environments (Yahoo Pipes, Google Mashup Editor, Microsoft Popfly ...). They propose to review those tools instantiated in a particular Mashup sort that is the "housing maps application". They identify the conceptual and practical features that will help to structure the analysis. At the conceptual level, two paradigms are distinguished: (i) the basic components that will be used to create a Mashup, which could be either data, application logic or user interface. this classification results in a layered view of Mashups creation that will include three layers: presentation layer (interface), data layer and functional processes layer. (ii) The second identified paradigm is the composition logic, in other words how the components are assembled. This operation

[10] https://mozillalabs.com/ubiquity/

depends on several parameters which include: the output type (data, application logic or interface), the orchestration style (flow-based, event-based or layout-based), inter-component communication (one -to-one interface, centralized communication media), and the composition execution (instance-based or continuous).

At the Mashup creation environment level, several characteristics have been identified and classified here through two concepts: (i) the user interface, which can be browser-based (sometimes plug-ins) is characterized by an environment type (drag & drop, textual or hybrid) in order to provide facilities for the user who could be a web-user, an advanced user, or a programmer. (ii) The execution environment is an important parameter to consider since it stands for delivering Mashups for users. It is characterized by the deployment type (hosting: local, Mashup provider or a third party), the integration operation which may occur on the server side (engine-based or webapp- based implementation style) or on the client side (for instance within the browser via JavaScript), and finally the scalability of the execution environment (number of data sources, composition models or users). This structured analysis allows a detailed comparison of different Mashup makers according to various criteria. However, unlike [Grammel 2008], this analysis mainly highlights the Mashup environment technical aspects from the service providers' viewpoint. It helps to identify the technical issues to consider when implementing a Mashup maker for social networking matter (e.g. scalability). Nevertheless, this study does not provide elements that help to identify requirements that each Mashup framework has to meet to become as user-friendly as possible.

In [Hoyer 2008], a similar study highlighted that Mashup creation can be separated into several conceptual levels. This has introduced the concept of "lightweight composition" which is just another name, from the end-user point of view, for Mashup creation process. Furthermore, authors have focused on Mashup makers with a special focus on community-related and social networks properties which they named "mass collaboration" features.

From the end-user point of view, Grammel et al. [Grammel 2008] investigated tools and environments for creating Mashups which they called "Mashup makers". This investigation provided an advanced analysis about the main characteristics and properties provided by these environments from the end-user point of view. Authors define a Mashup as "an end-user driven recombination of web-based data and functionalities". In this study, six Mashup makers are selected and classified into three categories: information Mashup, process Mashup and Web site customization. Seven dimensions were defined in order to analyze the selected Mashup makers, including the support for community features dimension which represents a particular interest in our context. Indeed, community members provide elements that can be reused by other members, create examples, and help each other. Some features were identified and classified as: (i) Mashups sharing, (ii) collaborative classification, notation or marking (iii) exchanges and discussion forums. Accordingly, the proposed analysis can be applied to the social network (of friends) case. For instance, this analysis could be useful for the specification of a "Mashup maker" in order to optimize end-users support features. We may notice that the authors have highlighted

the need to introduce the social networking features at the heart of the Mashup creation process.

2.6.4 End-user support

After reviewing the general properties of Mashup creation environment, the next section highlights, based on existing studies, the growing need of support features to the end-users in order to help them composing services. Table 2.1 summaries the main features provided by the Mashup creation environments cited above. Those features could potentially be used as support for end-users at several levels. Moreover, in order to facilitate service composition for end-users, current Mashup editors provide as abstraction layer that hides the technical specifications and simplifies them for the users. For example: providing a web service with abstract description in a form of input/output black box, and a composed service in a form of a graphic flow or sequence of services. Most Mashup editors also allow the reuse of created composed services as building blocks to compose other services. In addition, to help end-users to compose services, Mashup creation environments provide learning materials as videos, tutorials, and forums for assistance. Learning by example is also an approach that allows new users to reuse and edit Mashups that have been created by others. We categorize the features listed above as indirect support for users in the process of composition.

To provide direct assistance to end-users, most Mashup editors tend to ease the end-user intervention in the process of composition. This intervention can take place at three levels:

- Pre-composition support: by facilitating the selection of services by features that are either service categorization, textual or contextual selection.
- Post-composition support: by providing the ability to tag or rate basic services. This information is used later on by recommendation systems (collaborative filtering or content-based) at the pre-composition phase in order to allow the automatic selection of services that fit with users' preferences.
- In-composition support (at the services scheduling phase): For this case, no direct features have been identified in the current Mashup editors that help end-users in selecting services when he is creating a composite service (connecting services).

Nevertheless , several studies have shown the potential of exploiting the interactions of users with services as basis for supporting features to the end-user. In the same direction, through a use-case approach, Floyd et al. [Floyd 2007] highlight the APIs proliferation on the Web in parallel with the number of creative Web users. The study shows the benefits of the collaboration between end-users and developers that combines the innovation and creativity of end-users with the expertise of developers. Automating this collaboration is the important challenge we are looking to tackle. In that regard, an interesting study [Jones 2009] describes the

Table 2.1: Summary of the most relevant features offered by some Mashups environments

		MS. Popfly	Yahoo! Pipes	IBM Mashup Center	Intel Mashmaker
Abstraction Level	*Reuse of complete Mashup*	Y	Y	Y	Y
	Visual data-flow languages	Y	Y	Y	N
Learning support		Y	Y	Y	Y
Sharing Mashups		Y	Y	Y	Y
Community features	*Tagging*	Y	Y	Y	Y
	Rating	Y	Y	Y	N
Discovery and selection	*Text-Based Search*	Y	Y	Y	N
	Categorization of services	Y	Y	Y	N
	Context	Y	N	N	N

interactions of Yahoo! Pipes' users. This can be used to extract social structures based on an analysis of user interactions. Furthermore, those users interact with services through the Mashups they create. Soriano et al. [Soriano 2008] emphasizes the growing importance of the user-service relationship in a Service Oriented Architecture for composing services. In fact, the authors introduce EZWeb, an environment for sharing Mashups between colleagues, as a basis for co-production in an enterprise context. In addition, [Maamar 2009] emphasizes the phenomenon of what they call "social interaction" between services. In fact, the aspects of trust and reliability between services may impact the service selection for composition. Yu and Woodard [Yu 2009b] propose a very interesting view of the ecosystem of Mashups. This study, on the Programmableweb API repository[11], has truly shown that utilization of services follows a long-tail effect (power-law distribution), one of the major and interesting properties in social networks [Wasserman 1994]. We believe that service recommendation is a solution to spread expertise between users to enable them composing services.

2.7 Discussion

We presented in this chapter a literature review of web services, service composition and end-user oriented composition environments (Mashup editors). In fact, we have reviewed the concepts of service, SOA and Web services composition and key

[11] http://programmableweb.com

concepts that they emerge from. We classified the different approaches to service composition either from system or end-user perspectives. This review has allowed us to highlight the main issues in the service composition research area and particularly focus on the exact issues we have identified as central in our work.

Furthermore, we have pointed out the concepts of SOC and composition of services, which were originally developed for enterprise application integration, and that have recently evolved to end-users, typically Web users. Those end-users are characterized by limited technical and programming skills, but are nevertheless producing Web content. In fact, in the Web 2.0 context, one of the interesting property of end-users is their ability to produce or participate in producing content. The Web 2.0 has brought a set of different technologies dedicated for end-users (even in an enterprise context) so it becomes very easy for such users to publish or annotate resources (User Generated Content (UGC)). Furthermore, those end-users are tying new relationships based on interests to the generated content, and stay in touch with their social relatives through online social networks and collaborative environments. Consequently, the composition of services should nowadays to be driven by end-user needs, as it is encouraged by online environments of sharing and social interactions through the Internet.

Mashup editors have emerged as an answer to this evolution in order to overcome the technical complexity that the end-users were facing and to ease the composition process for them. Actually, through this mashup concept, existing works have provided (i) abstraction features such as visual workflow language, and (ii) community features such as rating and tagging, and (iii) service selection facilities such as text-based search. Even if those features are absolutly necessary, we hardly believe that they are sufficient. In particular, during the composition process itself, and as we have pointed out, existing features do not currently provide any direct support to end-users. In fact, The users have to manually select and connect all services in order to compose them according to specific requirements and the composition logic. This phase of the composition represents a relatively painful phase of the process due to the lack of support it is characterized with. This is the key issue that we are addressing in this thesis, the gap we are proposing to fill, and to which we are providing solutions in order to offer more and more support to end-users, and therefore contributing to the semi-automatic services composition approach. Some recent interesting works, that are being explored including ours, promote continuously assistance of the end-user when he is composing services. Next chapters introduce the concept of dynamic service recommendation original approach that is the proposed answer to the key issue mentioned above. It represents the main contributions of this work that is based on the introduction of the social dimension within the composition process.

Chapter 3

Service Dynamic Recommendation For End-User Support

3.1 Introduction

The main idea, as introduced in Chapter 1, consists of providing a framework for creating a Mashup driven composite service. This framework must provide basic features essential to compose a service, namely (i) basic services directory exposed through the framework (whether services are deployed locally or remotely); (ii) a user graphical interface to express the logic of the Mashup (composite service schema); and (iii) a platform to translate the introduced logic into an executable script representing the composed service which will then be deployed in a runtime environment. The framework, its basic features, and their implementation are described in Chapter 5. In addition to basic functionalities, we are committed to fully assist the end user in the task of composition. The main reason behind this commitment is the lack of expertise of the end user, and the growing number of services that are exposed. Our contribution consists of a dynamic recommendation service feature to assist the user during the task of composition. This chapter outlines the concepts and basic models of this proposed contribution. The model of completion followed by the proposed recommendation feature can be realized through two projections or instantiations. The first one, a step-by-step completion, is described and evaluated in this chapter, whereas the second one, a full Mashup completion, is described in Chapter 4.

Even if the dynamic recommendation is not a new concept by itself, the idea of dynamically recommending services in the context of Mashup creation environment is a novel feature. However, its novelty does not exclude the fact that it still needs to comply with a couple of requirements derived from Mashup creation environment and recommender systems requirements. Current related works, such as [Ennals 2007a] and [Greenshpan 2009], do highlight minimum set of these requirements that are summarized in two points: (i) ensure successful user experience in a user-centric interactive framework by optimizing the system response time, and (ii) improve the quality of the recommendation to meet the end user's actual needs.

Before presenting the adopted recommendation algorithm and its variants based on what we named the recommendation confidence metric in Section 3.8, several entities and concepts have to be introduced and presented in detail. Consequently,

Section 3.2 briefly reviews the recommender system related work. Section 3.3 highlights both the emergence and importance of the social dimension in the Web 2.0 context. It simultaneously introduces the idea of social networks based dynamic recommendation that preludes and justifies the path that led to the design of the algorithm. Section 3.4 details the construction of the implicit social graph which constitutes the foundation model of the algorithm. Section 3.6 explains the considerations (assumptions and simplifications) adopted when the dynamic service algorithm is instantiated within the Social Composer (SoCo) framework.

3.2 Brief overview of recommendation systems

The field of recommender systems, with its multiple applications, is a well established research area. Generally, all recommender systems are a variant or hybrid of the two conventional approaches: (i)collaborative filtering recommender, and (ii) content-based recommender.

In particular, applications of the recommendation for the selection of services in a semi-automatic service composition context are a variant of previously mentioned approaches. However, as we detailed in Section 2.5.2.3, when considering service selection recommender systems from the end user point of view, existing applications take two approaches: user-centric and community-centric. In fact some systems focus on a single end-user by tracking his own interests and/or preferences and leveraging it for his needs. An alternative way is to consider the user as part of a community; As this will be described in the following.

3.3 Providing support to end-user

The main features of the Social Composer (Web service composition framework detailed in Chapter 5) are intended to assist end-users in the composition process by providing support when selecting a service. This support is based on the current configuration of composition and user's social neighbors composition behavior. To help the user in the composition process, *SoCo* includes two main steps represented by: (i) a social knowledge extraction and modeling component and (ii) a recommendation manager as depicted in Figure 3.1

Figure 3.1 illustrates the general architecture of the framework. The first step consists in extracting and modeling the existing "knowledge" in a social network. Knowledge here stands for information which can be used in the context of services composition, e.g. which person uses which service in which composition and at which rate. This knowledge is captured in order to define and construct a set of rules and actions. These rules and actions describe in fact the composition behaviors that could be used to improve and customize a incomplete part of a particular composition schema in relation to a particular user. This operation is the role of the second component (recommendation manager).

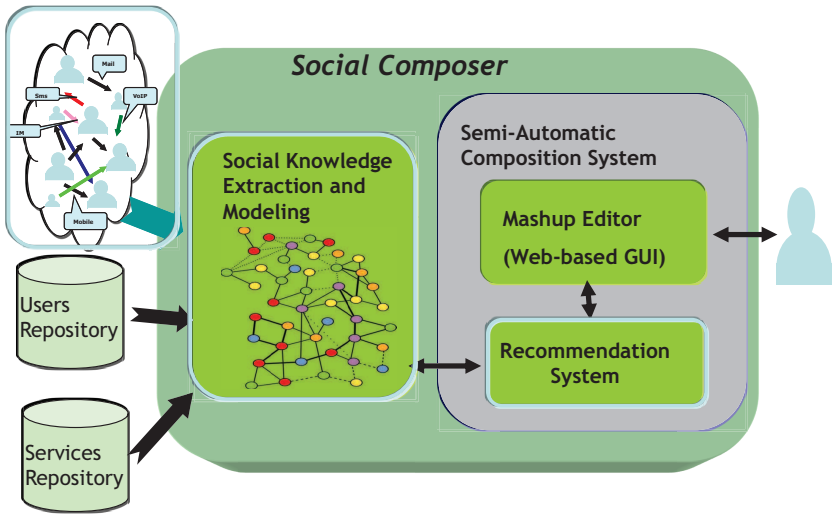

Figure 3.1: General architecture of *SoCo* framework for social network integration in the composition process

The main purpose of our framework is to support user in the process of composition. This consists of the dynamic recommendation of services as the user composes new services by linking existing ones. This feature must meet a certain number of requirements to ensure an efficient interactive user experience. Regarding interactive application requirement [Shneiderman 1984], end-users prefer a response time of less than one second. This implies the following constraint: the recommendation engine used in *SoCo* has to provide suggestions (recommendations) within the allotted timeframe of one second.

3.3.1 Uses cases

We discuss in this section some illustrative use-cases which help expose the different dimensions of the proposed framework. The first type of use cases groups the case of Mashup creation by typical Web end-users who seek to automate some tasks. The second type of use cases groups the cases of Mashup creation in a business context (enterprise environment) where service composition is seen as an easy-to-use workflow creation.

3.3.1.1 Web end-user typical use cases

The "Cinema fan" use case Consider Alice, a young fan of movies and cinema. To plan her cinema outings, Alice generally performs several tasks before selecting the movie she is going to watch. For instance, she needs to consult many Web pages to get information about the quality of a particular movie, ask her friends, read the different comments about the movies to find out whether the movie is related to her preferences, or locate a good cinema where she can watch the movie. This is

generally time and effort consuming. Alice is novice in services composition even if she already uses Mashup editors; but she has a good social network that could help her in the operation of selecting and combining services. However, her friends are generally not available to help her in building her service (Mashup application). The *SoCo* system can verily be of a great interest for Alice in this situation.

Thus, Alice decides to create a Mashup that will automate this kind of activity using the *SoCo* in order to take advantage of the Mashup created by her social relatives. These Mashups could gives some hints of the best practices when composing services and are effectively used to provide some dynamic suggestions to Alice. Alice first searches for relevant services related to cinema by querying the services repository. She finds out that *FilmAdviser*, a service that returns movies suggestions on the basis of a set of users preferences, is available for use. She selects the service and drops it into the *SoCo* editing area. Simultaneously, the recommendation system receives a query for finding services that can possibly come after the currently selected service (i.e., *FilmAdviser*).

Using the different information the system has about, for instance, the social relations of Alice, the usage rate of the different services by that social network, etc. *SoCo* recommends *Cine-Map-Calendar*, a service which is generally used in the social network just after *FilmAdviser*. *Cine-Calendar* is a service which considers a movie as an input and gives an output of the movie and its different planning during the Week. After that, *SoCo* proposes another service, *Cine-Map*, which, given the title of a movie and a specific city, displays on a map the different cinemas where that movie is shown in that city. The system also recommends the use of a service that allows Alice to finally buy cinema tickets. Alice prefers to buy the tickets when she gets to the cinema and thus she didn't use that service.

To complete the chain, *SoCo* recommends to Alice the use of the *RDVSchedular* service which allows a group of people to select available time slots. Alice accepts the suggestion of the system but will additionally need a reminder for that event. This is a typical situation when planning and event and is frequent in the social network of Alice. Thus, *SoCo* recommends *SMSReminder* as a service that sends an SMS reminder to participants of an event. Satisfied, Alice decides to stop at this level and initiates the deployment the resulting composed service.

The sequence composition use-case Consider three users Alice, Bob, and Carol who compose services using their favorite Mashup creation environment. Alice and Bob, who are more or less familiar to composing services, create various Mashups. In order to find the definition of a word in English, translate it into French and then email it, Alice creates a Mashup composed as follows: $Dictionary \rightarrow Translator \rightarrow Email$. Moreover, in order to find the weather forecast, translate it and receive it by SMS, Alice creates a new Mashup as follows: $Weather \rightarrow Translator \rightarrow SMS$. On his side, Bob would like to create a Mashup that finds the weather description from his location, send it on his blog, and then post a tiny URL of his blog on his Twitter profile. Thus, he creates the following

Mashups: $Mylocation \rightarrow Weather \rightarrow BlogPost \rightarrow tinyURL \rightarrow PostTwitter$.

Carol would like now to create a new Mashup based on a weather forecast service. Once she selects the weather service, she gets a list of completions. Each completion is a sequence (one or more linearly connected services) which when connected to that weather service forms a complete Mashup. These completions are based on other users' usage and are listed in the following: (i) $\rightarrow Translator$, (ii) $\rightarrow Translator \rightarrow Email$, (iii) $\rightarrow BlogPost$, (iv) $\rightarrow BlogPost \rightarrow tinyURL$, and (v) $\rightarrow BlogPost \rightarrow tinyURL \rightarrow PostTwitter$. However, since she doesn't really have experience with these services, Carol wishes that the completion list would preferentially rank completions related to her social relatives. If we suppose that Carol is socially close to Alice (shares more interest with her), it is more likely that the system would prefer a recommendation originating from Alice rather than from the whole community. Suppose that Carol selected the completion (i) $\rightarrow Translator$. After her selection, the completion list is updated dynamically. It will offer, following to $Weather \rightarrow Translator$ two completions: $\rightarrow sendSMS$ or $\rightarrow Email$, and so on until Carol chooses to terminate the Mashup.

3.3.1.2 Business context type use cases

Conference attending workflow use case The enterprise is a goldmine for social networking applications. Even if this environment appears well organized and hierarchical, it hides more complex social interactions and organizations. An enterprise environment is also interesting regarding privacy issues, as it is controlled and managed and knowing that interactions (typically e-mails exchange) between employees are supposedly dealing with company-related issues. It is therefore relevant to analyze the interactions and define the profiles of employees and the links between them. The profiles of employees represent their expertise areas and skills, when the links between them represent the formal and informal information flows and exchanges context.

We present hereafter a use case that occurs frequently in companies especially for new procedures for which no internal processes are defined and may involve several departments of the company. Participating in an external conference or workshop is a relevant example. Bob is an employee and would like to attend a conference or workshop. Bob decides to create a Mashup with *SoCo* framework to manage this special need and shares his experience with his colleagues.

On the basis of the process already defined in the company by other colleagues, *SoCo* assists Bob in the creation of his Mashup by proposing and recommending services. First, Bob selects the service that provided information about the conference (date, location, prices, ...). *Soco* proposes two services: *ManagerValidation* and *TravelBooking* with a stronger recommendation for *ManagerValidation*. Bob decides to choose *ManagerValidation*. As a result, *SoCo* recommends three services: *VisaApplication*, *TravelBooking*, and *HotelBooking*. Bob chooses *TravelBooking* then *HotelBooking*. The system continues to propose other services as a service that allows employees claim their travel expenses, but Bob decides to built and share this

Mashup on the company Web 2.0 portal.

Meeting report composed service use case In a business context, Bob is an employee and often makes phone calls to his partners for meeting. At each end of a meeting, he has to remember the various discussed topics and draft a report about those conducted phone calls. Bob decides to use *SoCo* in order to automate this task. He selects the first service *voice call*. At this level, *SoCo* recommendation system processes or has already processed compositions already made by Bob's colleagues and releases a recommendation list that proposes to use other services following the current *voice call* service. Thus, SoCo proposes to use a restricted list of services *redirection service* (redirect the call in case of failure), *SensSMS* (send an SMS when the call is done), *Calendar updating* (to report on the agenda that the call has been done), *speech-to-text* followed by *eMail* (in order to transcribe the call in text format then emailed), or *speech-to-text* followed by *SendSMS* (in order to transcribe the call in text format and then sent it by SMS).

3.4 Social-based approach

A social network can not be viewed as a community (see community approach in the previous section). The major difference is that community describes a gathering of individuals around "one" common topic of interest, generating communities specializing in particular areas (what justifies this approach). In contrast, social networks describe individual networks constructed on the basis of specific interests or friendship for each pair of individual relationships in the network. In other words, the knowledge defined in a social network can not be processed as a community knowledge since a social network necessarily includes one or many communities. Table 3.1 summarizes the most relevant differences between the two approaches. Therefore, a different approach needs to be applied for social network based recommendation system.

