Supporting Service Interactions with Semantics, Machine Learning, and Blockchain

Nizar Messai

HABILITATION A DIRIGER DES RECHERCHES

Discipline : INFORMATIQUE

Année universitaire : 2022-2023

Supporting Service Interactions with Semantics, Machine Learning, and Blockchain

présentée et soutenue publiquement par

Nizar MESSAI

le 14 décembre 2023

Devant le jury (par ordre alphabétique) :

M. Boualem BENATALLAH	Professeur des universités	Dublin City University, Irlande
Mme Amel BOUZEGHOUB	Professeure des universités	Institut Polytechnique de Paris
M. Thomas DEVOGELE	Professeur des universités	Université de Tours
M. Ernesto EXPOSITO	Professeur des universités	Université de Pau et des Pays de l'Adour
Mme Daniela GRIGORI	Professeure des universités	Université de Paris Dauphine
Mme Marianne HUCHARD	Professeure des universités	Université de Montpellier
M. Amedeo NAPOLI	Directeur de recherches CNRS émérite	INRIA Nancy

Résumé

Les services Web se sont imposés comme technologie incontournable pour le développement d'applications accessibles sur le Web. La prolifération des services Web a accentué les défis déjà connus liés à leur découverte et leur composition. Elle en a aussi soulevé des nouveaux notamment ceux liés aux interactions centrées utilisateur tels que la recommandation personnalisée et contextualisée de services, la prise en compte des retours et commentaires utilisateurs, le respect de la vie privée, la gestion de la sécurité et de la fiabilité des échanges, tout en évoluant dans des contextes très dynamiques. Parallèlement, des domaines de recherche complémentaires, tels que le Web sémantique, l'apprentissage automatique et la Blockchain, ont également connu beaucoup d'évolutions aboutissant à la proposition de techniques et d'outils qui peuvent être adaptés pour répondre à ces défis.

Nos travaux se situent à la croisée de ces domaines et visent à proposer des approches novatrices et efficaces capables de répondre aux défis liés aux interactions à base de services Web. Nos deux premières contributions ont pour but d'améliorer la qualité et l'efficacité des recommandations de services Web à travers l'intégration de la sémantique pour l'annotation des services et leur appariement. Nous proposons pour cela deux mesures de similarité sémantiques qui s'appuient respectivement sur les ontologies et sur les données liées. Nous faisons également appel à des formalismes tels que l'Analyse de Concepts Formels et les graphes. Nos deux contributions suivantes visent à améliorer la personnalisation et la contextualisation dans les interactions à base de services. Nous utilisons la théorie de configuration pour fournir des recommandations personnalisées et sensibles au contexte de l'utilisateur. Nous utilisons les principes du Liquid Software pour permettre la migration fluide des interactions à base de services d'un appareil utilisateur à l'autre. Nos deux dernières contributions répondent à l'évolution dynamique des interactions à base de services dans les environnements distribués, ouverts et non fiables. Nous combinons des modélisations en réseaux sociaux avec des modèles d'apprentissage automatique pour gérer les différents types d'interactions et prendre en compte les commentaires d'utilisateurs afin d'entretenir la qualité des recommandations dans le temps. Nous utilisons les Blockchain pour répondre à la problématique de fiabilité dans les collaborations à base de services.

Les approches proposées ont été implémentées en prototypes et évaluées sur des jeux de données, réels et synthétiques, afin de montrer leur efficacité et leur pertinence.

Abstract

Web services have established themselves as an essential technology for the development of real-world applications that may wrap a wide range of Web-accessible functions, resources, programs, databases, sensors, devices, etc. This unprecedented proliferation of Web services not only accentuated the already known challenges of service discovery and composition, but also raised new ones regarding extending service interactions to user-centric perspectives. This includes the challenges of contextualizing and personalizing service recommendation, handling user contributions and feedbacks, preserving privacy and trust while evolving in highly dynamic contexts. At the same time, other hot-topic research areas such as ***Semantic Web***, ***Machine Learning*** and ***Blockchain***, achieved high maturity level techniques and tools that can be adapted to address these challenges. *In our research, we focus on the convergence of the previous areas of study, aiming to introduce cutting-edge approaches that leverage semantic-awareness, to handle major challenges encountered throughout the cycle of life of web services. Our ultimate goal is to allow successful and effective interactions in service-service scenarios within compositions, as well as in user-service scenarios.* First, to address the mismatch problem in service discovery and recommendation, we propose semantic similarity measures based on ontologies and Linked Open Data. These measures are the basis for a semantically enabled Web service interaction framework including service discovery and composition which also appeals for additional formal models such as Formal Concept Analysis and Graphs. Second, we tackle the lack of personalized service recommendation and multiple-device service migration in user-centric service interactions. We propose a Configuration-based approach to meet highly personalized and contextualized service recommendation and Liquid Software based approach for smoothly migrating user-service interactions in multiple-device contexts. Third, we address the challenge of dynamic evolution of service interactions in open untrusted environments. We combine graph network representations with Machine Learning models to capture service interactions and user feedbacks to achieve accurate service recommendation. We finally address the untrusted collaboration issues in service interactions through the use of Blockchain and Smart contracts. We implemented our methodologies in prototypes. We tested them rigorously on contemporary datasets, proving their robustness and showcasing their effectiveness over prevailing state-of-the-art strategies.