As we are clearly in a Web 2.0 environment, we consider that the user needs to be part of in the process not only as a separate entity or a group of people but as an interlinked entity with other entities following a relation translating, e.g., common interests and friendship. This will lead to fine grained, more precise and personalized support for users. Introducing this dimension, i.e., the social dimension, incorporates the interesting observation that a user is more interested in the recommendations that come from members of his social networks (family, friends, colleagues, etc.) or from people with whom he shares common interests. Our full completion strategy takes into account users' specificities that are reflected in their different service composition behaviors. Table 3.2 enumerates some interactions between the different managed entities in a Mashup platform which may indicate a social behavior.

We consider the following types of interactions between entities in the system: the *Composition* and *Diffusion* as for the interactions between users and services, *Follow* interactions between services, and *Friends* and *Communities* interactions

Table 3.1: Community Vs social networks services composition

Community	**Social Networks**
describes a gathering of individuals around a common topic of interest	describes friends networks constructed on the basis of specific interests for each relationship in the network
specific domain knowledge and Community generated knowledge	The extracted knowledge can not be processed as community knowledge
Knowledge processing methods already exist in the literature (CommonKADS and OilEd)	Existing Social networks analysis methods are not appropriate and need to be adapted
involved either in the pre-composition process at the discovery and selection levels, or at post-composition process by annotating, ranking, and rating service	In addition to be involved in pre and post-composition, social knowledge in directly involved in the composition process itself.
Simple recommendation system	Complex recommendation system
High granularity level of users consideration (i.e., community level)	Detailed level of granularity for user consideration (i.e., individual level)

Table 3.2: Example of possible interactions that could be extracted from a Mashup creation environment

Users and services interactions	Service interactions	User interactions
- Creation	- Follow	- Friends
- Discovery	- Competition	- Communities
- Composition	- Replacement	- Influence
- Diffusion	- Collaboration	- Mentor
- Annotation	- etc.	- Hierarchy
- etc.		- etc.

between users. This is motivated by the well known observation from Web 2.0: 90% of users consult content (i.e., equivalent of using an existing composition in our context), 9% comment on the content (make a recommendation in our context), and 1% create new content (creation of new services in our context) [Nielsen 2006]. Thus, the interactions that are our focus are those that impact the largest audience: the 90% of users who interact with existing service compositions. We consider interactions that involve end-users as social interactions, and part of the social dimension.

3.5 A new approach to service recommendation

In Web 2.0, people can create, use, and share services in communities and social networks. These services can be simple Web services or more sophisticated services

as Mashups. The question we address is: *how can social interactions be leveraged to enable and facilitate composite services creation for end users?*

Regarding this problem statement and the related work, we propose a general approach for dynamic service recommendation based on the transformation of both *user* → *users* and *user* → *services* interactions into social networks on top of which statistical processes may be applied to fire recommendations for assisting the user in constructing the Mashup (a composition of services). In other words, we leverage knowledge retrieved from social networks, applied to a Mashup and Web services composition. Before detailing our proposal, let's clearly define the two notions of composition pattern and social network that are used frequently:

Définition 3 ***[Composition pattern]*** *A composition pattern is the repetitive schema or a part of schema that describes a composition of services representing the logic of a Mashup. In general, this schema is represented as a workflow of basic services. Therefore, every part of that workflow represents a composition pattern.*

It comes out from the definition that given a Mashup schema created by a given user, one can deduce one or many composition patterns related to that user. Actually, those composition patterns tell us about the behavior of that user, which consequently allows us to answer the question: "how does each user behave when composing services?". As described hereafter a pattern may take different forms depending on how a composition schema is modeled. For instance, the pattern could be either a simple sequence of two or many services.

To clarify the meaning of a social network in our approach, we propose the following definition.

Définition 4 ***[Social network]*** *A social network is a graph representation of all interactions that occur between people and services in a composition environment.*

In our context, this structure may be directly extracted or inferred (deduced) from common interests between the users of the composition platform. In other words, the social network we consider at this stage, may be either (i) explicitly declared by users or (ii) an implicit structure inferred from the common interests of users. We have compared these two variants and detailed each one in Section 3.5.1, which results in a focus on the implicit case as sophisticated approach for modeling interaction between end-users.

By construction, this social network includes a profile for each user containing information that describes his/her special interests and preferences, and the history of his/her interactions with the system (i.e., dynamic profiles). Typically, these include statistics on services utilization (consumption and composition). This information enables us to learn about the expertise of a given person in a particular area, and thus the relevance of services used by that person. The social network includes as well the description of links that define the social graph itself. These links are used to calculate the social proximity between two users according to a particular context. This information allows us to calculate the service recommendation confidence between two individuals based on specific joint interests. To leverage

this information, it is therefore very important to extract, analyze and model the information contained in a social network.

Information regarding user interactions are used by the recommendation system to support users during the creation of new Mashups. It dynamically suggests services according to the current status of the services composition task, i.e., which pattern (service or sequence of services) is likely to come after the currently introduced one? Thus, it intervenes in the services selection process. This recommendation logic considers different parameters as the user's position within his/her social network, and usage information of services by the social neighbors.

More concretely, during the creation of a composed service through the *SoCo* service creation environment, a user generally is undecided about the selection of a service going to follow an already introduced Mashup part in the composition diagram. In this situation, the recommendation system will propose a list of services ranked on the basis of information provided from the social network analysis. Thus, the relevance of a service recommendation is proportional to its usage history with respect to the user's social proximity. This means that the more a service is used in this social network the more the recommendation becomes significant. Similarly, when users are close in the social network to the current user, services they use are more important to be recommended (according to the current need). Moreover, when users who use certain services are experts, their choices are supposed to be more relevant which implies better recommendations as well.

3.5.1 Social graph variants

In the context of a Web 2.0 environment, many types of social networks (meaning social graph) is emerging taking different forms. Generally, the most common type of social network is the explicitly dedicated social networking web sites like Facebook or Twitter where the end-user solely and actively builds his/her relationships. In this case, we talk about explicit social networks. Verily, end-users decide themselves whether they are interested or not in connecting to other users, which leads to a user-user explicit social graph. Moreover, users are able to declare their interests in terms of shared content as well. In the particular case of service composition, end users can declare their interest to a given service (which service they usually like to use in a composition schema). We talk in this latter case about user-service explicit social graph, from which we can perceive shared interest between users. As explained in using collaborative filtering recommender approach, those shared common interests actually represent potential hidden relationships that lead to what we call an implicit social graph. We describes in [Maaradji 2010b] how this explicit form of social network can be leveraged in a dynamic recommendation system.
By contrast to explicit user-service interactions, implicit user-service interactions refers to users' usage of services. In fact, users consume services as is (for instance a calling service or geographic mapping service) or use it to compose other services. This type of user-service activity actually represents user behavior regarding services and it can therefore be considered as a set of implicit interactions. We believe that

user-service interactions can be leveraged to unveil the non-explicit user-user hidden relationships that we define as the implicit social network. The next section focuses on describing the concept of implicit social graph.

3.6 Assumptions

We have seen in section 2, many tools are available for this operation. Besides, we have already argued that the visual workflow concept (graphical workflow) is one of the most relevant and appropriate tool to express the composition pattern in order to satisfy the abstraction requirements. This workflow consists a set of services connected together to express the logic of composition. Depending on how services are connected, the graphical workflow could more or less have a complex form. Basically, there are two forms that a graphical workflow may have.Firstly, the sequence of services, also known as pipeline, is the simplest way to define a composition logic. Secondly, a more complex form is possible particularly to express parallel logic, or even conditional and loop logic.

However, given the current state of the art, we are considering only the sequences of services as a possible composition pattern. We have already shown in Section 2 that, either in actual realizations [Erlewine , Shiaa 2008] or theoretical modeling [Greenshpan 2009], a Mashup is often considered as a sequence (pipeline) of services. This is not only due for the need of theoretical simplification but also because end-users tend to describe a complex process as in sequence of simple processes [Koop 2008]. More over a more complex composition could be represented more easily by combining sequences fo services.

Given this facts, completing a Mashup becomes nothing but predicting remaining parts of a given partial sequence introduced by the user. By "remaining part" here, we mean a subsequence of services which when connected at the end of the partial sequence introduced by the user will result in complete Mashup. In a summary, we lay down assumptions that a Mashup (a composed service) is a linear sequence of services (the composition pattern), and Mashup completion is suggesting the forward remaining subsequence of a given subsequence.

3.7 Implicit social graph construction

An implicit social relation is inferred according to the activities of the different users, e.g. when two users use the same composition pattern. In this case, we end-up with a graph linking the users according to their interests defined by their composition activities.

We firstly consider $users \longrightarrow pattern$ interactions as a bipartite graph [Guillaume 2004]. This graph represents the usage rate users have on services they use. It should be noticed here that the bipartite graph is solely based on the usage frequency of services by the users in composition schemes and without taking into account services succession in those schemes. Figure 3.2 illustrates a bipartite graph of services and

users. The links represent how frequently a user u_i uses a service p_j in all the compositions he created. We denote this usage frequency by $f(u_i, p_j)$.

In order to interpret and leverage social interactions in a Mashup environment, one needs to process the *users* $\rightarrow$ *patterns* interactions into a social graph between users. In fact, we have two sets of distinct inputs: users and services sets. There may certainly exist several techniques for doing so (collaborative filtering method). We describe in the following our approach which uses three levels of information extracted from this representation: (i) local information, (ii) semi-global information, and (iii) global information. These three levels are combined at the end to fire recommendations. This process guarantees to take into account all the types of interactions that could exists between users and services.

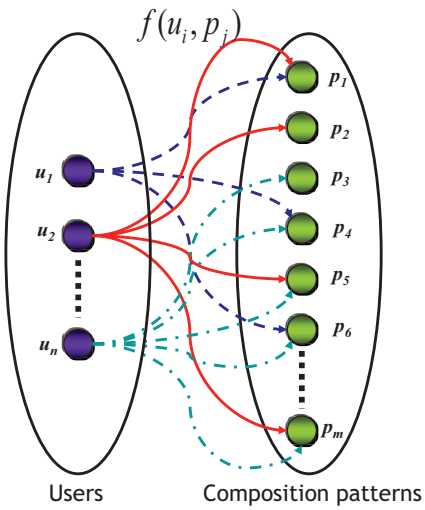

Figure 3.2: Illustration of a bipartite graph between users and patterns

3.7.1 Local information

The *local information* considers only the interaction between a specific user and a specific composition pattern. This means that we consider a user and a pattern independently of the other users and services of the system. This information tells us whether a specific user is confident (i.e., expertise indicator) using this pattern among other patterns. The more a given user is confident about his usage of a given pattern, the more the recommendation of this pattern matters.

To materialize this idea, we define this information in a quantity called *Activity* defined in Equation 3.1 where M is the total of pattern a user exploited in his different compositions.

$$Act(u_i, p_j) = \frac{f(u_i, p_j)}{\sum_{k=1}^{M} f(u_i, p_k)} \tag{3.1}$$

3.7.2 Semi-global information

In this level of semi-global information, we consider the interest a user may have in other users regarding a given pattern. Thus, for a given user u_i we calculate how much the service s_j recommended by the user u_l matters to him. This is called *Special Interest* (SI) and is calculated using Equation 3.2.

$$SI(u_i, u_l, p_j) = \frac{f(u_l, p_j)}{f(u_i, p_j)} \tag{3.2}$$

This results into a strata of social graph, as shown in figure 3.3, where each stratum represents a transition specific interest social graph.

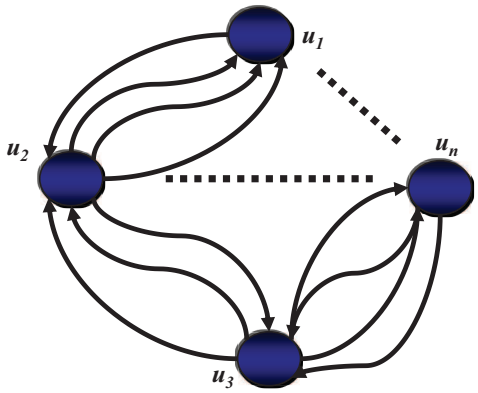

Figure 3.3: Illustration of the output after applying a semi-global information level

3.7.3 Global information

In order to have as precise transformation as possible that we can tune later, we add another level of information in the transformation process. The global information captures whether a couple of users have common general interest or not. At this stage of our study, and for simplification reasons, we consider that the general interest of a couple of users is equal to the sum of their specific interests, thus building the implicit graph as illustrated in Equation 3.3.

$$IG(u_i, u_l) = \sum_{k=1}^{M} SI(u_i, u_l, p_k) \tag{3.3}$$

The output of this step is a graph aggregating all the specific interests graphs obtained previously as shown in Figure 3.4.

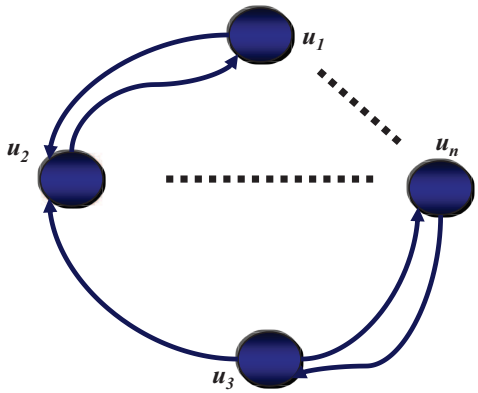

Figure 3.4: Illustration of the output after applying a semi-global information level

3.7.4 Graph reduction (Top-k links)

For optimization goal, and based on observations that each user may have few edges that are much greater than the rest in the implicit graph, we decided to reduce those corresponding edges following a *Top-K* strategy. The resulting graph is called the *Top-K* implicit graph. This post-processing step cleans up the obtained outputs. The recommendation strategy will be built on top of the outcome of this step.

In fact, we use a very simple variant of Fagin's Top-k algorithm [Ilyas 2008]. The algorithm separately selects the highest weighted out-edges (top 5) for each node. The Top-k implicit graph is obtained by aggregating the remaining edges. We chose this approach to avoid dropping nodes from the graph and to guarantee that each node (user) is not excluded from the recommendation system. However, a threshold-based Top-k algorithm is more relevant to keep only significant edges (interests) even if some nodes in the graph are lost [Leskovec 2007]. Nevertheless, we believe that ultimately a more elaborated approach need to be designed in order to avoid loosing the so coveted social network properties (small word, etc.). In fact, [Ilyas 2008] shows that selective node dropping in a social graph could induce loosing social network properties. From this point on onwards, each reference to implicit graph (IG) refers to a Top-k implicit graph.

3.7.5 From user-transition interactions to implicit social graph: Application on transition composition pattern

As we have introduced it before, several patterns of composition may exist, among them, the transition between two services. Indeed, when we reduced the expression of a composite service into an ordered sequence of many services, the basic element of this sequence becomes the transition between two services (sequence of two services). In the following, we applied the previous model to build the implicit graph in the case of a simple transition to define the relationship between users, and then use it

in a context of dynamic recommendation system.

Following the described model, we consider $users \rightarrow Transition$ interactions as a bipartite graph which represents the usage rate users have on transitions they use. To transform the bipartite graph into a social graph for successor recommendation we rely on the three main steps described before that we summarize in the following.

The *local information* considers only the interaction between a specific user and a specific transition. This information tells us whether a specific user is confident (i.e., expertise indicator) using this transition among others. The more a given user is confident about his usage of a given transition, the more the recommendation of this service matters. In this regard *Activity* in updated in Equation 3.4 where M is the total number of services a user exploited in his different compositions.

$$Act_{trans}(u_l, (s_j \rightarrow s_k)) = \frac{f(u_l, (s_j \rightarrow s_k))}{\sum_{g=1}^{M} \sum_{h=1}^{M} f(u_l, (s_g \rightarrow s_h))} \tag{3.4}$$

In this level of semi-global information, we consider the interest a user may have in other users regarding a given transition. Thus, for a given user u_i we calculate how much the transition t_j recommended by the user u_l matters. This is defined and shown in Equation 3.5.

$$SI_{trans}(u_i, u_l, (s_j \rightarrow s_k)) = \frac{f(u_l, (s_j \rightarrow s_k))}{f(u_i, (s_j \rightarrow s_k))} \tag{3.5}$$

As defined in the general model, this phase is about integrating all the special interest graphs within a single graph that expresses the overall interest between individuals. The aggregate function chosen here is a simple sum of individual interests as illustrated in Equation 3.6. The output of this step is a graph aggregating all the specific interests graphs obtained previously.

$$IG_{trans}(u_i, u_l) = \sum_{g=1}^{M} \sum_{h=1}^{M} SI_{succ}(u_i, u_l, (s_g \rightarrow s_h)) \tag{3.6}$$

3.8 The Completion process

This section aims at presenting the completion process which is the visible part the recommendation system from the end-user point of view. As we explained earlier, the completion consists of retrieving the most likely termination of the currently introduced composition pattern, and recommending the most relevant with respect to user's interest. To perform this recommendation, we need first to retrieve all possible terminations of the given part of a pattern. In other terms, when a user introduces a service or a sequence of services, the first step is to consider this input as a request to figure out all possible completions, named here completion list (or recommendation list). The second step is to sort this completion list according to many parameters (including to the social proximity between users). At this regard we define a global metric that we named the *Recommendation Confidence*.

Let's consider the completion operation is the step-by-step approach which define a transition of two services as the basic pattern of a composition. Computing the completion becomes simply the completion (termination) of the last transition (if possible). In other terms, we need to consider the last service introduced by the user as the first part of a transition, and then retrieve the second part of the transition (constituted by both services) if it exists. Then, each selected transition (a candidate termination) is sorted in the recommendation list based on its calculated *Recommendation Confidence* metric.

3.8.1 Completion pattern Recommendation Confidence (RC)

Coming back to the general model, once the bipartite graph is transformed to a social graph thanks to the three previously described steps, we proceed to recommendation calculation to suggest a coming completion pattern according the completion list. Thus, considering the intrinsic user's usages frequency (local information), the specific interests between two users (semi-global information), and the implicit graph which expresses the global interest between users, we define the recommendation confidence of a given pattern p_k with respect to a current pattern p_j for the user u_i as follows:

$$RC_{imp}(u_i, p_j, p_k) = \sum_{l=1}^{N} SI(u_i, u_l, p_k) \times Act(u_l, p_k) \times IG(u_i, u_l) \tag{3.7}$$

In the case where the composition pattern is simple such as simple transition, we define the recommendation confidence of a given service s_k to follow a current service s_j for the user u_i as follows:

$$RC_{trans}(u_i, s_j, s_k) = \sum_{l=1}^{N} SI_{trans}(u_i, u_l, (s_j \rightarrow s_k)) \times Act_{trans}(u_l, (s_j \rightarrow s_k)) \times IG_{trans}(u_i, u_l) \tag{3.8}$$

3.8.2 Recommendation algorithm design

Algorithm 1 summarizes the recommendation process. Given a configuration (u_i, p_j) where u_i represents the current user and p_j the introduced pattern, the algorithm returns Rec_{List}, a sorted list of recommended patterns that are socially relevant to complement p_j for user u_i. It should be noted that since the output of the algorithm is a list and the selection follows a $Top-K$ principle (K patterns), the values are not necessarily inside the interval $[0, 1]$ even if this could be easily integrated by, e.g., maintaining the maximum value of the recommendation to normalize the output.

3.8.3 Basic enhancements

In recommender systems, the problem of new users or items (here services) continues to generate a significant research output and contributions [Adomavicius 2005].

Algorithm 1 The Recommendation algorithm for semi-automatic services composition

input $u_i \in U$, $s_j \in S$
for each s_k where $serv_i \rightarrow serv_k$ exists **do**
  $RC_K = 0$
  for each u_l, where u_l is neighbor of u_i in a specific graph of s_k **do**
    $RC_K = RC_K + Act(u_k, s_k) \times SI(u_i, u_k, s_k) \times IG(u_k, s_k)$
  Add (s_k,RC_k) in Rec_{List}
Sort Rec_{List} in descending order of RC_k
output $Rec_{List} = \{(s_k, RC_K)\}$

In fact, various and diverse methods have been proposed to overcome the lack of knowledge about newcomers into the system. Those methods are mainly based on users or services popularity in the system (community-oriented approach mentioned in section 2.5.2.3) or diversification methods. In our system, it is quite possible to incorporate such methods. However, a difference should be notied between a newcomer user and newcomer service.

3.8.3.1 Combination of social and community approaches for newcomer users

By definition, a newcomer user has not previously used any service and therefore the system has no usable traces. In this case, it is possible to rely on current practices in the community (all other users) to recommend a composition behavior to this newcomer user in order to facilitate him interaction with the system, hence allowing the collection of a maximum of interactions to build this user implicit social graph. More concretely, to calculate the recommendation confidence of a candidate service to this specific user, it is possible to use a weighted balance between the quantity defined (Form RC), and popular services in the community:

$$RC(u_i, s_j, s_k) = \alpha RC_{imp}(u_i, s_j, s_k) + \beta Popularity(s_j, s_k) \tag{3.9}$$

Where $Popularity(s_j, s_k)$ refers to how many times s_k follows s_j over all existing succesions.

The weighting parameters (α, β) gradually evolve from (0,1) to (1,0) to as the amount of knowledge on a user increases. Moreover, the concept of "a posteriori interest" complements the previous approach in addressing newcomers issue. In fact, this concept allows the establishment of a temporary link between a given user and a service candidate that was never used by that user. This is a practical way to open up the system to new users. At the last resort for our system, the explicit user self-declared graph can also boost his starting phase in the system.

3.8.4 Service Post-interest for new comer services

Newly introduced services in the platform are never going to be recommended by the system under its basic implementation. In other words, if a service has never been used before by any user, it will never be recommended. This represents a major issue for the recommendation strategy. Diversification methods are the primary solution for this issue: They aim at making such services "visible" in the system. A random selection among this population could be sufficient. In the service composition context, we are proposing to use non-functional properties of services to select candidates for diversification. In a more detailed description, it is possible to match potential service successions from interfaces description of each newcomer service (e.g., input/output description). This will identify the new services as potential candidates for diversified recommendation.

Furthermore, By construction, our recommendation system based on modeling of social graphs could enhance diversification approach [Yu 2009a]. Indeed, as mentioned before, the purpose of diversification is to make visible the new introduced services in the system.

One way of realizing the diversification is to recommend the newly introduced services primarily for key individuals in the social graph. Because these users have an important influence in the graph, the recommendation of new services to them will make these services more visible as illustrated in what follows:

- The properties of the social graph allow the identification of the key users (also called key players) in the social network
- Recommending the new services to those key players will result in having these services overweighted in the recommendation system

Furthermore, the recommendation of the new services to these key players has another benefit. Because key users are also identified by the system as experts regarding a set of services, recommending newly introduced services to them will help improve the quality of services. They implicitly play the role of good trusted filters of the system.

These proposed approaches provide some hints of solutions to the newcomers issue. However, it is necessary to experiment how efficient these methods are. Definitely, the problems of newcomers service and users both combined represent the startup problem of a recommender system, also called cold start or bootstrap problem. We can not predict whether the proposed approaches will be sufficient to overcome this critical phase of a recommendation system but we believe that practical experimentations are the appropriate way to properly evaluate these approaches.

3.9 Step-by-step approach evaluation

In this section, we propose to evaluate the performance and the behavior of the algorithm through the observation of different parameters and variation of different

variables through the step-by-step approach implementation. In fact, before going beyond, we needed to have a deep understanding of how this algorithm may behave regarding different tuning parameters.

3.9.1 Dataset generation

For evaluation purpose, we have generated many datasets representing users usages for services. Datasets were generated randomly following two statistical distributions namely the uniform and power-law distributions.

3.9.2 Experimentation protocol

To implement the proposed algorithm, we have used the *SoCo* framework introduced in [Maaradji 2010a]. This framework provides a graphical environment for the user to create Mashups. It includes the recommendation algorithm to provide dynamic suggestion to users. Figure 3.5 shows a screenshot of the *SoCo* framework. On the left side, *SoCo* displays the list of basic services below the reduced list fo suggested services (recommendation list).

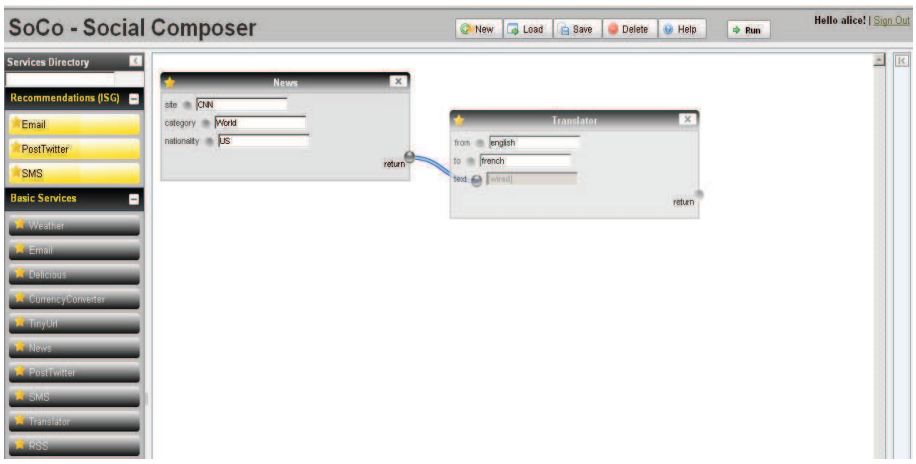

Figure 3.5: A SoCo screenshot of services suggestion after selecting a service

Based on a Web client/server architecture, SoCo captures user-service interactions on the client side; information that is stored and then modeled to supply the recommendation system (algorithm) on the server side. During the editing process, the user receives dynamic recommendations computed by the algorithm. For implementations needs, we used the PHP programming language, MySQL and Apache server (Intel Core 2 Duo P8600 machine with 2.4GHz and 2GB RAM).

We have considered in our evaluations, as a first step, the response time (i.e., recommendation time for a successor of a service) as the main performance indicator. More explicitly, we measure the duration it takes to the algorithm to respond to a query for a given configuration. It is naturally suitable that the algorithm answers

quickly to queries to ensure the system's interactivity with the user. To perform this measurement, the following parameters are used in this simulation:

- The size of Mashups represented by the number of services that make up those Mashups;
- The overall number of services $|S|$ or Mashups $|M|$ stored in the system. The number of Mashups is generally proportionally larger from 1 to 3 times to the number of services according to observations collected on *ProgrammableWeb*[1] ($|M| = 3 \times |S|$);
- The number of users in the system $|U|$.

Another parameter that may impact performances is the of the recommendation list. This parameter is not considered in our case. In fact, the proposed algorithm doesn't consider this information since it calculates the recommendation confidence for all potential successors to the current service. Given the three performance parameter defined above, we conducted three experiments. For each experiment, we fixed two parameters and we varied the last one. Due to the lack of benchmarks, the data sets used in these experiments were generated randomly. More concretely, services that compose a Mashup are independently and uniformly selected. A less important parameter representing the creation of a Mashup by a user is also a randomly generated relation. We run each experiment 25 times: 5 times for 5 randomly generated couple of *(u_i: current user, s_j: current service)*, and each point in the curves shows the average.

3.9.3 Evaluation

3.9.3.1 Users directory size

As a third experiment, we evaluated the impact of the number of users on the algorithm's performances. Indeed, we set (i) a number of services $|S| = 10^4$ (the number of Mashups $|M| = 30 \times 10^3$), (ii) the size of Mashups is uniformly distributed between 2 and 5, and we vary the number of users $|U|$. Figure 3.6 shows the obtained results where the algorithm response time remains stable even if the number of users increases to up to $|U| = 10^5$.

3.9.3.2 Services directory size

As a second experiment, we vary the number of services $|S|$ in the system. Thus, we fix the number of users set to $|U| = 10^4$ and the size of Mashup uniformly distributed between 2 and 5. Varying the number of services has an impact on the number of Mashups. Figure 3.7 illustrates the obtained results. We can notice that the algorithm's response time is not exponential, but could be approximated by a linear regression.

[1] http://www.programmableweb.com

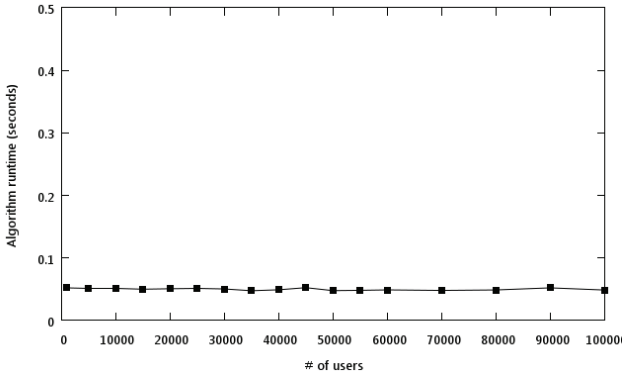

Figure 3.6: Impact of the number of users on the the performances of the algorithm

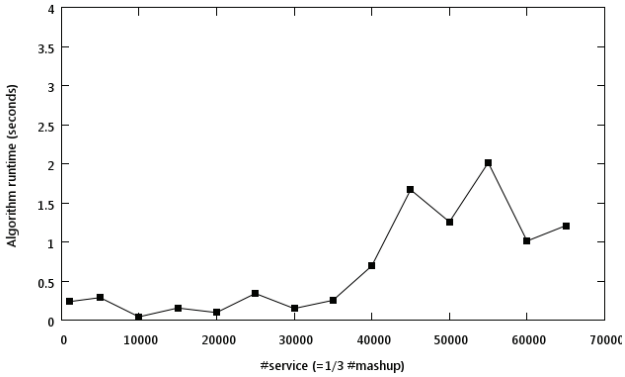

Figure 3.7: Impact of the number of services on the performances of the algorithm

3.9.3.3 Mashup size

To measure the impact of the Mashups size on the performances of the algorithm, we have started by fixing the number of users $|U| = 15 \times 10^2$ and the number of services $|S| = 1740$. We played with the variation of the size of Mashups (in terms of services composing each Mashup) from 2 to 7 services. Figure 3.8 shows the impact of the Mashups size on the behavior of the algorithm. Generally, the size of a Mashup has a tangible impact on the algorithm which is materialized by the response times for recommending a service. We can notice also that the response time of the algorithm increases linearly with respect to the increasing size of the Mashup (even if the takeoff of the curve is dry).

Other simulations were performed with more realistic statistical distributions that has impacted the behavior of the algorithm. Indeed, as pointed out previously, several studies [Yu 2009b] have shown that the popularity of services used for Mashup creation follows a long-tail distribution meaning that some services are much more frequently used than others. For example, mapping services are the most used for Mashups. As a social network, the graph representing the links between users, is characterized by its special features such as small-world property,

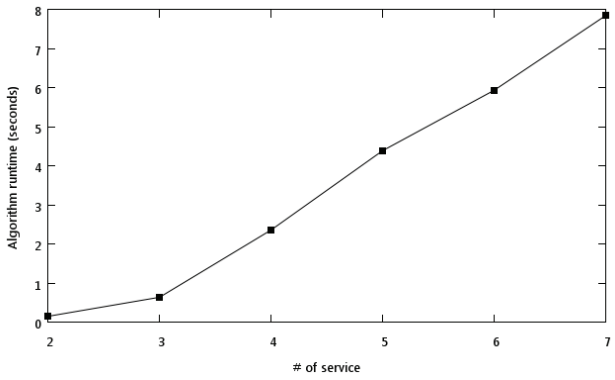

Figure 3.8: Mashup size impact on algorithm execution time

nodes' degree distribution following a power-law distribution. These features could then be leveraged for better evaluating the system. Concretely, we have generated datasets following a long-tail distribution (Zipf's law) that have been used as input of the recommendation algorithm. The results show that algorithm response time has decreased compared to the response time for uniformly distributed datasets (see Figure 3.9).

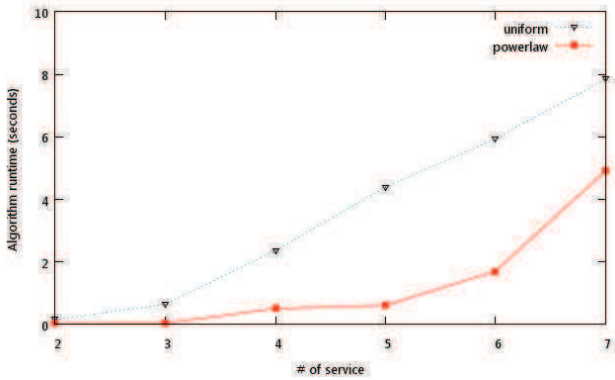

Figure 3.9: The long-tail distribution impact on algorithm performance

3.10 Synthesis

3.10.1 Results interpretation and Learned lessons

Overall, these results are interesting and show that this algorithm is suitable for an interactive application. However, they show that the response time of the algorithm is particularly sensitive to the size of Mashups which is, according to related studies and our observations on *ProgrammableWeb*, distributed between 2 and 5 (rather close to 2). From a more general perspective, we believe that there is still room for improvement by optimizing the recommendation strategy.

Even if one could consider step-by-step completion as interesting in that it provided ongoing assistance to the user, the fact remains that it could be improved. Considering the analogy between Mashup completion and word completion (as in a Web search engine). Completing a Mashup step by step would by analogy for word completion be the suggestion at each step the following letter for the part of word already introduced by the user. This is obviously onerous and heavy for word completion and is also for Mashup completion. The natural solution would be to suggest the entire completion for the part of a word introduced but the user, in the case of word completion. Similarly, it would be more useful for the user to receive suggestions that are full completions of the partially introduced Mashup. The advantage of this method is to further accelerate the process of composition.

Chapter 4

Full Mashup Completion Based on Frequent Sequence Mining

4.1 Introduction

Since creating Mashups is now an emerging trend, as discussed in Chapter 2, semi-automatic Mashup composition will assist in developing Mashups in a faster and user-tailored manner. In fact, different types of interactions occurring between entities are involved in a Mashup's creation. By learning from a sequence of services that are integrated progressively, one can proceed to recommend services that complete a Mashup. Typically two categories are distinguished: (i) *single completion*, in which a list of potential services is suggested to the user on the basis of the currently selected service [Goethals 2003] and described in the previous Chapter, and (ii) *full completion*, in which the whole composition (or a part of it) is recommended to the user [Greenshpan 2009]. This Chapter focuses on an approach to full Mashup completion and aims at predicting and recommending the most interesting combination of services that should follow a current service creation flow. The problem we are tackling here could be summarized as follows: *Given a user creating a Mashup within a Mashup creation platform, how can the platform suggest the finished Mashup that best meets his/her intentions, within a reasonable amount of time?* The goal of the proposed approach is to improve Mashup creation time and quality while addressing the following constraints:

1. *terminating condition:* In recommending a full completion, there is no specific boundary to completions size due to the fact that the number of required services to complete the Mashup is not known a-priori. Therefore, recommendation of services involves an unknown parameter that increases the complexity in Mashups full completion.

2. *scalability:* since the number of potential candidates to a full completion is combinatory larger than the number of candidate services to a transition completion (see Section 3.7.5). This directly impacts the system's scalability for proposing solutions.

3. *recommendation detail level:* Given the enormous variety of different services and resources that are at the disposition of a recommender system for meeting the end user's intent and to consider his social environment.

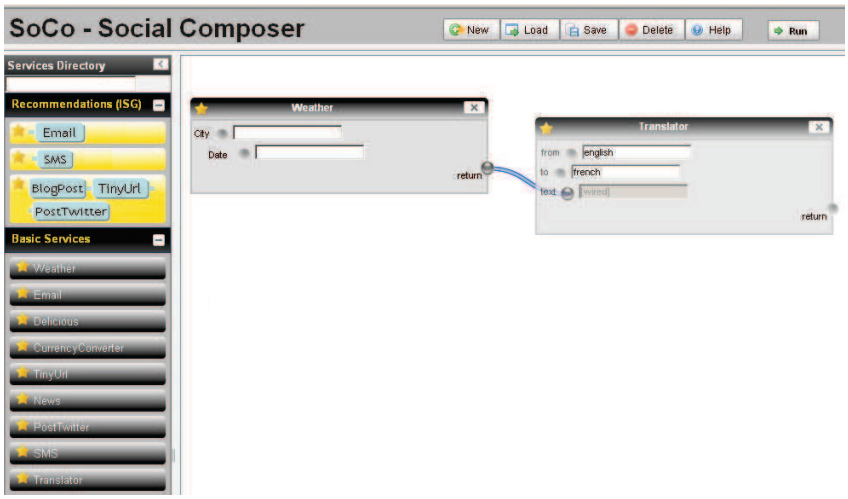

Figure 4.1: Illustration of the front-end of SoCo, the tool integrating our proposal

This work addresses the aforementioned constraints by identifying frequent Mashup combinations and capturing users' social interactions over them. This approach is used for predicting and suggesting the next services that will complete an initiated Mashup by exploiting both co-occurrence frequencies and social interactions on earlier composed services. In this regard, we propose to follow the enumerated steps that meet one-by-one the previously mentioned problems:

1. modeling the problem of full completion as a frequent sequence mining problem. Simply use existing frequent sequence mining techniques as basic solution.

2. addressing the scalability issues related to the existing frequent sequence mining algorithms. This is performed via the introduction of a new frequent sequence mining algorithm, called *FESMA*. *FESMA* offers a high performances in terms of computation time outperforming the existing algorithms in our context.

3. more precise and personalized full completions. This is achieved through the introduction of a social dimension in the process. The social dimension is essential to this work. In fact, in Web 2.0, people can create, use, and share services. We assume that Mashup environments reflect the social behaviors of users and thus, social structures can be extracted from the interactions between users and services (and between users). These interactions can be analyzed and injected as a social information into the process of full completion for services discovery and composition.

The proposed approach has been implemented and evaluated in the framework developed at Bell labs. Figure 4.1 show the interfaces corresponding to the Mashup full completion feature with three dynamic full completion suggestions.

4.2 Data model

Basically, the idea of full Mashup completion is to predict the remaining part of a Mashup during its creation by a user, based on the current introduced initial part. Figure 4.2 illustrates three Mashups created by different users in which different services are combined together to achieve specific goals. It can easily be observed that there is a recurring configuration of services that appear together in the different compositions. We illustrate one of them represented by the chain: $w_3 \rightarrow w_4 \rightarrow w_5$. Since this configuration is repeated, it would be interesting to suggest it whenever similar composition schemes are started.

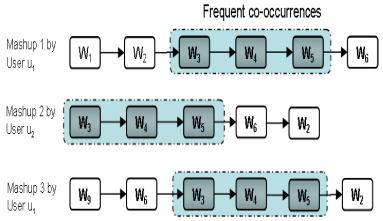

Figure 4.2: Example of services co-occurrences in different compositions

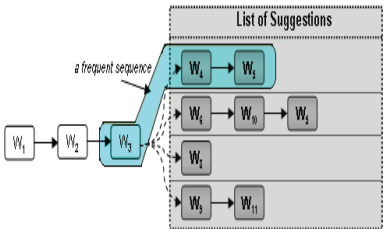

Figure 4.3: Illustration of full completion applied to the example of Figure 4.2

Intuitively, a services composition is based on the combination of different services together where the output of a service w_i is the input of service w_{i+1} (or a part of it) immediately following w_i. This results in different chains of services producing different patterns. Thus, Mashups (and services compositions in general) can be considered as sequences of services [1]. Let $W = \{w_1, ..., w_n\}$ be a set of n items ($|W| = n$) that we explicitly call Web services from now on. Let $S = \{s_1, ..., s_m\}$ be a set of m sequences ($|S| = m$). A sequence, representing a Mashup in our case, is defined as an ordered set of services denoted by $s_i(w_1 \rightarrow w_2 \rightarrow ... \rightarrow w_k)$ where w_i ($i = 1, ..., k$) represents the i^{th} Web service.

For each sequence we associate: (i) the length k, denoted $len(s_i)$, of the sequence defining the number of succession of services included in the sequence (i.e., with repetition) (ii) a set $pref(s_i)$ representing the set of prefixes of a sequence s_i. As commonly used for strings, a prefix represents a subsequence having as a first service the first service of s_i with a length less than $len(s_i)$. As an example, let's consider the sequence $s_i(w_1 \rightarrow w_2 \rightarrow w_3)$ then $pref(s_i) = \{(w_1), (w_1 \rightarrow w_2)\}$. Finally, we associate a set $suf(s_i)$ representing the set of suffixes of a sequence s_i which represents all the subsequences having as a last service that of s_i. As an example, let's consider the sequence $s_i(w_1 \rightarrow w_2 \rightarrow w_3)$ then $suf(s_i) = \{(w_3), (w_2 \rightarrow w_3), (w_1 \rightarrow w_2 \rightarrow w_3)\}$. Note that we didn't consider the longest sequence in this example for simplicity matters.

[1] This should not be confused with the composition patterns that include sequence operations, parallel operations, etc. but rather the way services are modeled at a logical level.

4.3 Frequent sequence mining for full completion

Coming back to the problem of services full completion, the basic idea is to identify and count recurrent subsequences in compositions that have been previously created by users within the system. Those frequent sequences represent actually, on one hand, the composition behaviors of each individual, and, on the other hand, the common habits and behaviors shared implicitly between groups of users. The problem then is reduced to finding frequent subsequences having the same prefix as the *query sequence* of a certain length $k \in \{1, ..., max(k)\}$.

The natural approach is to investigate the state of the art of *frequent pattern mining* algorithms, and select one of them that is adapted to the Mashup full completion context. In this regards, we briefly review in the following related work to full Mashup completion and frequent pattern mining algorithm.

4.3.1 Frequent sequence mining applications

Mashup full completion is an emerging field that seeks to complete a composition of services supplied by the user. To the best of our knowledge, there is no study that addresses directly this issue (except [Greenshpan 2009]). Therefore, the problem has been compared to similar work in other areas leading to studies in many fields: words and phrases full completion, DNA sequence prediction, travel itinerary recommendation, and others dealing with sequence mining [Dong 2007].

The most frequent case in full completion occurs in the context of search engines where full completion of words displays strings that are the most relevant to complete the introduced prefix (typed words or letters). Classical approaches, such as suffix trees [McCreight 1976], could not be used directly in our case. In fact, in words full completion, only whole words are considered in the training phase without taking into account sub-strings within those words. However, ideas coming from this field were a source of inspiration for different proposals. Typically, [Chaudhuri 2009] has shown fault-tolerant full completion considering variants of the introduced prefix, a feature targeted as a future direction to our work. Another similar topic is predicting user actions based on user logs and preferences. For instance, [Davison 1998] predicts *UNIX* commands that a user may enter based on previously entered sequences. As mentioned before, full completion is based on frequent pattern mining.

In the area of frequent sequence mining, many algorithms , issued from the area of frequent item sets mining, have been proposed like Apriori [Agrawal 1993] and Eclat [Zaki 2000]. However, the most known algorithm for frequent sequence mining is SPADE [Zaki 2001]. SPADE has been defined for the particular case of frequent sequent mining. Similar to Eclat, SPADE uses a vertical representation of a sequence database with simple joins (intersection). Furthermore, it uses a lattice-theoretic approach to decompose the original search space in order to be processed separately in the main memory. SPADE algorithm scans the database only 3 times, leading it to outperform some sequence mining algorithms as AprioriAll [Agrawal 1995] and GSP [Srikant 1996]. However, authors in [P. 2001] have argued that the *prefixspan*

proposed algorithm is mostly faster than the SPADE algorithm. Finally, Bodon in [B. 2005] has shown that their APRIORI implementation (named here *ArioriSeq*) always outperformed (time and memory usage) *prefixspan* [P. 2001] with high support thresholds and also with low thresholds on databases with long transactions.

In order to choose the appropriate algorithm, the previous section has shown that no algorithm had shown significantly better performances than the others [Goethals 2003]. Actually, frequent pattern algorithm performance are heavily tied to the nature of the dataset that has been used to measure it. Each algorithm has shown better or worse performance depending on the dataset density and distribution.

In our context, scalability was given the first priority since we need to process completion queries in real time and return better personalized completion suggestions. This latter (personalized completion) require fine grained suggestion that negatively impact existing algorithms performance. We propose then a new algorithm for frequent sequence mining to tackle at a first stage, the scalability problem.

4.4 Fast and efficient sequence mining algorithm FESMA

The algorithm we propose is called *FESMA* for *Fast and Efficient Sequence Mining Algorithm*. Just like the *FP-growth* algorithm [Han 2000], *FESMA* doesn't generate any candidates and uses a compact prefix tree representation to store all sequences (i.e., sub-Mashups) that exist within the transactions database (i.e., all created Mashups). By contrast to other sequence mining algorithms, *FESMA* scans the database only once. During this scan, and for each transaction, all sequences are added to the tree representation by updating the support associated with each sequence and the user's specific supports. We named that tree *FSTree* for frequent sequences tree.

4.4.1 Algorithm design

4.4.2 Algorithm complexity analysis

From the *FESMA* algorithm definition, we can see that one needs exactly one scan of the database to parse existing Mashups (i.e., transactions). The cost of parsing the database is $O(m)$, where m is the size of the database. In order to update the sequence tree *FSTree* with subsequences, each transaction is parsed once. The cost of inserting a sequence in the tree depends on the sequence length (depth of the tree). In the worst case, this operation costs $O(K^2)$ with K corresponding to the size of the longest sequence. In summary, the overall complexity of the algorithm in the worst case is $O(m \times K^2)$.

4.4.3 Some illustrations

In order to illustrate the algorithm, let's consider the simple example of Table 4.1 which shows three Mashups and their associated users as input.

Algorithm 2 Fast and Efficient Sequence Mining Algorithm (FESMA)

```
1: S = {s_j, j = 1, ..., m} {list of sequences}
2: for j = 1 to m do
3:   {scan all Mashups}
     s_j := GetSequence(j)
4:   for k = 1 to len(s_j) do
5:     {parse all subsequences of s_j}
       SSeq := subsequence(s_j, k)
6:     if SSeq ∈ FStree then
7:       Update the corresponding branch by incrementing nodes support in
         FSTree
8:     else
9:       Create a branch and set its node support to 1
         {update the FSTree prefix tree}
10:    Update users supports
11: Return FStree
```

Table 4.1: illustration of Mashups database

ID, user	Transactions
$Mashup_1$, user 'a'	$w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6$
$Mashup_2$, user 'b'	$w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6 \rightarrow w_2$
$Mashup_3$, user 'a'	$w_9 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_1$

As the algorithm visits every Mashup in the database, the FSTree is updated to keep a current count of all the subsequences encountered, as follows: for every possible suffix of the current Mashup, a path corresponding to that suffix is followed through the $FSTree$ incrementing the value of existing nodes that are visited, and creating new nodes if necessary (with a value of 1) to finish the path. For instance, let's consider $Mashup_1$ of user 'a' which generates the following subsequences: $(w_1), (w_1 \rightarrow w_2), (w_1 \rightarrow w_2 \rightarrow w_3), (w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4), (w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5), (w_1 \rightarrow w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_2), (w_2 \rightarrow w_3), (w_2 \rightarrow w_3 \rightarrow w_4), (w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5), (w_2 \rightarrow w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_3), (w_3 \rightarrow w_4), (w_3 \rightarrow w_4 \rightarrow w_5), (w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6), (w_4), (w_4 \rightarrow w_5), (w_4 \rightarrow w_5 \rightarrow w_6), (w_5), (w_5 \rightarrow w_6), (w_6)$. When updating the $FSTree$ with $(w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6)$, this will actually update the tree with $(w_3), (w_3 \rightarrow w_4), (w_3 \rightarrow w_4 \rightarrow w_5), (w_3 \rightarrow w_4 \rightarrow w_5 \rightarrow w_6)$. This process is repeated on the whole sequences (all Mashups). The tree illustrated in Figure 4.4 is provided as an output.

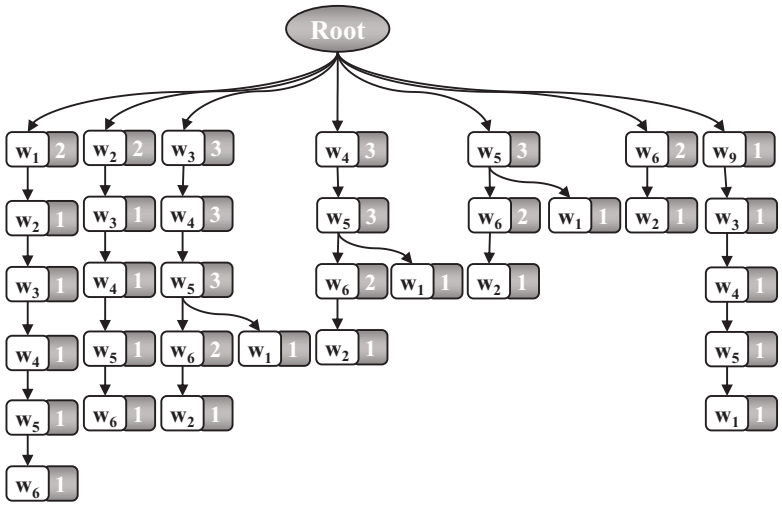

Figure 4.4: Illustration of the output tree after the execution of FESMA on the example of Table 4.1

4.5 From community to social fine grained full completion

At this stage, we have succeeded in adapting the frequent sequence computation and making it more efficient via a faster computation and limited database scans. In this section, we focus on the use of the generated representation and the computed sequences. Intuitively, when processing the set of sequences using *FESMA* or any other frequent sequence mining algorithm, the only information we have is the frequencies of subsequences, providing a strictly global perspective for possible completion strategies. In other words, since the co-occurrences are computed according to their appearances over all existing sequences, this process considers only the aggregation of the behavior of all existing users regarding the most popular sequences. Thus, any assistance can only operate at a high level of granularity, i.e., community or global, equally valid for one user as for another, yet customized for neither.

In the following, we describe an enhanced community-based strategy for ranking the completion lists which improves on this global perspective of the direct application of frequent sequence mining algorithms to the full completion problem. Afterwards, we introduce and motivate a fine grained strategy based on social networks implicitly extracted from the analysis of interactions between the entities of the system.

4.5.1 Community-based recommendation

This functionality can be achieved by using any frequent sequence mining algorithm. At this stage, it is necessary to keep in mind that we are aiming at offering support

for end-users (e.g., under the form of recommendations) to easily build her Mashup. Applying the aforementioned algorithms produces a set of subsequences with their frequencies as defined in Equation 4.1:

$$S' = \left\{(s'_i, freq(s'_i))/s'_i = (w_1 \rightarrow ...w_l) \wedge l \leq Argmax(len(s_i))\right\} \quad (4.1)$$

Where $freq(s'_i)$ is the frequency of subsequence s'_i in the initial set of sequence S, and $Argmax(len(s_i))$ is the length of the longest sequence in the initial set playing the role of highest limit, i.e., it is obviously not possible to find subsequences longer than the longest sequence in the initial sequence set. Depending on the algorithm used, this output could be represented and indexed as a tree. A *query sequence* s_q is sent to the system in the form of a service or a sequence of services (i.e., built from an initial successive combination of services). The system selects candidate sequences from S', where the prefix of the candidate subsequence is a suffix of the *query sequence*. All selected subsequences represent potentially interesting answers for completing s_q. At this stage, according to a predefined strategy, the recovered sequences are ranked by their relevance and only the $top-k$ sequences are proposed to the user. Algorithm 3 provides an abstract description of the completion process.

Algorithm 3 Completion abstract algorithm

1: s_q: {*the query sequence*}
   S: {*the set of existing Mashups*}
2: $S' = FSM(S)$ {*mine frequent sequences*}
3: **for all** each $sq'_i \in suf(s_q)$ {*all suffixes of the query*} **do**
4: $TempList$ = Select s'_i from S' where sq'_i is prefix of s'_i
5: $RecList \Leftarrow RecList \cup TempList$
   {*building the recommendation list*}
6: Rank $RecList$
7: Return $top-k$ elements of $RecList$

Basically, the full completion algorithm cost (complexity) depends on the length of the *query sequence* $|s_q|$. In fact, for each suffix of the *query sequence*, the algorithm retrieves completions from the frequent subsequences list. This makes the use of traditional frequent sequence mining algorithm unsuitable in this context[2]. Our alternative approach uses the $FSTree$ representation which can be traversed with more efficient computation and access times. Once the branch of the *query sequence* suffix is retrieved, one needs just to browse that branch to access the most frequent sequences (with additional "meta-data" if it exists).

4.5.2 Social networks based recommendation

Based on the limitations of the community based approach, we propose to integrate a social dimension to the previous solution in order to leverage *Social Networks*

[2]The execution times of existing algorithms are discussed in the evaluation section.

Analysis (SNA). This is motivated by the need for deeper exploitation of the huge amount of information generated by users interactions in the Web 2.0 paradigm. We believe that this will lead to a fine grained, more precise and personalized support for users. Introducing this dimension incorporates the interesting observation that a user is more interested in the recommendations that come from members of his social networks (family, friends, colleagues, etc.) or from people with whom he shares common interests. Our full completion strategy takes into account users' specificities that are reflected in their different service composition behaviors. We consider interactions that involve end-users as social interactions, and part of the social dimension. The remaining type of interactions, i.e., those between services, is considered as a structural support for the approach since such information is necessary to, e.g., ensure that the input of service w_{i+1} is compatible with the output of service w_i.

4.5.2.1 Augmented FESMA

The frequent sequence mining algorithms, and even *FESMA*, do not consider a very fine level of granularity since (i) they mainly operate at a global level and (ii) they reason about one type of entities, i.e., services. Thus, they need to be adapted not only to keep track of social information but also to support the high number of possible combinations due to the introduction of the user in the process. A social network in this context is then defined as an abstraction of interactions that occur between people and services in Web services environments, capturing the behavior of social entities in the form of a social graph. This structure may be inferred or extracted directly from common interests between the users of the composition platform. The principle is based on the transformation of $user \rightarrow services$ interactions to a $user \rightarrow users$ social network on top of which statistical processes are applied to, e.g., fire recommendations for assisting the user in constructing the Mashup. Impacts and interests of the social dimension have been introduced in [Maaradji 2010b] and were heavily discussed. Since the objective here is to show how this dimension is leveraged for building sophisticated full completion strategies, we don't detail this aspect further in this section of the document.

With this new constraint, the method has to enumerate and count not only the support for each subsequences (i.e., number of occurrences), but the specific sequence support for each user. In other words, each node is related to the users who have used the subsequence it represents. This information is associated in the form of an array capturing: user u_i has used subsequences s_j, l times. In order to reduce the construction cost, this information is updated while building the tree. The result is illustrated in Figure 4.5. Thus, Equation 4.1 needs to be revisited to incorporate this level of granularity, as in Equation 4.2:

$$S' = \left\{(s'_i, u_j, freq(s'_i, u_j))/s'_k \in S' \wedge u_j \in U\right\} \tag{4.2}$$

In terms of algorithm complexity, adding the users' specific sequence occurrence within the FESMA sequence mining algorithm, i.e., Algorithm 3, impacts not im-

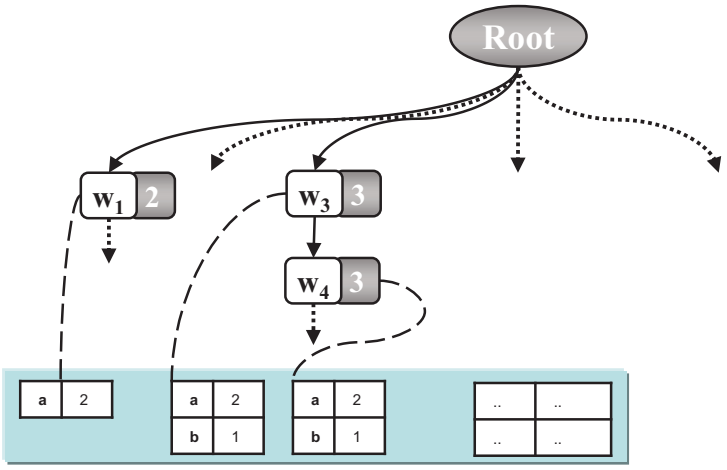

Figure 4.5: *FSTree* after the addition of the users information

pacts not only the algorithm computation resources but also the memory space occupied.

4.5.2.2 From services sequences to social completion

In order to consider that social dimension, we propose in the following an efficient strategy for a social-based full completion approach based on the construction of an implicit *social graph* between users. We consider the resulting graph as social since it captures the behavior of users regarding services composition and their potential common interests.

Besides the $users \rightarrow (single\ services)$ relationship, we consider $users \rightarrow sequences$ interactions as a bipartite graph [Guillaume 2004] that represents how frequently users include sequences in composition schemes. Figure 4.6 illustrates a bipartite graph of sequences and users. The links represent the usage frequency denoted by $f(u_i, s_j)$, which a user u_i has of a sequences s_j in all the compositions he created. To transform the bipartite graph into a social graph to help rank recommendations we rely on three main steps: (i) local information extraction, (ii) semi-global information extraction, and (iii) global information extraction.

Local information extraction The *local information* considers only the interaction between a specific user and a specific sequence. This information tells us whether a specific user is confident (i.e., expertise indicator) using this sequence among other sequences. To materialize this idea, we define this information in a

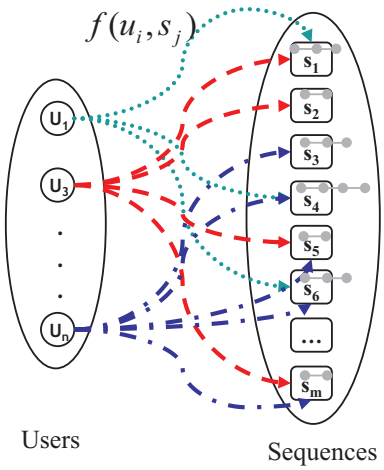

Figure 4.6: Illustration of a bipartite graph between users and services

quantity called *Activity* defined in Equation 4.3 where M is the total number of sequences a user u_i exploited in her different compositions.

$$Act(u_i, s_j) = \frac{f(u_i, s_j)}{\sum_{k=1}^{M} f(u_i, s_k)} \tag{4.3}$$

Semi-global information extraction At the level of semi-global information, we consider the interest a user may have in other users regarding a given sequence. Thus, for a given user u_i we calculate how much the sequence s_j recommended by the user u_l matters to her. This is called *Special Interest* (SI) and is calculated using Equation 4.4.

$$SI(u_i, u_l, s_j) = \frac{f(u_l, s_j)}{f(u_i, s_j)} \tag{4.4}$$

Global Information extraction In order to have as precise transformation as possible with less data loss, we add another level of information in the transformation process. The global information captures whether a couple of users have common general interests or not. At this stage of our study, and for simplification reasons, we consider that the general interest of a couple of users is equal to the sum of their specific interests, thus building the implicit graph as illustrated in Equation 4.5. The output of this step is a users' social graph aggregating all the specific interests graph obtained previously.

$$IG(u_i, u_l) = \sum_{k=1}^{M} SI(u_i, u_l, s_k) \tag{4.5}$$

Services recommendation strategy Once the bipartite graph is transformed to a social graph thanks to the three previously described steps, we proceed to recommendation calculation to suggest a coming sequence according to a selected *query sequence*. Thus, considering the intrinsic user's usages frequency (local information), the specific interest between two users (semi-global information), and the implicit graph (global interest between users), we define RC of a given prefix $freqseq$ according to the introduced sequence s_q for the user u_i as follows (see Equation 4.6):

$$\begin{aligned} RC(u_i, s_q, freqseq) = & \sum_{l=1}^{N} SI(u_i, u_l, freqseq) \\ & \times Act(u_l, freqseq) \\ & \times IG(u_i, u_l) \end{aligned} \tag{4.6}$$

The recommendation confidence is the metric that indicates how a completion is important to the user. Concretely, when the user u_i is creating a new composition of services, and has entered s_q as prefix for Mashup full completion, a ranked list of recommended Mashup completions is proposed in decreasing order of recommendation confidence RC.

4.6 Implementation and evaluation

In this section, we discuss the performed evaluations on the method to validate our proposal. We have performed mainly two kinds of evaluation: (i) a comparison evaluation in which we compare the performances of our approach to four existing frequent sequence mining algorithms, and (ii) an evaluation of particular properties of *FESMA* to measure the overhead generated by the consideration of the social dimension.

4.6.1 Experimentation protocol

The choice of the dataset is important to measure the performance of the proposed approach. Evidently, being a succession of services, Mashups have their own statistic properties, e.g., their distribution and their length (according to analysis of available data on ProgrammableWeb [Yu 2009b]). Thus, the dataset which can be used need to respect the behavior of real world observations. On the other hand, another important aspect, especially when comparing to other methods, is to select datasets which are supported by existing approaches. We have decided to use the synthetic data generator from "*IBM quest data generator*". For instance, IBM-Artificial dataset $T10k - L5$ contains 10^5 transactions (defining Mashups in our case) and the average sequence length is equal to 5.

Generally speaking, the main performance criteria used to evaluate this kind of methods are: (i) the execution time and (ii) the memory space required by each

algorithm to find frequent sequences in a dataset. It should be noted that in the case of *FESMA*, this time includes reading the dataset from an input file and writing results to an output file (costly operations in terms of time). *FESMA* is implemented using a standard C++ library. Finally, all the test are performed on an Intel Core 2 Duo T9600 With 2.8GHz of processor and 3GB of RAM.

Table 4.2: List of datasets used to evaluate the performances of FESMA

Dataset name	Number of transaction	Transactions average length
T100k-L2.5	10^5	2.5
T200k-L2.5	2×10^5	2.5
T500k-L2.5	5×10^5	2.5
T1000k-L2.5	10^6	2.5
T100k-L5	10^5	5
T200k-L5	2×10^5	5
T500k-L5	5×10^5	5
T1000k-L5	10^6	5
T100k-L10	10^5	10
T200k-L10	2×10^5	10
T500k-L10	5×10^5	10

4.6.2 FESMA Vs AprioriSeq

In order to compare FESMA to state-of-the-art frequent sequence mining algorithms, we run it over a bunch of datasets. Table 4.2 shows a list of datasets used to evaluate the proposal with a comparison to existing algorithms. We illustrate the comparison results between FESMA and Apriorise q [B. 2005] on 3 different datasets represented in Figures 4.7, 4.8, and 4.9 respectively. We could not reproduce the experiments using other algorithms due to some compilation problems of the code available on different Web sites. We could not even reproduce the obtained results published the literature, even with the use of the same datasets. The hardware configuration not being the same, we believe the comparison would be inaccurate and unfair.

Regarding the obtained results, it appears that there is a clear gap between the results obtained by *AprioriSeq* and *FESMA* with a better performances for *FESMA* on all the configurations of the support (i.e., x-axis). We believe that with these results, even other algorithms will be outperformed. Another interesting observation regarding *FESMA* is its ability to manage large datasets formed by very short frequent sequences that generally pose a problem to existing methods. Finally, it can be easily observed that *FESMA* is stable after considering a minimum support of 2 services. This means that the support doesn't influence the performances of the method too much contrary to other existing methods.

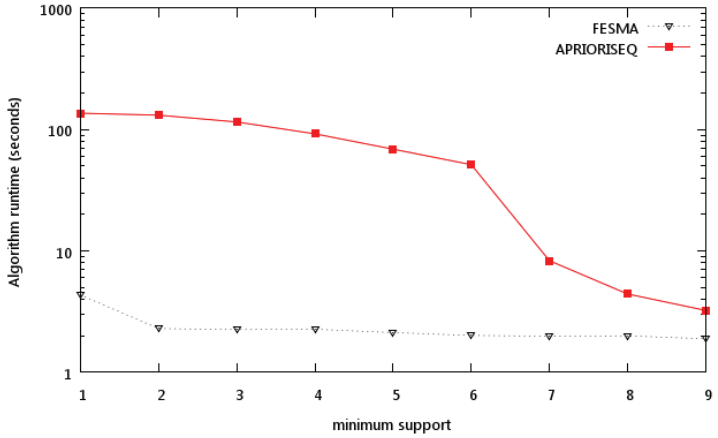

Figure 4.7: FESMA vs Apriori runtime over support (loglog scale) on $T100kL2.5$ dataset.

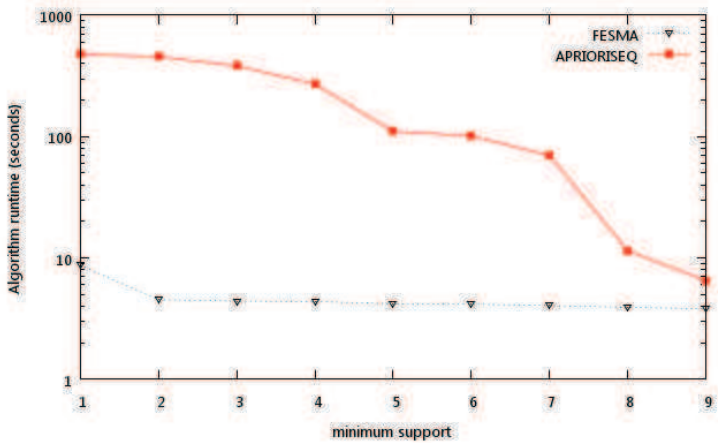

Figure 4.8: FESMA vs Apriori runtime over support (loglog scale) on $T200kL2.5$ dataset.

4.6.3 FESMA performances

Once this information checked, and since we could not use larger datasets with the implementation of the *AprioriSeq* that we have, we wanted to check the scalability of the proposed approach. We have considered the same datasets described in Table 4.2. The results are illustrated in Figure 4.10. Considering the size of the datasets and the minimum support, the results are satisfactory since the maximum time needed to build the tree with 10^6 rows is about 90 seconds. Note also the behavior of the algorithm which reproduces exactly the same stability for all the situations like the one observed before.

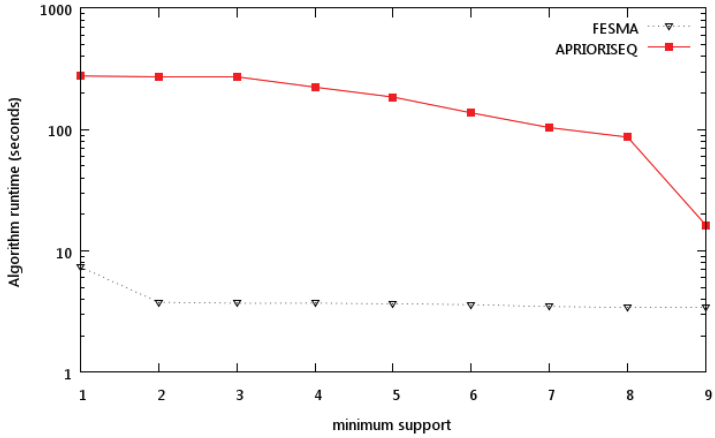

Figure 4.9: FESMA vs Apriori runtime over support (loglog scale) on $T100kL5$ dataset.

4.6.4 Social overhead

As a second experiment, we wanted to measure the overhead generated by the integration of the social dimension within *FESMA*. To evaluate this, we have modified an initial dataset, i.e., $T1014D100K$, by associating to each sequence a user identifier who is supposed to be the creator of such sequence (i.e., Mashup). We have generated a $User \longleftrightarrow Mashups$ association satisfying the most important property of social networks, i.e. the long tail of the activity distribution. This property argues that some users (Web-users) are much more active than others in terms of generated content (Mashups). Figure 4.11 illustrates the obtained results.

The results clearly show that the algorithm's runtime responses keep the same behavior with an average of 25% of overhead for social dimension which is reasanable regarding the personalizarion and social added-value features provided to users. In the same time, even with the overhead generated by the social dimension, the results are more interesting than all the existing algorithms without the consideration of the social dimension.

4.6.5 Completion runtime

At this stage we wanted to measure the response time of the completion strategy. Indeed, the completion strategy needs to satisfy interactive application requirements since it's supposed to provide real-time and dynamic recommendations and suggestions to users who are creating (editing) Mashups. As mentioned before, the completion algorithm uses the prefix frequent sequence tree generated by *FESMA* in order to retrieve the remaining piece of a sequence introduced by the user.

To perform this evaluation, we construct on the previous dataset and $FSTree$ and associate for each frequent sequence its users. Then, we consider different initial queries by different users (randomly selected from the database) while varying the size of each *query sequence* from 1 to 10, i.e. $len(s_q) \in \{1, ..., 10\}$. We recover then

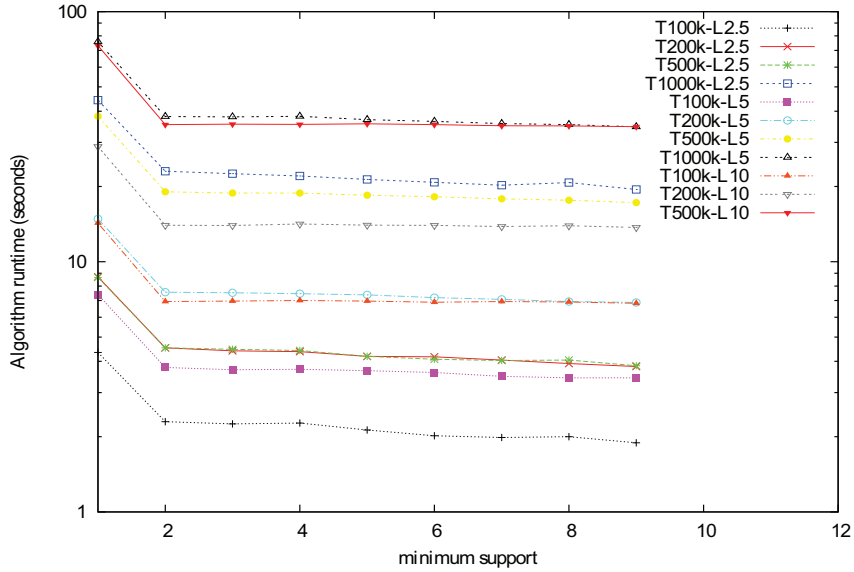

Figure 4.10: *FESMA* runtime on all the datasets

the maximum time for each value of the size. The results are illustrated in Figure 4.12 with time unit expressed in milliseconds. The results shown in Figure 4.12 illustrate the efficiency of the proposed completion strategy and its ability to support real-time queries , and show that the more the length of a *query sequence* is high, the more the response time decreases.

4.7 Comparison to the closest related work and conclusion

We have proposed and evaluated a full Mashup dynamic completion approach in order to assist end-users when composing services. We have clearly seen that the performance of our algorithms allow perfectly to satisfy application interactivity requirements. In order to better assess this approach, we propose in the following to slightly discuss our results with respect to a similar study in **??** even if they are not directly comparable. This is, to the best of our knowledge, the only work in the literature which is directly related to full Mashups completion. Our work has the same objective as that in [Greenshpan 2009] but from a different perspective. In fact, this latter study has shown a full mashup completion solution based on community approach with contrast to the social-based approach we are proposing. In Greenshpan et al. [Greenshpan 2009], the authors rely on services categories to compute completions of Mashups using a top-k strategy. In fact, we consider not only the community level but especially the individual level (how the end-users are related in social network) to compute completions. Another difference is that in our approach considers each service when computing the completions list, in contrast with service categories in [Greenshpan 2009]. Instead of requiring more

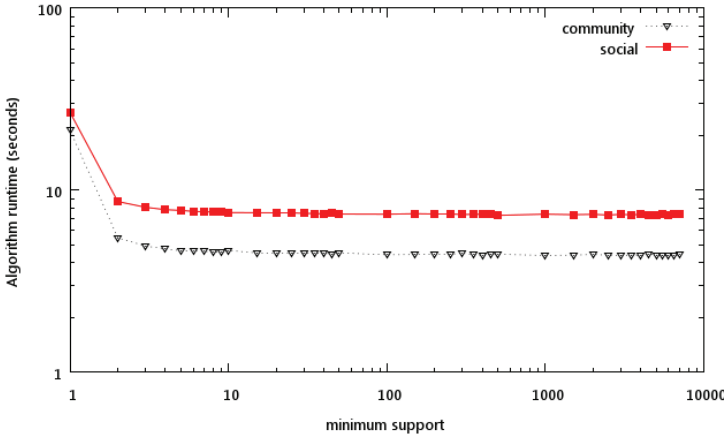

Figure 4.11: Overhead generated by the social dimension

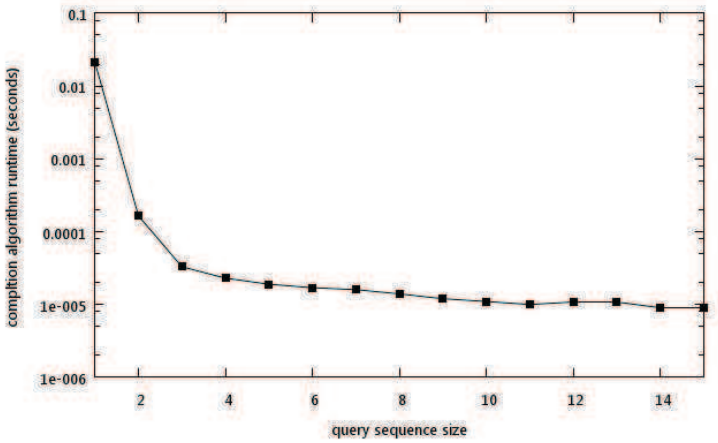

Figure 4.12: Obtained results on completion times

resources, the computational efficiency of our approach adequately copes with the increase in precision. The scalability of our approach is advantageous, using datasets five times larger and with more precision than in [Greenshpan 2009]. The main performance parameter used in **??** is the completion algorithm response time which presents results of about 0.5 second. Even if the social approach, that we are proposing, introduces more complexity by considering differentiation between users' composition patterns both in mining and completion steps, we succeed to reach less then 0.05 second for completion response time which outperforms the state of the art results. Our performance is mainly due to the FSTree data structure which presents optimal results (direct access) when querying it with any given partial mashup introduced by the user. Our approach to service full completion is innovative since this is the first approach that offers fine grained recommendations. Moreover, our approach leverages both the community-based principles and a social dimension with a well balanced importance thus providing the user with well targeted and

more personalized recommendations.

Chapter 5

Social Composer: An Augmented Mashup Creation Environment

5.1 Introduction

We presented in the previous two chapters two approaches that aim to assist end users in the process of service composition using a Mashup Creation environment. In addition to putting into practice these features, we implemented a complete environment for creating Mashups enriched with the dynamic service recommendation called Social Composer.

The current chapter describes in detail the Social Composer (*SoCo*). In the first section we present the general requirements of any environment for creating Mashups. Then, we detail the design and implementation of *SoCo*. Afterwards, we focus on the dynamic recommendation functionality by illustrating it through some use cases. We, also, show the integration of *SoCo* in a global service composition framework developed in the SERVERY European project.

5.2 Classic service composition environment requirements

In chapter 2, we reviewed existing Mashup creation environments such as Yahoo Pipes! and Open Mashup Studio. These environments share more or less the same basic architecture. They all offer, through a Graphical User Interface(GUI), the ability to end user to link graphically basic services. This abstract description entered by the user is then translated into an executable language which invokes services according to the logic described by the user. Thus, the common components of any Mashup creation environment are the directories of basic services and users, the GUI, and the orchestration engine.

5.2.1 Graphical User Interface

As previously mentioned, there are several tools that allow the user to express the composition logic to perform. The graphical interface particularly instantiates the concept of semi-automatic composition by offering the possibility to interact with the end user. It is used for graphically connecting services of particular interest to the user in a way that express the composition logic. It provides a level of abstraction that incorporates the principle of workflow, which already exists as a workflow in the enterprise context. Indeed, a Mashup expressed through this type of interface allows

expressing, with relative accuracy, the logic of the desired composition. Depending on the level of abstraction, expressed composition schemas are to a certain extend similar services orchestrating scripts described below. Commonly, basic services are presented in a list from which the user selects the services he wants, drags/drops them on an edition area, and then connects them.

5.2.2 User directory

Any Mashup creation environment has to provide the basic functionality for managing user accounts. This feature allows the user to manage the schemas of composition with the following operations: create, edit, deploy (implement), and publish a composition schema. These operations are particularly important in our system because they express the user-service relationship. A directory of users and their interactions are maintained in a database to ensure these operations.

5.2.3 Service directory

In every Mashup creation environment, a directory of basic services is maintained. It stores descriptions of basic services. As shown in chapter 2, many languages for service description and protocol for service discovery and publication are proposed. The couple UDDI et WSDL represents the most popular configuration used for Web services. The WSDL description represents a service (operations input/output and the endpoint). The UDDI technology database ensures the discovery and publication operations of WSDL descriptions. The created composed service (Mashup) should be registered to be reused later as a basic service in a new composition schema.

5.2.4 Orchestration engine for dynamic service composition aspect

Once a composition logic is expressed in the edition area and validated by the end user, that description is translated into an executable script that has to orchestrate basic services according to the expressed logic. There are a number of service orchestration languages but BPEL has been accepted as the De-facto standard in the market due to its usefulness. There are other languages that could orchestrate the services that include SPATEL, BPEL4J and BPMN. To support the semi-automatic service composition, we had to choose an easy way to implement orchestration language, so we narrowed down our research to BPEL and SPATEL after carefully examining the pros and cons of other languages.

Compared to BPEL, SPATEL is not a fully specified language, and has no support material on the Internet. It doesn't have any mechanism to seamlessly integrate external services that are not hosted within the SPATEL orchestration engine. Currently, the SPATEL engine requires that every external service has to be explicitly encapsulated with a customized wrapper, itself written in SPATEL, to be properly integrated within the environment (orchestration engine). In contrast, BPEL is complete, open source and is an adopted standard with many supporting engines. Being easy to implement, scalable, and flexible, BPEL was quickly accepted by the

industrial community and is now the dominant technology in the field of Web service composition. BPEL supports all the constructs that any structural programming language contains (conditional statement, loop statement, ...) allowing the description of any complex composed service. Deployment and reuse of the composed service is versatile in BPEL.

For those reasons, and in order to achieve dynamic service composition, we have chosen BPEL as an orchestration language. Similar to several other orchestration languages having a corresponding number of engines, BPEL also has several corresponding engines. After selecting the BPEL as an orchestration language, the orchestration engine had to be chosen as well. We shortlisted few BPEL engine candidates like ActiveBPEL, Apache ODE, and Open ESB. We found Apache ODE to be more suitable for dynamic service composition due to its simplicity in deploying the composition script, the good quality of its available support on the Internet, and for its performance.

5.3 *SoCo* application design and implementation

After reviewing the main general requirements of any Mashup environment, we focus in the following on the *SoCo* as a Mashup creation environment augmented with dynamic services recommendation feature. we propose first an overview of the GUI. Then, we present the *SoCo* application general design. After that, we detail the technical implementation choices.

5.3.1 The *SoCo* GUI

First, we present here *SoCo* GUI which is the visible part of the Social Composer that interacts with the end-user. Figure 5.1 shows this interface includes two main parts. The service repository, exposes existing basic services in the form of a list. The user is able to drag/drop any service in the edition area where services could be linked and manipulated as black boxes. In fact, this operation (drag/drop) automatically transform the service to a box according to a service description, for instance WSDL. As the user is editing a new Mashup, and automatically when dropping a service into the edition area, dynamic suggestions are automatically provided according to the approach detailed in Chapter 3. Once the user has finished to edit his Mashup, he can actually run the created Mashup and publish it (by clicking on the ”Run Button”). This last operation feeds in parallel the user-service interactions repository in order to allow the recommendation system to compute relevant dynamic recommendations.

5.3.2 *SoCo* framework design

In this part we explicit the internal architecture of *SoCo*. Figure 5.2 shows the main components including the GUI, the service and user repositories, and the translator of the graphical user description into an executable script to be interpreted by an

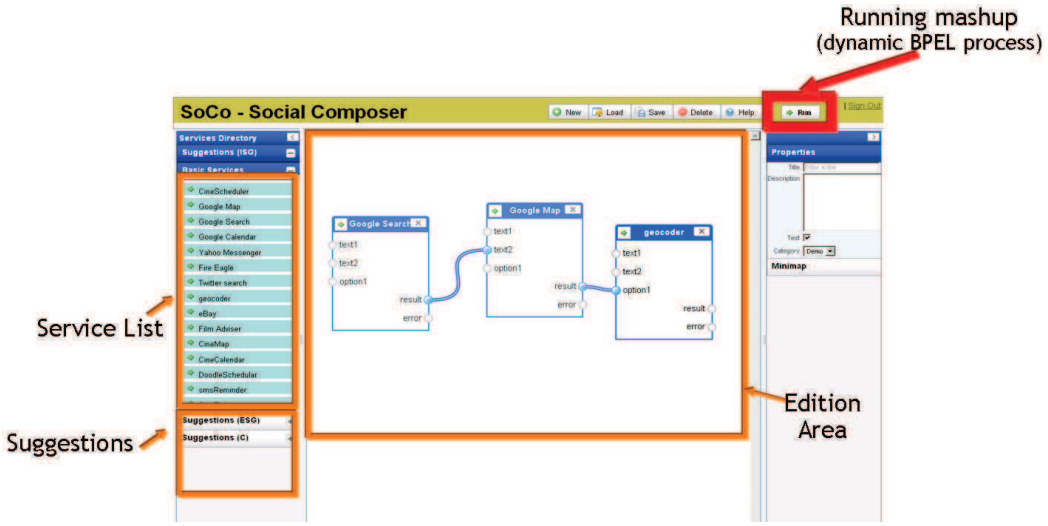

Figure 5.1: Social Composer Graphic User Interface

orchestration engine, itself deployed in an executable environment. In addition, interfaces between those components are described in the following. We will not detail the recommendation system in what is described further.

1. **Server repository** It contains all the basic services that a system could need to compose the services. This is an UDDI server that has the endpoints of all the services, which are registered.

2. **Service repository - GUI list** This is the interface for sending the service list from the Service repository to the GUI. The GUI can only get the list of service from the service repository. To register service a separate interface is used.

3. **Loading service description into the Mashup editor(edition area)** This module is responsible for showing the list in a panel or Mashup editor in a way that any service could be added to the editor as an abstract input/output box that is conforms to its description.

4. **GUI - Mashup BPEL Translator (MBT)** This is the first used module after the user sends the request for services composition by clicking on "Run" button. The request contains the information about the services that are to be interacted during the process execution and how they are connected to each other.

5. **Mashup BPEL Translator (MBT)** Based on the description sent by the previous module, the MBT generates mandatory files that translate the introduced description into an executable script (the BPEL script and resulting WSDL file) in order to deploy the process on the orchestration engine.

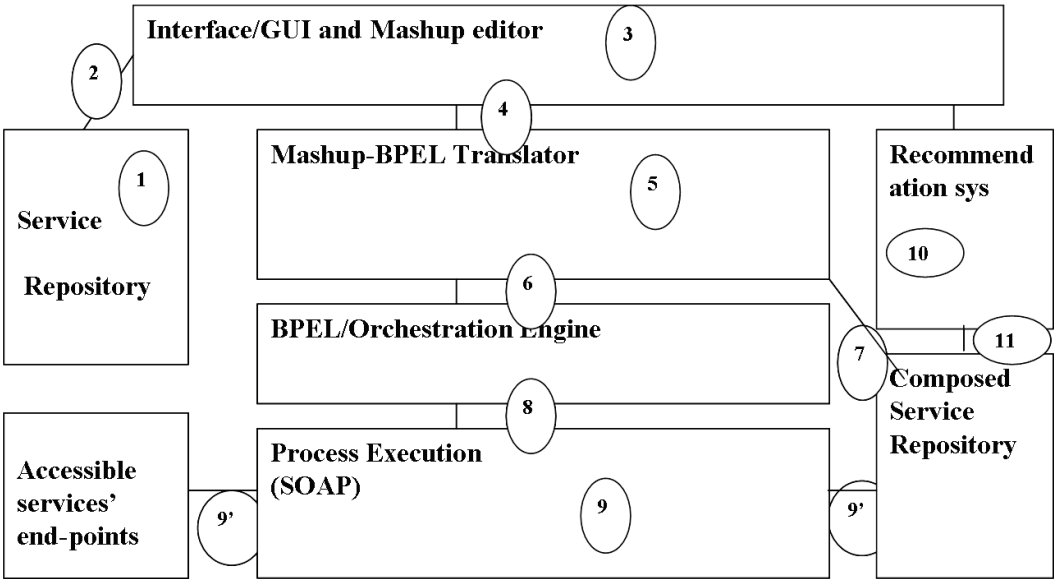

Figure 5.2: Social Composer Internal Architecture

6. **MBT - BPEL Engine** This is the deployment module, MBT sends information to BPEL engine in order to deploy and run the BPEL process.

7. **MBT - Composed Service Repository** This interface is responsible for publishing the created composed service by sending information to composed service repository; This information consists of a WSDL file including the composed service endpoint. The Composed Service Repository is also a UDDI server.

8. **BPEL Engine - Process Execution** This interface is responsible for sending a signal that the process has been deployed over the engine and is ready to receive any SOAP request.

9. **Process Execution** This module executes the process by sending the SOAP request to the process's endpoint. It also sends back the response of the execution (results).

5.3.3 SoCo implementation

This section explains the technical details of the system i.e., how the application/system is implemented and what choices are made during the implementation part. For more clarity, we run some examples of service composition.

Graphical User Interface (GUI) This Social Composer GUI is a Javascript-based Web client that uses Yahoo! User Interface Library (YUI). As shown in Figure

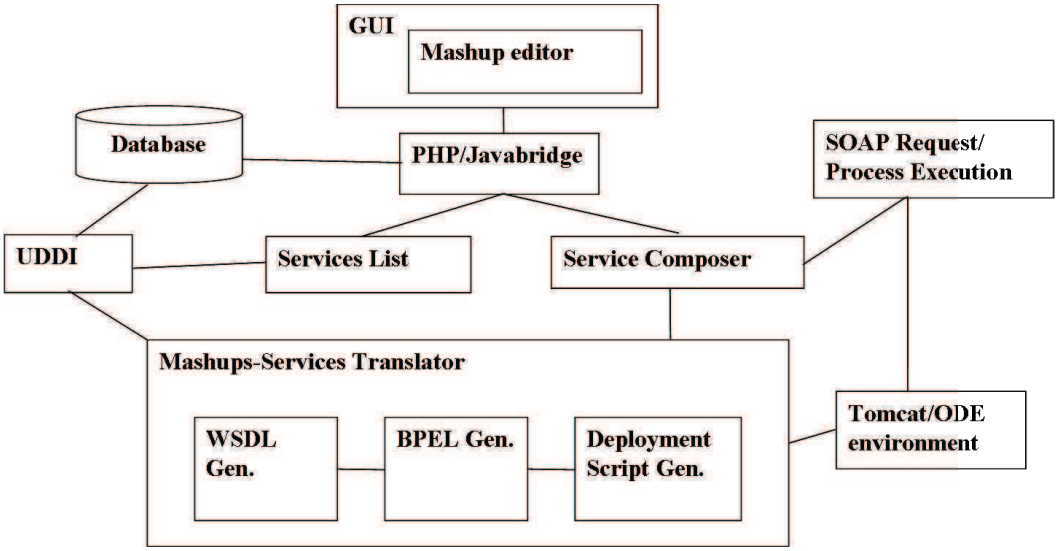

Figure 5.3: Social Composer implementation technical properties

5.1, SoCo's GUI is quite simple: it has a list of services that are divided into categories or sections e.g., these include basic services, composed services, recommended services, etc. The client side (JavaScript) uses JSON-RPC format to interact with other components of SoCO. More over, it uses the JSON String for describing composed service in the edition area. This includes listing, creating and linking the basic services.

Mashup Editor (edition area) It is a panel where the user can add, remove, or link the services to each other. Although there are many kinds of services abstract box templates that are provided by YUI library, we only use one kind of abstract box i.e., an input/output box with the possibility of filling inputs manually (text) or making links between service inputs and outputs according to the logic of composed services. The user can also save the Mashups he/she created into the panel and load the previously saved Mashups. The most important thing in the GUI is the Run button that actually does the work. By clicking on it, the user invokes the process of service composition mechanism. In order to compose a service, the user is required to drag and drop the services he wants to compose into the panel. If a user likes to get the weather information on his mobile phone by SMS, then he would have to drag the Weather service first, and then the SMS service(Figure 5.4).

PHP/Java Bridge An other important feature of SoCo is the ability to run the composed service. This part of the framework was developed by generating WSDL and BPEL files corresponding to the created service. Developed in Java and PHP, we have used an open source PHP/Java Bridge API to establish a channel to call the Java methods from PHP back-end. This bridge is responsible for getting the

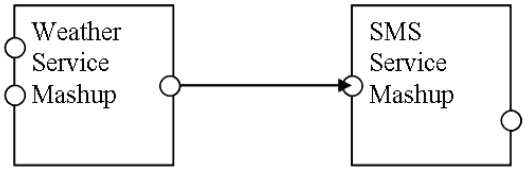

Figure 5.4: Simple example of a composed service schema

request for service lists and forwards it to the Java module i.e., Services List. The other task is to get the request for composing the services and sending it to the service composer module.

Services List This module receives the request for Services Lists in the UDDI server from PHP/Java Bridge and acts accordingly. It queries the UDDI server for the services that are SoCo related. In other terms, it queries the WSDL to built the abstract description of each service (input/output abstract box) and their deployment endpoint i.e., what operation to execute and what the relevant elements are. Based upon this information, it builds the JSON string. On receiving the JSON string, the client side script displays the services list accordingly.

UDDI Server This repository contains all the WSDL files of services that are in SoCo framework (internal services) or provided by external partners. If a user wants to compose services of any type, then the endpoints of each service are retrieved from the WSDL description in the UDDI server. The UDDI server that is being used in our system is Apache JUDDI, which is an open source UDDI server. It uses MySql as database. Note that SoCo framework includes one MySql server shared by the UDDI server and other components.

Service Composer This module receives the request from PHP/Java Bridge for services composition. It gets the JSON string describing the name and logic of the composed service introduced by the end user (as described above). This module is in charge of parsing the JSON string received from the bridge. During the parsing, it gets the services that have to be composed and the relation between those services i.e., what service is linked to which other service or which service to execute when another service execution is done. It sets out the required information/parameters that are to be used in the generation of WSDL and BPEL files. Once WSDL, BPEL files are created and deployed, the service composer invokes the composed service with the user's inputs.

Mashups-Services Translator This module is responsible for translating the user's request to a real composed service. For every composed service description built by the service composer based on user's composition logic description, the

Mashups-Services Translator launches three file generators because Apache ODE needs these files in order to execute the process i.e., WSDL, BPEL and deployment script.

WSDL Generator This module is responsible for generating the new composed service WSDL file. Besides generating the WSDL it also handles partner services WSDL files. This module downloads the WSDLs and corresponding XSDs of partner services locally. A modified WSDL4J API is used to parse the WSDLs. An important part in this module is to identify the XSDs related to WSDLs. This module also parses the XSD that is imported or embedded as <types> tags. It gets the required elements from XSD that are related to a particular <wsdl:message>, which is further related to the operation that a user wants this service to invoke. According to the JSON string received from the GUI, this model creates the new WSDL. It sets the inputs of the first service as the inputs of new WSDL, in addition to this, the new service's input also contains any other input of any service that is not wired/linked to the other service's output as shown in the example below. Any error/exception during the generation operation would stop the further flow and would return an error message to the user.

BPEL Generator After generating the WSDL in the previous module, the control comes to the BPEL generator. The previous module passes the required information to this module i.e., new WSDL definition and information about other services. The important information for this module is the links between the services i.e., how the services are connected to each other, their order and what terminal of one service is connected to the terminal of the other service as shown in Figure 5.5 ('return' is connected to 'text'). This module, also, uses a similar kind of APIs as WSDL4J, but it does not fully support the BPEL specification because it has been locally developed just for the purpose of this prototype.

Deployment The deployment of the BPEL process in Apache ODE is quite simple as compared to other BPEL engines. It needs a deployment script that has information about the partner services. BPEL generation module sends all the required information to this module after generating a BPEL process. Besides generating the deployment script for the process, this is also responsible for deploying all the files to the execution environment of the ODE as well.

Apache Tomcat/ODE Environment This is the environment where the BPEL process gets deployed and executed. The ODE environment is actually in the Tomcat running environment.

Process Execution The service created as a result of composing the different services is a Web service invokable through a SOAP request. At runtime, the composite service is supposed to be up and running. Triggered by *SoCo*, the Process

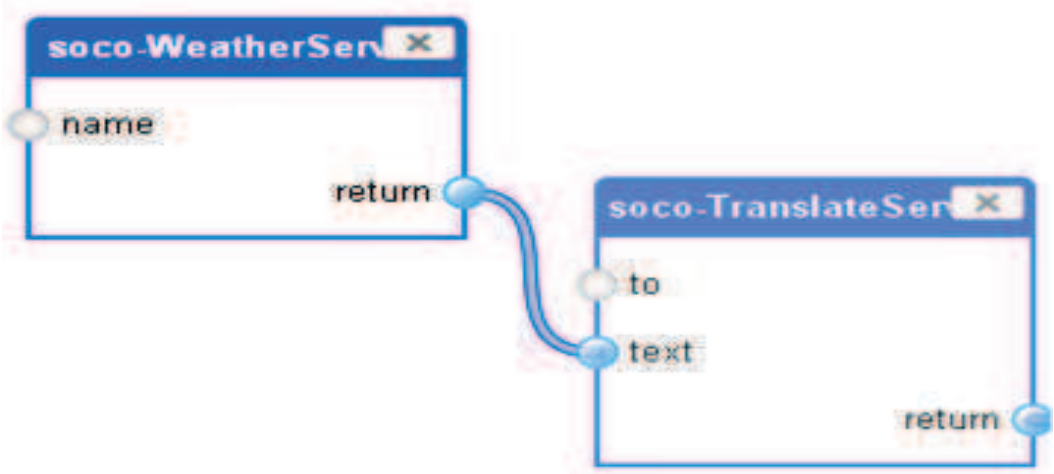

Figure 5.5: Simple example of SoCo composed service schema

Execution module is responsible for sending a SOAP request to the ODE engine that exposes the required process as another Web service. In order to send the SOAP request, another open source API is used: SAAJ (the SOAP with Attachments API for Java). The system gets the required input values from parsed JSON string (done by service composer) and sends the request to the server, on getting the result response in case the service is invoked successfully. Otherwise an error has occurred.

5.3.4 Illustrative class diagram and sequence diagram

Figure 5.6 presents the class diagram of important classes, whereas Figure 5.7 shows a sequence diagram of a main use case.

5.3.5 A running example

The example of a simple service composition is shown below. It includes Weather, Translator, SMS, and Email services. The sequence of the services is quite visible i.e., first Weather service would be called, then its results are translated through Translator service. After that SMS and Email services would be called to send the translated weather information to the filled in email address. Note that all the services have been simplified for the purpose of illustration.

Figure 5.9:

1. This is the request that is being sent to the process in a SOAP envelope with all possible input values from the user. This request is generated from the Process Execution module.

2. This is the response that the process receives from the Weather service.

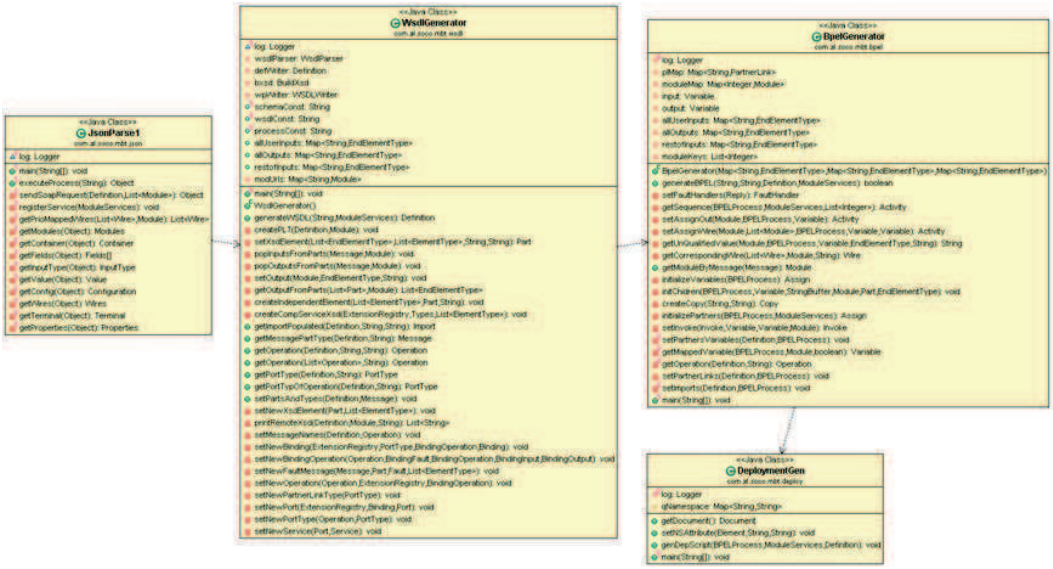

Figure 5.6: Class diagram of important classes

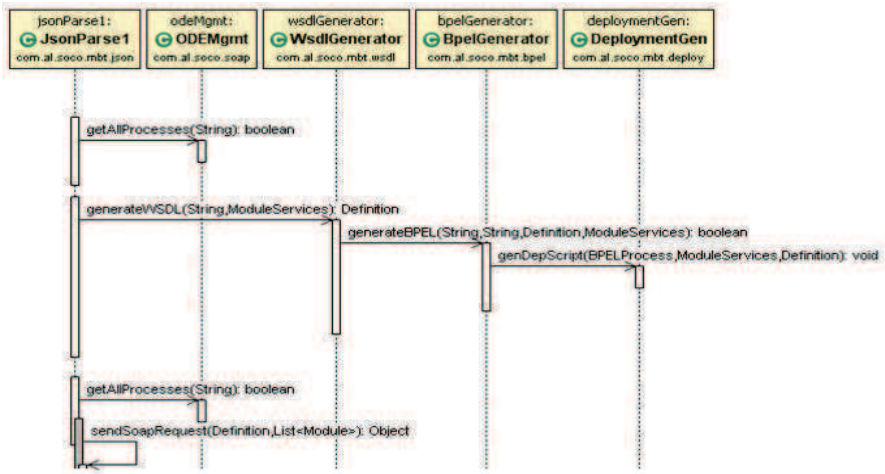

Figure 5.7: Sequence diagram of a main use case

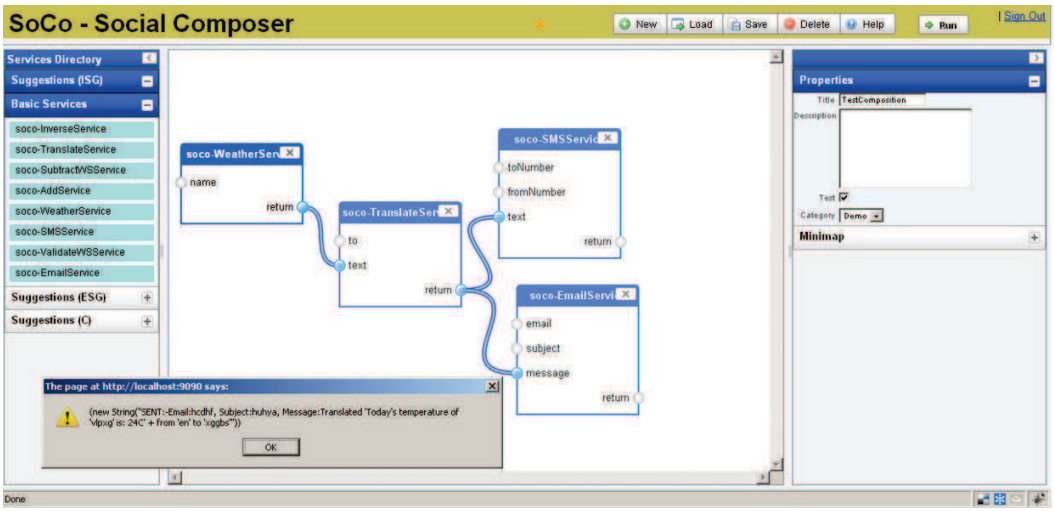

Figure 5.8: Simple example of SoCo composed service schema

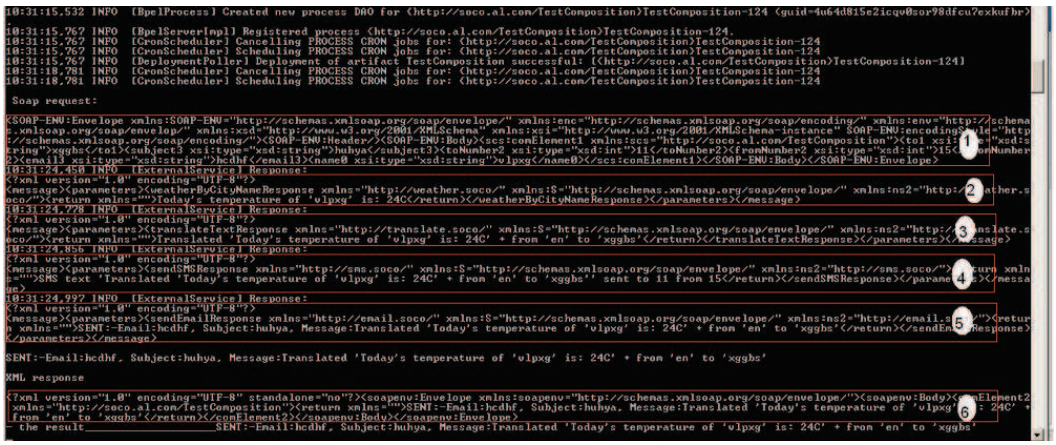

Figure 5.9: Simple example of SoCo composed service schema

3. This is the response from the Translator service.
4. Response from the SMS service and,
5. The response from the Email service.
6. This is the SOAP response of the request that was sent initially. This response is being received by the Process Execution module.

5.4 Conclusion

In order to implement the proposed algorithms and techniques, we developed a Mashup creation framework, called Social Composer (SoCo). This framework, dedicated to end users, initially implements the requirements established in the state of the art that any Mashup editor should provide in terms of level of abstraction and usability through the user interface. As well, it implements all the mechanisms needed to deploy a composed service starting from an abstract description entered by the user. We have selected BPEL as an orchestration language after comparing it with other existing language. In addition, *SoCo* provides capabilities to enrich the services directory by integrating external ones. Furthermore, SoCo has been augmented by including a dynamic recommendation functionality. This feature was notably demonstrated during the conference CSCW'2010. To measure the recommendation completion quality from the end-user perspective, an experiment study has to be conducted.

Chapter 6

Conclusion

Service composition shines out the need of making information systems more flexible and open. This concept has become the reference architecture model for applications carried by the impetus of the Internet (Web). Information systems are able to expose interfaces through the Web or Web services which has increased the number of available services on a daily basis. Furthermore driven by the Web, but this time by the Web 2.0, service composition has evolved to Web users characterized by their limited technical skills. Those end-users, named Y generation, participate, create, share, and comment content through the Web. This service composition evolution is incarnated in the Mashup concept and realized through Mashup editors such as Yahoo Pipes!. Thanks to the Mashup paradigm, service composition has been well established within the end users community enabling their creativity to flourish. For example, creating new applications without manually coding them constitutes a significant advancement. Web 2.0 has added its social dimension to the paradigm, allowing users to interact, either directly through the online social networks or indirectly through sharing, modifying content, or adding metadata.

In this specific context, this thesis is a directed effort towards the evolution of the service composition concept. The introduction of the social dimension within the process of composing services represents the main contribution of this thesis. The consideration of the social dimension is by itself an original path that hasn't been addressed in the literature before. Mainly, the concept of social dimension considers the activity of composing services (creating a Mashup) as a social activity. The exercise of this activity reveals social links between users based on their similarity in selecting and combining services. These links could be viewed as interesting dissemination means of expertise that is accumulated by users when composing services. In other terms, based on frequent composition patterns, and similarity between users, when a user is editing a Mashup, dynamic recommendations are proposed. Recommendations that aim at completing the initial part of a Mashup when it is being created.

The work in this thesis has launched a new direction of research and a unique view on the problem of end-users service composition through the introduction of a social dimension in the process of composition. The considered social dimension is closer to what is known in social network through the consideration of the user as the main actor of the system to socialize. The interest of this work is starting to be seen since several initiatives are continuing to flourish around this area [Maamar 2011, Zhang 2010].

Modeling, robustness, and response-time sensitivity were the main factors that

were taken into account during the development of the completion strategy. Interactive systems requirements dictated the response time restriction. The end result shows a reasonably acceptable response time within the context of large and complex datasets as reported in Chapter 5. Whereas a composite service is considered as a sequence of basic services, finding similarities between users in terms of service composition behavior comes first to find frequent patterns (subsequences). Compared to existing frequent pattern mining algorithms, the novelty of the proposed FESMA algorithm, is the use of an appropriate data structure to represent the frequent patterns. The FESMA algorithm meets the requirements of robustness and speed based on the FSTree data structure. As a result of the use of a tree-based data structure, the recommendation action is reduced to a simple traversal of a tree. The recommendation algorithm has shown significant improvements compared to the prior art. It particularly outperforms the work cited in [Greenshpan 2009] in term of response time.

Recommendation algorithm drawbacks such as cold start and newcomer services have also been studied. Possible solutions have been proposed to address both issues using the community approach combined with some diversification techniques.

Moreover, to implement the proposed algorithms and methods, we have developed a Mashup creation framework, called Social Composer (SoCo). This framework, dedicated to end users, initially implements the requirements established in the state of the art that any Mashup editor should provide in terms of level of abstraction and usability through the user interface. As well, it implements all the mechanisms needed to deploy a composed service starting from an abstract description entered by the user. In addition, it provides capabilities to enrich the services directory by integrating external ones. Furthermore, SoCo has been augmented by including a dynamic recommendation functionality. This feature was notably demonstrated during the conference CSCW'2010. To measure the recommendation completion quality from the end-user perspective, an experiment study has to be conducted.

During the elaboration of our approach, we were led to start from simplified definitions of the entities handled in order to formally constraint the model. One of these definitions is restricting the expressiveness of a composition of services to an ordered sequence of basic services. One future direction is to extend the proposed algorithms and methods to use a more complex service representation model. Actually, it would certainly be interesting to consider the activity of workflow creation, in an enterprise context, for defining business processes. In this case, a workflow is represented as a composite service. Our approach, when extended, would ease the creation of new workflows by letting the recommendation system dynamically suggest relevant completions. This will require mining of frequent complex patterns.

Another direction to explore in the field of service composition which is to introduce non-functional properties in the recommendation feature when composing services. These properties are mainly related to quality of service and service level agreement. The way these properties can influence the recommendation system is of particular interest.

Bibliography

[Adomavicius 2005] G. Adomavicius and A. Tuzhilin. *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. IEEE transactions on knowledge and data engineering, vol. 17, no. 6, pages 734–749, 2005. 51

[Agrawal 1993] R. Agrawal, T. Imieliński and A. Swami. *Mining association rules between sets of items in large databases*. ACM SIGMOD Record, vol. 22, no. 2, pages 207–216, 1993. 62

[Agrawal 1995] R. Agrawal and R. Srikant. *Mining sequential patterns*. pages 3 –14, mar. 1995. 62

[B. 2005] Ferenc B. *Trie-based APRIORI Implementation for Mining Frequent Item-sequences*. In Bart Goethals, Siegfried Nijssen and Mohammed J. Zaki, editeurs, ACM SIGKDD IW on OSDM'05, pages 56–65, Chicago, IL, USA, August 2005. 63, 71

[Balbiani 2006] P. Balbiani and F. Cheikh. *Computational analysis of interactiong Web services: a logical approach*, 2006. 3, 23, 24

[Bond 2006] G.W. Bond. *An introduction to ECharts: The concise user manual*. Transition, vol. 4, page 2, 2006. 16

[Bosca 2005] A. Bosca, A. Ferrato, F. Corno, I. Congiu and G. Valetto. *Composing Web services on the basis of natural language requests*. In Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on, pages 2 vol. (xxxiii+856), 2005. 30

[Box 2000] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte and D. Winder. *Simple object access protocol*. URL: http://www. w3. org/TR/SOAP/, 2000. 17

[Broy 1991] Manfred Broy. *Towards a Formal Foundation of the Specification and Description Language SDL*. Formal Aspects of Computing, vol. 3, pages 21–57, 1991. 16

[Chaudhuri 2009] S. Chaudhuri and R. Kaushik. *Extending autocompletion to tolerate errors*. In SIGMOD '09, pages 707–718, New York, NY, USA, 2009. ACM. 62

[Chen 2003] L. Chen, N.R. Shadbolt, C. Goble, F. Tao, S.J. Cox, C. Puleston and PR Smart. *Towards a knowledge-based approach to semantic service composition*. Lecture Notes in Computer Science, pages 319–334, 2003. 26

[Christensen 2001] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana. *Web services description language (WSDL) 1.1*, 2001. 17

[Clarke 1999] Edmund M.(Jr.) Clarke, O. Grumberg and D.A. Peled. Model checking. MIT, 1999. 15

[Curbera 2002] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana. *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI*. Internet Computing, IEEE, vol. 6, no. 2, pages 86 –93, 2002. 17

[Davison 1998] B.D. Davison and H. Hirsh. *Predicting sequences of user actions*. In AAAI/ICML 1998 Workshop on PF AI ATSA, 1998. 62

[Díaz 2007] O. Díaz, S. Pérez and I. Paz. *Providing personalized mashups within the context of existing web applications*. Web Information Systems Engineering–WISE 2007, pages 493–502, 2007. 27

[DiBernardo 2008] M. DiBernardo, R. Pottinger and M. Wilkinson. *Semi-automatic web service composition for the life sciences using the BioMoby semantic web framework*. Journal of Biomedical Informatics, vol. 41, no. 5, pages 837–847, 2008. 26

[Dimitrov 2007] M. Dimitrov, A. Simov, V. Momtchev and M. Konstantinov. *WSMO Studio–A Semantic Web Services Modelling Environment for WSMO*. The Semantic Web: Research and Applications, pages 749–758, 2007. 22

[Dong 2007] G. Dong and J. Pei. Sequence data mining. Springer-Verlag New York Inc, 2007. 62

[Elenius 2005] D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati and R. Senanayake. *The OWL-S editor–a development tool for semantic web services*. The Semantic Web: Research and Applications, pages 78–92, 2005. 21

[Ennals 2007a] Rob Ennals and David Gay. *User-friendly functional programming for web mashups*. SIGPLAN Not., vol. 42, no. 9, pages 223–234, 2007. 4, 27, 37

[Ennals 2007b] Robert Ennals and Minos N. Garofalakis. *MashMaker: mashups for the masses*. In SIGMOD Conference, pages 1116–1118, 2007. 27

[Erl 2005] Thomas Erl. Service-oriented architecture: Concepts, technology, and design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. 9

[Erlewine] M.Y. Erlewine. *Ubiquity: Designing a Multilingual Natural Language Interface*. In Proceedings of the SIGIR 2009 Workshop on Information Access in a Multilingual World, Boston, July 23, 2009, pp 45-48. 32, 46

[Fernandez 1997] J.C. Fernandez, C. Jard, T. Jéron and C. Viho. *An experiment in automatic generation of test suites for protocols with verification technology* 1.* Science of Computer Programming, vol. 29, no. 1-2, pages 123–146, 1997. 16

[Filipowska 2007] A. Filipowska, A. Haller, M. Kaczmarek, T. van Lessen, J. Nitzsche and B. Norton. *Process ontology language and operational semantics for semantic business processes.* Deliverable 1.3. Project IST 026850 SUPER, 2007. 22

[Floyd 2007] Ingbert R. Floyd, M. Cameron Jones, Dinesh Rathi and Michael B. Twidale. *Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software.* In HICSS '07, page 86, Washington, DC, USA, 2007. IEEE Computer Society. 34

[Goethals 2003] B. Goethals and M.J. Zaki. *FIMI03: Workshop on frequent itemset mining implementations.* In Third IEEE ICDM FIMI Workshop, pages 1–13. Citeseer, 2003. 59, 63

[Gordon 2000] M. Gordon. *From LCF to HOL: a short history.* Proof, language, and interaction: essays in honour of Robin Milner, pages 169–185, 2000. 15

[Grammel 2008] L. Grammel and M.A. Storey. *An End User Perspective on Mashup Makers.* University of Victoria, Tech. Rep. DCS-324-IR, September 2008. 4, 33

[Greenshpan 2009] Ohad Greenshpan, Tova Milo and Neoklis Polyzotis. *Autocompletion for mashups.* Proc. VLDB Endow., vol. 2, no. 1, pages 538–549, 2009. 23, 37, 46, 59, 62, 74, 75, 90

[Guillaume 2004] J.L. Guillaume and M. Latapy. *Bipartite structure of all complex networks.* Information processing letters, vol. 90, no. 5, pages 215–221, 2004. 46, 68

[Han 2000] Jiawei Han, Jian Pei and Yiwen Yin. *Mining frequent patterns without candidate generation.* SIGMOD Rec., vol. 29, no. 2, pages 1–12, 2000. 63

[Hassine 2006] A.B. Hassine, S. Matsubara and T. Ishida. *A Constraint-Based Approach to Horizontal Web Service Composition.* Proc. ISWC, Athens, GA, USA, 2006. 22

[Hoyer 2008] V. Hoyer and M. Fischer. *Market Overview of Enterprise Mashup Tools.* In Proceedings of the 6th International Conference on Service-Oriented Computing, pages 708–721. Springer, 2008. 33

[Ilyas 2008] Ihab F. Ilyas, George Beskales and Mohamed A. Soliman. *A survey of top-k query processing techniques in relational database systems.* ACM Comput. Surv., vol. 40, no. 4, pages 1–58, 2008. 49

[Jones 2009] M. Cameron Jones and Elizabeth F. Churchill. *Conversations in developer communities: a preliminary analysis of the yahoo! pipes community.* In CCT '09, pages 195–204, New York, NY, USA, 2009. ACM. 34

[Kirda 2001] E. Kirda. *Engineering of Web services with XML and XSL.* In Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, pages 318–319. ACM, 2001. 8

[Koop 2008] D. Koop. *Viscomplete: Automating suggestions for visualization pipelines.* | IEEE Transactions on Visualization and Computer Graphics, pages 1691–1698, 2008. 46

[Kotler 1984] P. Kotler and R.E. Turner. *Marketing management: Analysis, planning, and control.* 1984. 7

[Lausen 2005] H. Lausen, J. de Bruijn, A. Polleres and D. Fensel. *Wsml-a language framework for semantic web services.* In W3C Rules Workshop, Washington DC, USA, 2005. 21

[Law 2007] T. Law. *Social Scripting for the Web.* Computer, vol. 40, no. 6, pages 96–98, 2007. 25

[Lécué 2008] F. Lécué, E. Silva and L.F. Pires. *A framework for dynamic web services composition.* Emerging Web Services Technology, Volume II, pages 59–75, 2008. 24

[Leskovec 2007] Jure Leskovec, Jon Kleinberg and Christos Faloutsos. *Graph evolution: Densification and shrinking diameters.* ACM Trans. Knowl. Discov. Data, vol. 1, no. 1, page 2, 2007. 49

[Liu 2007] X. Liu, Y. Hui, W. Sun and H. Liang. *Towards service composition based on mashup.* In Services, 2007 IEEE Congress on, pages 332–339. IEEE, 2007. 4, 25

[Lord 2005] P. Lord, P. Alper, C. Wroe and C. Goble. *Feta: A light-weight architecture for user oriented semantic service discovery.* The Semantic Web: Research and Applications, pages 17–31, 2005. 25

[Maamar 2007] Z. Maamar, M. Lahkim, D. Benslimane, P. Thiran and S. Sattanathan. *Web Services Communities-Concepts & Operations.* In Proceedings of The 3rd International Conference on Web Information Systems and Technologies (WEBIST'2007), Barcelona, Spain, 2007. 23

[Maamar 2009] Z. Maamar, L.K. Wives, Y. Badr and S. Elnaffar. *Even Web Services Can Socialize: A New Service-Oriented Social Networking Model.* In INCOS'09, pages 24–30, 2009. 35

[Maamar 2011] Z. Maamar, H. Hacid and M.N. Huhns. *Why Web Services Need Social Networks.* Internet Computing, IEEE, vol. 15, no. 2, pages 90–94, 2011. 89

[Maaradji 2010a] Abderrahmane Maaradji, Hakim Hacid, Johann Daigremont and Noël Crespi. *Social Composer: A Social-Aware Mashup Creation Environment.* In ACM CSCW '10 (Demos session), 2010. 54

[Maaradji 2010b] Abderrahmane Maaradji, Hakim Hacid, Johann Daigremont and Noël Crespi. *Towards a Social Network Based Approach for Services Composition.* In IEEE ICC'10., may 2010. 45, 67

[Martin 2004] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne*et al. OWL-S: Semantic markup for web services.* W3C Member Submission, vol. 22, pages 2007–04, 2004. 20

[McCreight 1976] Edward M. McCreight. *A Space-Economical Suffix Tree Construction Algorithm.* J. ACM, vol. 23, no. 2, pages 262–272, 1976. 62

[Milanovic 2004] N. Milanovic and M. Malek. *Current solutions for Web service composition.* Internet Computing, IEEE, vol. 8, no. 6, pages 51 – 59, 2004. 20, 24

[Nielsen 2006] J. Nielsen. *Participation inequality: Encouraging more users to contribute.* Jakob Nielsens Alertbox, vol. 9, page 2006, 2006. 43

[P. 2001] Jian P. and al. *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth.* Data Engineering, International Conference on, vol. 0, page 0215, 2001. 62, 63

[Palfrey 2010] J. Palfrey and U. Gasser. Born digital: Understanding the first generation of digital natives. Basic Books, 2010. 3

[Paolucci 2004] M. Paolucci, N. Srinivasan and K. Sycara. *Expressing wsmo mediators in owl-s.* In Proceedings of the workshop on Semantic Web Services: Preparing to Meet the World of Business Applications held at the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan. Citeseer, 2004. 22

[Papazoglou 2003] Mike P. Papazoglou. *Service -Oriented Computing: Concepts, Characteristics and Directions.* Web Information Systems Engineering, International Conference on, vol. 0, page 3, 2003. 8, 11

[Peltz 2003] C. Peltz. *Web services orchestration and choreography.* Computer, vol. 36, no. 10, pages 46 – 52, 2003. 14

[Pistore 2004] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau and P. Traverso. *Planning and monitoring web service composition.* Artificial Intelligence: Methodology, Systems, and Applications, pages 106–115, 2004. 14

[Roman 2005] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler and D. Fensel. *Web service modeling ontology.* Applied Ontology, vol. 1, no. 1, pages 77–106, 2005. 21

[Scicluna 2004] J. Scicluna, C. Abela and M. Montebello. *Visual modeling of owl-s services.* In Proceedings of the IADIS International Conference WWW/Internet. Citeseer, 2004. 21

[Shiaa 2008] M. Shiaa, P. Falcarin, A. Pastor, F. Lécué, EM Goncalves da Silva, F. Pires*et al. Towards the automation of the service composition process: case study and prototype implementations.* Proc. of the ICT Mobile and Wireless Communications Summit, Stockholm, Sweden, 2008. 31, 46

[Shneiderman 1984] Ben Shneiderman. *Response time and display rate in human performance with computers.* ACM Comput. Surv., vol. 16, pages 265–285, September 1984. 39

[Soriano 2007] J. Soriano, D. Lizcano, M.A. Cañas, M. Reyes and J.J. Hierro. *Fostering innovation in a mashup-oriented enterprise 2.0 collaboration environment.* UK, sai: sisn, vol. 24, pages 62–68, 2007. 27

[Soriano 2008] Javier Soriano, David Lizcano, Juan J. Hierro, Marcos Reyes, Christoph Schroth and Till Janner. *Enhancing User-Service Interaction through a Global User-Centric Approach to SOA.* In ICNS '08, pages 194–203, Washington, DC, USA, 2008. IEEE Computer Society. 35

[Spohrer 2007] Jim Spohrer, Paul P. Maglio, John Bailey and Daniel Gruhl. *Steps Toward a Science of Service Systems.* Computer, vol. 40, pages 71–77, 2007. 7

[Srikant 1996] R. Srikant and R. Agrawal. *Mining sequential patterns: Generalizations and performance improvements.* EDBT'96, pages 1–17, 1996. 62

[ter Beek 2007] Maurice H. ter Beek, Antonio Bucchiarone and Stefania Gnesi. *Web Service Composition Approaches: From Industrial Standards to Formal Methods.* In ICIW, pages 15–20, 2007. 3

[Turner 2005] K. Turner. *Formalising web services.* Formal Techniques for Networked and Distributed Systems-FORTE 2005, pages 473–488, 2005. 19

[Wasserman 1994] S. Wasserman and K. Faust. Social network analysis: Methods and applications. Cambridge Univ Pr, 1994. 35

[Wong 2007] J. Wong and J.I. Hong. *Making mashups with marmite: towards end-user programming for the web.* In Proceedings of the SIGCHI conference on Human factors in computing systems, pages 1435–1444. ACM, 2007. 27, 29

[Yelmo 2008] J.C. Yelmo, J.M. del Alamo, R. Trapero, P. Falcarm, Jian Yi, B. Cairo and C. Baladron'. *A user-centric service creation approach for Next Generation Networks.* pages 211–218, May 2008. 11

[Yu 2007] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel and M. Matera. *A framework for rapid integration of presentation components.* In Proceedings of the 16th international conference on World Wide Web, pages 923–932. ACM, 2007. 27

[Yu 2008] J. Yu, B. Benatallah, F. Casati and F. Daniel. *Understanding Mashup Development.* IEEE Internet Computing, vol. 12, no. 5, pages 44–52, 2008. 4, 32

[Yu 2009a] Cong Yu, Laks V. S. Lakshmanan and Sihem Amer-Yahia. *Recommendation Diversification Using Explanations.* In ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, pages 1299–1302, Washington, DC, USA, 2009. IEEE Computer Society. 53

[Yu 2009b] Shuli Yu and C. Jason Woodard. *Innovation in the Programmable Web: Characterizing the Mashup Ecosystem.* pages 136–147, 2009. 27, 35, 56, 70

[Yuan 2007] Y. Yuan, J.J. Wen, W. Li and B.B. Zhang. *A Comparison of Three Programming Models for Telecom Service Composition.* In AICT. IEEE Computer Society Washington, DC, USA, 2007. 3, 11

[Zaki 2000] M.J. Zaki*et al. Scalable algorithms for association mining.* IEEE TKDE, vol. 12, no. 3, pages 372–390, 2000. 62

[Zaki 2001] M.J. Zaki. *SPADE: An efficient algorithm for mining frequent sequences.* Machine Learning, vol. 42, no. 1, pages 31–60, 2001. 62

[Zeithaml 1996] V.A. Zeithaml, M.J. Bitner and D.D. Gremler. Services marketing. McGraw-Hill New York, 1996. 7

[Zhang 2003] R. Zhang, I.B. Arpinar and B. Aleman-Meza. *Automatic composition of semantic web services.* In 1st International Conference on Web Services, pages 38–41. Citeseer, 2003. 24

[Zhang 2010] H. Zhang, Z. Zhao, S. Sivasothy, C. Huang and N. Crespi. *Quality-assured and sociality-enriched multimedia mobile mashup.* EURASIP Journal on Wireless Communications and Networking, vol. 2010, page 11, 2010. 89

[Zhao 2006] X. Zhao, H. Yang and Z. Qiu. *Towards the formal model and verification of web service choreography description language.* Proc. of WS-FM 2006, 2006. 16

[Zhovtobryukh 2006] D. Zhovtobryukh. Context-aware web service composition. University of Jyvaskyla, 2006. 24

RESUME DU RAPPORT DE THESE

Le paradigme de service dans les nouvelles technologies de l'information et de communication est omniprésent, si bien qu'on parle de science des services. Le W3C définit le service Web comme un système logiciel qui permet l'interaction entre machines sur le réseau à travers des interfaces. Les services Web sont définis dans le cadre des architectures orientées services (SOA) qui permet de distinguer le fournisseur de service, le répertoire de services, et enfin le consommateur du service (le client). Cette distinction a donné des opportunités à opérer des compositions de services qui consistent à créer de nouveaux services en réutilisant des services déjà existants. Cependant, la composition de services est principalement bénéfique aux utilisateurs expérimentés comme les développeurs de logiciels car elle requiert un niveau technique élevé. Par opposition, la tendance actuelle traduite par l'émergence du Web2.0, vise à permettre aux utilisateurs du Web de créer leurs propres services à travers les environnements de Mashup, ou de collaborer et de capitaliser des connaissances à travers les réseaux et les médias sociaux. Nous croyons qu'il existe un grand potentiel pour "démocratiser" la composition de services dans de tels contextes. L'émergence du Web 2.0, exprimée dans les paradigmes qui le définissent tels que le contenu généré par l'utilisateur (UGC, Mashups) et le web social, constitue, de notre point de vue, une opportunité intéressante pour améliorer la productivité de services de l'utilisateur final et accélérer son processus créatif en capitalisant les connaissances générées par tous les utilisateurs.

Dans ce contexte, cette thèse vise à soutenir l'évolution du concept de composition de services par le biais de contributions significatives. La principale contribution de cette thèse est en effet l'introduction de la dimension sociale dans le processus de construction d'un service composite à travers les environnements dédiés aux utilisateurs finaux. Ce concept de la dimension sociale considère l'activité de composition de services (création d'un Mashup) comme une activité sociale. Cette activité révèle les liens sociaux entre les utilisateurs en fonction de leur similitude dans le choix et la combinaison des services. Ces liens sont un moyen intéressant de diffusion d'expertise de composition de services. En d'autres termes, sur la base des schémas fréquents de composition, et la similitude entre les utilisateurs, quand un utilisateur est en train d'éditer un Mashup, des recommandations dynamiques sont proposées. Ces recommandations visent à compléter la première partie de Mashup

déjà mis en place par l'utilisateur. Ce concept a été exploré à travers (i) la complétion de Mashup étape par étape en recommandant à chaque étape un service unique, et (ii) la complétion totale de Mashup en recommandant la séquence complète de services qui pourraient le compléter.

Au-delà de l'introduction de la dimension sociale dans le processus de composition de services, cette thèse a adressé une contrainte particulière du système de recommandation liée aux exigences des systèmes interactifs en termes de temps de réponse. À cet égard, nous avons développé des algorithmes robustes et adaptées aux spécificités de notre problème. Alors qu'un service composite est considéré comme une séquence de service, la recherche de similarités entre les utilisateurs revient d'abord à trouver des modèles fréquents (séquences), puis de les représenter dans une structure de données avantageuse pour l'algorithme de recommandation. L'algorithme proposé FESMA répond à ces exigences en se basant sur la structure FSTREE et offrant des résultats intéressants par rapport à l'art antérieur.

Enfin, pour mettre en œuvre les algorithmes et les méthodes proposées, nous avons développé un environnement de création de Mashup, appelé '*Social Composer*' (SoCo). Cet environnement, dédié aux utilisateurs finaux, respecte les critères d'utilisation en se basant sur le workflow graphique. En outre, il met en œuvre tous les mécanismes nécessaires pour déployer le service composé à partir d'une description abstraite introduite par l'utilisateur. De plus, SoCo a été augmentée en y incluant la fonctionnalité de recommandation dynamique, démontrant la faisabilité de ce concept.

1. La composition semi-automatique de services et le Web 2.0

La composition de services consiste à créer des services à valeurs ajoutées en réutilisant des services existants. Elle constitue un sujet de recherche qui a été largement étudié aussi bien du point vue académique qu'industriel. En créant des services par composition, il est possible de réduire les couts de développement, la durée de production et répondre de cette façon à la demande croissante des applications. Initialement, les techniques de composition de services sont employées par les utilisateurs expérimentés comme les développeurs. Il existe trois approches de composition de services : la composition manuelle, semi-automatique ou automatique.

La première approche reste à portée limitée car elle requiert un niveau technique très élevé. La deuxième approche permet de construire automatiquement des services composés en réponse à une requête donnée. Cette requête peut correspondre à une demande explicite de l'utilisateur ou construite à partir de données traitées automatiquement (contexte de l'utilisateur). Cependant, cette approche trouve ses limites dans les problèmes d'indécision dus aux requêtes incomplètes. La dernière approche, qui est la composition semi-automatique, a pour but de fournir à l'utilisateur final un environnement de création de services composés. Cet environnement offre un support pour l'utilisateur par l'automatisation de certaines parties de la composition. Dans notre travail, nous nous intéressons en particulier à ce type de composition. Ainsi, l'approche semi-automatique a l'avantage principal de faire participer l'utilisateur au processus de composition en exploitant, par exemple, les informations générées par l'utilisateur ou de sa communauté.

La composition semi-automatique de services Web a évolué au fil du temps allant de simples outils graphiques pour arriver jusqu'à des outils sémantiques sophistiqués. Une étape importante dans cette évolution, qui s'inscrit dans le cadre du Web 2.0, est ce qui est aujourd'hui communément appelé les Mashups. Cela a contribué à l'émergence d'une multitude de méthodes pour la composition semi-automatique de services Web que nous avons classifiés en trois grandes catégories :

Composition orientée utilisateur final : Cette approche vise à construire un profil de l'utilisateur afin d'assister le processus de composition, en particulier dans les situations d'arbitrage du processus de sélection de services de base. Cette approche vise aussi à fournir des outils et des interfaces pour faciliter la composition des applications (des outils basés sur BPEL, Mashups, etc.).

Composition orientée communautés : Cette approche vise à considérer les connaissances produites dans une communauté, un domaine spécifique ou dans une entreprise. L'extraction des connaissances du domaine ou de la communauté permet la définition d'un ensemble de règles et de construire un système de recommandation dans le processus de composition.

La composition orientée réseaux : Nous avons identifié une troisième approche qui est basée sur les réseaux sociaux. Un réseau social ne peut être pris en compte dans l'approche communautaire (le paragraphe ci-dessus), car il décrit une structure relationnelle spécifique.

La différence majeure est que la communauté désigne un regroupement d'individus autour d'un thème d'intérêt commun, générant des communautés spécialisées dans des domaines particuliers (ce qui justifie cette approche), alors que le réseau social décrit les réseaux d'"amis" construit sur la base d'intérêts spécifiques pour chaque relation dans le réseau. Par conséquent, les connaissances générées dans un réseau social ne peuvent pas être traitées par les mêmes méthodes que celles des communautés.

Durant les quatre dernières années, le Web2.0, nourri par des plateformes sociales (Myspace, Facebook, etc.), est devenu une source intéressante de connaissance pour la communauté de recherche de la composition de services. Ainsi, plusieurs travaux ont été lancés autour de cette opportunité à exploiter la connaissance de la masse afin d'améliorer le processus de composition en examinant les environnements collaboratifs. Plusieurs acteurs majeurs du Web offrent des environnements en ligne pour la création Mashups tels que : Microsoft Popfly, Yahoo pipes, et Open Mashups Studio. Les Mashups représentent aussi un nouveau paradigme pour la composition de services. Il est décrit dans la littérature les aspects techniques des environnements de création de Mashup du point de vue des fournisseurs de services. Aussi , il est décrit les approches et les points à considérer pour un système de création de Mashup. Il faut souligner aussi la nécessité de considérer les aspects internes (modèles de données) et externes (présentations) pour l'intégration des données.

Du point de vue de l'utilisateur, un Mashup est défini comme une combinaison de données et fonctionnalité Web construit par l'utilisateur final. Cette analyse souligne aussi la nécessité considérer les communautés dans la spécification des environnements de création de Mashups. Après une analyse détaillée de ces propriétés, il est intéressant de noter qu'aucun environnement actuel de création de Mashups n'offre une dimension de réseau social dans le processus de composition. Le travail que nous présentons dans la section suivante tente de répondre à ce besoin en utilisant les réseaux sociaux afin de faciliter la composition de services. Notre objectif à ce stade est de montrer quelles sont les informations qui peuvent être extraites à partir de réseaux sociaux afin d'être exploitable dans le processus de composition.

2. Vers la composition orientée réseaux sociaux

La problématique à laquelle nous essayons de répondre est : comment tirer profit des interactions sociales pour faciliter la création de

services composés? . En vue des éléments présentés dans la section 2, et de la problématique posée ci-dessus, nous proposons une approche, concrétisée par un Framework, pour la composition de services basée sur les connaissances extraites des réseaux sociaux. Pour notre approche, un réseau social que nous considérons à ce stade est une structure implicite construite à partir des intérêts pour les services qu'une paire d'utilisateurs pourraient avoir en commun. Notre Framework est appelé SoCo (pour Social Composer) et sera introduit dans le reste de cet article. La Figure **Erreur ! Source du renvoi introuvable.** illustre l'architecture générale du Framework SoCo. Afin d'aider l'utilisateur dans le processus de composition, SoCo comprend deux volets principaux : (i) l'extraction et modélisation de connaissances sociales et (ii) un système de recommandations dynamique de services.

2.1 Extraction et modélisation de la connaissance sociale

D'une manière générale, nous considérons deux approches différentes pour construire des relations sociales dans ce contexte : explicite ou implicite. Pour le cas explicite, l'utilisateur se voit offrir la possibilité de déclarer, par lui-même, une relation avec un autre utilisateur spécifique (à l'image des réseaux sociaux sur le Web). Pour le cas implicite, une relation sociale est déduite en fonction des activités des différents utilisateurs. Par exemple une relation particulière est déduite si une personne utilise un grand nombre des services créés par une autre personne. Dans les deux cas, explicite et implicite, nous construisons un graphe reliant les utilisateurs en fonction de leur intérêt traduit par leurs activités de composition.

Concrètement dans notre Framework, les phases d'extraction et d'analyse fournissent en sortie deux types de données : (i) le profil de l'utilisateur qui contient des informations qui décrivent les intérêts particuliers et les préférences, l'historique de ses interactions avec le système. Typiquement, il s'agit de statistiques sur l'utilisation des services (consommation et composition). Par exemple, le nombre de services utilisés dans la composition et les schémas de composition des services créés qui nous permettent d'apprendre d'avantage sur l'expertise d'une personne donnée dans un domaine particulier de services, et donc la pertinence des services utilisés par cette personne. (ii) La description des liens qui définissent le réseau social lui-même. Ces liens sont utilisés pour calculer la proximité sociale entre deux personnes selon un contexte particulier. Ce deuxième type d'information nous permet de calculer, par exemple, la recommandation d'un service par

rapport à la confiance entre deux individus et leurs intérêts communs respectifs.

2.2 Recommandation dynamique de services

L'idée principale de cette phase est de capitaliser sur l'information et la connaissance calculée dans la phase d'analyse. Il s'agit de la modélisation du réseau social pour construire un système de recommandation dynamique de services qui sera intégré dans un environnement global pour la composition de services. Ce système de recommandation vise à aider l'utilisateur du Framework SoCo lors de la création d'un service en proposant des services pour la composition en fonction de l'état actuel du processus de composition de services. Ainsi, ce système de recommandation intervient dans le processus de composition en sélectionnant les services les plus pertinents en fonction d'un schéma de composition donné. Plus concrètement, quand il s'agit de la création d'un service, c'est à dire un service composé, dans l'environnement de création de services SoCo, un utilisateur est généralement indécis sur le choix d'un service successeur d'un service donné dans le diagramme de composition. Dans cette situation, le système de recommandations proposera une liste de services ordonnée sur la base des informations fournies par l'analyse du réseau social. Ainsi, la mesure de l'importance de la recommandation d'un service donné est proportionnelle à son utilisation antérieure et à la proximité sociale vis-à-vis des membres qui l'ont utilisé.

3. Applications à la complétion des Mashup

Dans ce qui suit, nous abordons le problème de la complétion totale des Mashups qui consiste à prédire l'ensemble le plus pertinent de combinaison des services en fonction d'une composition de services initialement fourni par l'utilisateur final. Le Mashup résultant ainsi composé devrait réponde au mieux à l'objectif de l'utilisateur final. Nous modélisons le problème de la complétion totale des Mashups comme un problème d'analyse des séquences fréquentes et nous montrons comment les algorithmes existants peuvent être appliqués dans ce contexte. Pour surmonter certaines limitations des algorithmes fréquentes minières séquence, par exemple, l'efficacité et la granularité recommandation, nous proposons FESMA, un algorithme nouveau et efficace pour le calcul de séquences fréquentes de services et de recommander des finitions. FESMA intègre également une dimension sociale, extrait de la transformation des interactions utilisateur-service en interactions utilisateur-utilisateur, la construction d'un graphe

implicite qui permet de mieux prédire les achèvements de services d'une manière adaptée à des utilisateurs individuels.

3.1 Description de l'approche proposée

La composition Mashup semi-automatique aidera à créer des Mashups d'une manière plus rapide et facile à utiliser. En général, deux mécanismes de complétion de Machups peuvent être distinguées: (i) la complétion étape par étape, où une liste de services uniques est proposée à l'utilisateur sur la base du service actuellement sélectionné, ainsi de proche en proche l'utilisateur pourra construire un Mashup complet, et (ii) la complétion totale, où un liste de schémas de composition est recommandé à l'utilisateur, chaque schéma recommandé est susceptible de compléter la partie déjà introduite par l'utilisateur. Dans cette section nous détaillerons en particulier la complétion totale des Mashups. L'approche proposée vise à fournir une réponse à ce problème grâce à l'extraction de séquence fréquentes, leur modélisation, et la suggestion de la combinaison la plus intéressante de services qui devraient suivre une séquence de services. Les principaux challenges de cette approche est d'améliorer le temps de création Mashups et leur qualité tout en faisant face aux défis suivants:

- L'évolutivité: le nombre de candidats potentiels à une complétion complète est plus grand (et de manière combinatoire) que le nombre de services candidats dans l'approche de complétion étape-par-étape. Cela a une incidence directe sur évolutivité du système.
- La condition de terminaison en suggérant une complétion totale n'a pas de taille limite précise puisque le nombre de services requis pour compléter le Mashup n'est pas connu a priori. Par conséquent, la recommandation de services implique un paramètre inconnu qui augmente la complexité.
- La granularité au niveau de la recommandation compte tenu de la grande variété de services et des ressources différentes qui sont disponibles pour le système de suggestion.

Le problème que nous attaquons peut être résumé comme suit: Compte tenu d'un utilisateur en cours de création d'un Mashup au sein d'une plateforme de création de Mashup, comment la plate-forme peut suggérer le Mashup compléter qui répond au mieux à ses intentions, dans un laps de temps raisonnable Notre travail aborde ce problème en identifiant des combinaisons fréquentes de services et en capturant les interactions dites sociales entre les utilisateurs. Cette approche est utilisée pour prédire et proposer les services suivants qui complètent un

Mashup par l'exploitation de la fréquence de cooccurrence et des interactions sociales sur les services précédemment composés. Plus précisément, une stratégie de complétion de Mashups comprend les étapes suivantes:

- Modéliser le problème de complétion sur la base d'exploration des séquences fréquentes. Ainsi, nous montrons que des techniques d'exploration de séquences fréquentes pourraient être mises à profit pour apporter une solution à ce problème.
- Traiter les problèmes d'évolutivité liés aux algorithmes d'exploration de séquences fréquentes. Ceci est réalisé via l'introduction d'un nouvel algorithme de séquence fréquente l'exploitation minière, appelé FESMA. FESMA offre des performances élevées en termes de temps de calcul surpassant les algorithmes existants dans notre contexte.
- Les complétions personnalisées réalisées grâce à l'introduction d'une dimension sociale dans le processus. La dimension sociale est essentielle pour ce travail car, dans le Web 2.0, les utilisateurs peuvent créer, utiliser et partager des services. Nous supposons que les environnements Mashups reflètent les comportements sociaux des utilisateurs et donc, les structures sociales peuvent être extraites à partir des interactions entre les utilisateurs et les services (et entre les utilisateurs). Ces interactions peuvent être analysés et injectées en tant qu'information sociale dans le processus de complétion de composition de service.

L'approche proposée est intégrée et mis en œuvre dans un Framework de composition 'sociale' nommé *Social Composer* (SoCo).

3.2 Un nouvel algorithme pour l'exploration rapide des séquences fréquentes

L'algorithme que nous proposons est appelé FESMA pour Fast and Efficient Sequence Mining Algorithm. Tout comme l'algorithme FP-Growth, FESMA ne génère pas de candidats et utilise une représentation compacte en arbre pour stocker toutes les séquences (c.-à-d. sous-Mashups) qui existent au sein de la base de données des transactions (c.-à-tous les Mashups créés). Contrairement à d'autres algorithmes d'exploration de la séquence, FESMA scanne la base de données qu'une seule fois. Au cours de cette analyse, et pour chaque opération, les séquences sont ajoutés à la représentation de l'arbre en mettant à jour le support associé à chaque séquence et les supports spécifiques à chaque l'utilisateur. Nous avons nommé la structure en arbre stockant toutes les

séquences fréquentes avec leur support FSTree pour Frequent Sequences Tree.

De la définition d'algorithme FESMA, nous pouvons voir que l'on a besoin d'exactement d'un seule balayage (scan) de la base de données pour analyser les Mashups existants (c.-à-d. les transactions). Le coût de l'analyse de la base de données est O (m), où m est la taille de la base de données. Afin de mettre à jour l'arbre de séquence FSTree, chaque transaction est analysée une fois. Le coût de l'insertion d'une séquence dans l'arbre dépend de la longueur de la séquence (profondeur de l'arborescence). Dans le pire des cas, cela coûte de $O(k^2)$ avec k correspondant à la taille de la plus longue séquence. En résumé, la complexité globale de l'algorithme dans le pire des cas est $O(m*k^2)$.

3.3 Une nouvelle approche pour la complétion à granularité fine

A ce stade, nous avons réussi à adapter le calcul de séquences fréquentes et la rendre plus efficace par l'intermédiaire d'un calcul plus rapide et des analyses de bases de données à moindre coût. Dans cette section, nous nous concentrons sur l'utilisation de la représentation générée et les séquences calculées. Intuitivement, lors du traitement de l'ensemble des séquences à l'aide FESMA ou autre, la seule information que nous avons est la fréquence des séquences, offrant un point de vue seulement globale pour les stratégies de complétions possibles. En d'autres termes, puisque les cooccurrences sont calculées en fonction de leurs apparences pour toutes les séquences existantes, ce processus ne considère que l'agrégation des comportements de tous les utilisateurs existants, et qui en somme donne les séquences les plus populaires. Ainsi, la complétion ne peut fonctionner qu'à un niveau 'gros grains', c'est à dire la communauté, sans aucune personnalisation des complétions.

Dans ce qui suit, nous décrivons une meilleure stratégie à base communautaire pour classer les listes de complétions. Ensuite, nous introduisons et motivons une stratégie à grain fin basée sur les réseaux sociaux implicitement extraites de l'analyse des interactions entre les entités du système.

Approche communautaire

Cette fonctionnalité peut être obtenue en utilisant un algorithme d'exploration de séquences fréquentes. A ce stade, il est nécessaire de garder à l'esprit que nous visons à offrir un soutien aux utilisateurs finaux (par exemple, sous la forme de recommandations) pour faciliter la

construction de son Mashup. En appliquant les algorithmes d'exploration de séquences fréquentes nous obtiendrons un ensemble de séquences avec leurs fréquences. Selon l'algorithme utilisé, cette sortie pourrait être représentée et indexée comme un arbre. A partir du schéma de composition introduit par l'utilisateur, une requête est envoyée au système sous la forme d'un service ou une séquence de services (c.-à-d. construite à partir d'une combinaison initiale de services). Le système sélectionne des séquences candidates ayant comme préfixe la séquence requête ou une partie de celle-ci, où le suffixe restant de la séquence candidate représente la complétion à proposer. Toutes les séquences sélectionnées représentent les réponses potentiellement intéressantes pour la complétion. A ce stade, selon une stratégie prédéfinie, les séquences récupérées sont classés selon leur pertinence et seuls les meilleurs k séquences sont proposées à l'utilisateur.

Par définition, le coût de complétion de l'algorithme est fonction de la longueur de la séquence de requête. En fait, pour chaque suffixe de la séquence requête, l'algorithme extrait les complétions de la liste des séquences fréquentes. Cela rend l'utilisation de l'algorithme traditionnel d'exploration des séquences fréquentes inadaptée dans ce contexte. Notre approche alternative utilise la représentation FSTree qui peut être parcouru avec un calcul plus efficace en réduisant le temps d'accès. Une fois l'accès à la branche d'un des suffixe de la requête séquence est réalisé (parcours d'arbre), on a juste besoin de parcourir cette branche pour accéder aux séquences les plus fréquentes.

Approche basée sur les réseaux sociaux

Nous considérons les interactions qui impliquent les utilisateurs finaux comme des interactions sociales. Cependant, les algorithmes d'explorations des séquences fréquentes, même FESMA, ne considèrent pas un niveau de granularité à grains fins puisque (i) ils opèrent principalement au niveau global et (ii) ils raisonnent sur un seul type d'entité, c.-à-d. les services. Ainsi, ils doivent être adaptés non seulement au niveau de l'information sociale, mais aussi pour soutenir le nombre élevé de combinaisons possibles en raison de l'introduction de l'utilisateur dans le processus. Un réseau social dans ce contexte est alors défini comme une abstraction des interactions qui se produisent entre des personnes et des services dans des environnements de composition de services. Le réseau social permet de représenter les affinités de comportement des entités sociales sous la forme d'un graphe social. Cette structure peut être déduite ou extraite directement à partir des intérêts communs entre les utilisateurs de la plate-forme de composition. Le

principe est basé sur la transformation des interactions à un réseau social.

Avec cette nouvelle contrainte, l'algorithme FESMA doit non seulement énumérer et compter les séquences fréquentes, mais aussi le support (fréquence) spécifique des séquences par rapport à chaque utilisateur. En d'autres termes, chaque nœud est lié à des utilisateurs qui ont utilisé la séquence qu'il représente.

Afin d'introduire cette dimension sociale, nous réutilisons le même modèle décrit précédemment basé sur (i) l'extraction de l'information locale, (ii) l'extraction d'information semi-globale, et (iii) l'extraction de l'information globale. La stratégie de recommandation des séquences de complétion tiens compte des usages de l'utilisateur intrinsèque de la fréquence (informations locales), l'intérêt spécifique entre deux utilisateurs (semi-globale de l'information), et le graphe implicite (intérêt mondial entre les utilisateurs). La confiance recommandation RC est la métrique qui indique l'importance de chaque complétion par rapport à un utilisateur cible. Concrètement, lorsque l'utilisateur est en train d'éditer une nouvelle composition de services, et a introduit un commencement de Mashup, une liste des complétions de Mashup est proposée dans l'ordre décroissant de RC confiance recommandation.

En termes de complexité de l'algorithme, en ajoutant les fréquences des séquences spécifique à chaque utilisateur à l'algorithme FESMA, les impacts ne se limitent pas seulement aux ressources de calcul (temps d'exécution), mais se répondent sur l'occupation de l'espace mémoire.

3.4 Etude expérimentale

Comparaison de FESMA aux autres algorithmes

Nous avons effectué principalement deux types d'évaluation: (i) une comparaison l'algorithme FESMA et les autres algorithmes d'exploration des séquences fréquentes, et (ii) une évaluation des propriétés particulières de FESMA pour mesurer la charge générée par la prise en compte de la dimension sociale. Pour cela nous avons utilisé le générateur de données de synthèse de ''IBM quest data generator", avec lequel il est possible de générer une base de données (T10k-l5) qui contient 10.000 transactions (qui représentent des Mashups dans notre cas) où la longueur moyenne des séquences est égale à 5.

De manière générale, les principaux critères de performance utilisés pour évaluer ce genre d'algorithme sont les suivantes: (i) le temps d'exécution et (ii) l'espace mémoire nécessaire pour chaque algorithme

pour trouver des séquences fréquentes dans un dataset. Il convient de noter que dans le cas de FESMA, ce temps comprend la lecture de l'ensemble de données à partir d'un fichier d'entrée et l'écriture des résultats dans un fichier de sortie (des opérations coûteuses en termes de temps). FESMA est implémenté en utilisant la librairie standard C++. Enfin, tous les tests sont effectués sur un processeur Intel Core 2 Duo T9600 cadencé à 2,8 GHz avec de processeur et de 3 Go de RAM.

Les résultats obtenus montent un écart évident entre les résultats obtenus par AprioriSeq et FESMA avec une meilleure performance pour FESMA sur toutes les datasets sélectionnés . Une autre observation intéressante concernant FESMA est sa capacité à gérer des datasets constitués essentiellement de très courtes séquences, une propriété aussi observés sur les portails recensant les Mashups. Enfin, il peut être facilement observé que FESMA est stable par rapport au minimum support et ceci pour des configurations différentes des datasets. Cela signifie que le minimum support n'influence pas les performances de l'algorithme FESMA contrairement aux autres algorithmes existantes.

Calcul du coût d'intégration de la dimension sociale

Pour la seconde expérience, nous avons voulu mesurer la surcharge générée par l'intégration de la dimension sociale au sein de FESMA. Pour l'évaluer, nous avons modifié un jeu de données, en associant à chaque séquence un identifiant d'utilisateur qui est censé être le créateur de cette séquence (Mashup). Nous avons généré l'association User-Mashups en respectant la propriété importante de la distribution d'activité selon le phénomène de la longue queue. Cette propriété fait valoir que certains utilisateurs (Web-utilisateurs) sont beaucoup plus actifs que d'autres en termes de contenu généré (Mashups).

Le résultat montre clairement que les réponses d'exécution de l'algorithme conserver le même comportement avec une moyenne de 25% de surcharge pour l'intégration la dimension sociale. Dans le même temps, même avec la surcharge engendrée par la dimension sociale, les résultats sont plus intéressants que tous les algorithmes existants sans la prise en compte de la dimension sociale.

Evaluation du temps de réponse

A ce stade, nous avons voulu mesurer le temps de réponse de du système de recommandation (complétion). En effet, la stratégie de complétion doit satisfaire les exigences d'applications interactives où les recommandations doivent être calculées en temps réel (en moins d'une

seconde). Comme mentionné précédemment, l'algorithme utilise l'arbre des séquences fréquentes FSTree généré par FESMA afin de récupérer la partie restante d'une séquence introduite par l'utilisateur (la complétion).

Pour effectuer cette évaluation, nous construisons des séquences, en variant la taille de 1 à 10 services, comme requêtes afin de recueillir les complétions suggérées et nous calculons le temps nécessaire par rapport à chaque taille de séquence. Les résultats illustrent l'efficacité de la stratégie de complétion avec des temps de réponses moins de 0,1 seconde. Notez que dans un contexte réaliste, la latence du réseau doit être considérée.