Contents

List of Tables

List of Figures

Part I

Research Context, Research Problems, and Contribution Synthesis

Chapter 1

Introduction

1.1 Short biography

Dr. Nizar Messai is an Associate Professor in Computer Science at the University of Tours, France, since November 2011. He received his Master and PhD in Computer Science from University of Lorraine, Nancy, France, in 2004 and 2009 respectively. Before joining the University of Tours, he spent 2 years as Teaching Assistant at the university of Lorraine, 1 year as R&D engineer at Procton Labs Paris and 14 months as a post-doctoral fellow at École Centrale Paris, France. His research interests mainly include Semantic Web, Linked Open Data, Web services, Internet of Things, and Blockchain, with a particular focus on their interactions and combinations. He co-supervised 7 PhD students, of which 5 have defended their thesis and two are about to do in 2023. He also supervised several Master's and Bachelor's degree internships related to his research work. He has published over 50 articles in international and national peer-reviewed journals and conferences, including 2 best papers. He is the PI of 2 regional research projects on semantics and Web service based recommendation applied to Tourism and Healthcare domains. On the teaching front, Dr. Nizar Messai has set up several teaching modules in Computer Science. He was Deputy Head of the Computer Science Department for 6 years. He is co-responsible for the computer science bachelor's degree at Tours, and co-director of studies for 1st and 2nd year bachelor. He is an elected member of the board of the Faculty of Science and Technology from 2018 to 2023. He is also elected member of the board of LIFAT, Computer Science Research Laboratory of the University of Tours. He was awarded RIPEC bonus for the quality of his research and teaching at University of Tours.

1.2 General research context and motivation

The emergence of Service Oriented Computing (SOC) paradigm two decades ago has deeply revolutionized the way of developing software systems. This paradigm relies on

the concept of Service as an abstraction of autonomous software component regardless of any underlying implementation. Any service can then be run separately to provide the function it was designed for or be combined with other services into composite ones to provide a new value-added function. In this way, services act as loosely coupled building blocks that can be assembled into a network of services to create flexible, dynamic business processes and agile applications [134]. The SOC paradigm promise was quickly realized through Software Oriented Architectures (SOA) that enable the complete ecosystem for interoperable service discovery, utilization, and combination to support virtually any business process in any organizational structure or user context [74]. One of the most successful realizations of SOA is Web services which extends SOA principals over the Web as the largest infrastructure for service interactions. Web services took advantage of Web protocols and their highly dynamic evolution. They in turn contributed to the emergence of new application-to-application interactions over the Web in a service centric perspective that progressively replaced the data centric perspective of the Web. Web services was first designed over XML-based standards resulting from intensive standardization activities for all Web services lifecycle aspects including WSDL, SOAP, and UDDI, among others, respectively for service description, messaging, and registration/discovery. This variant of Web services implementation usually referred to as SOAP Web services gave way to Representational State Transfer (REST) [67] Web services also referred to as RESTful Web services [141, 136]. REST architecture relies on HTTP protocol and brings more flexibility for service interactions. The simplicity and flexibility of REST motivated the shift towards RESTfull Web services which consequently became the most commonly advocated approach for developing service-based applications. REST architecture is also the most used for Application Programming Interface (API) developments which supports more features and communication possibilities than Web services. Quickly adopted and supported by the major IT firms, REST APIs are today established as the means for Web application interactions. They range from the most complex applications including Web giants (Google, Amazon, Microsoft, etc.) tools and processes to the most basic cases of connected devices in Internet of Things (IoT) contexts. Following the service integration paradigm, APIs are also composed into bigger applications called Mashups. The last most-known variant of service implementations exposing the functionalities on the Web are Cloud services. They go further than just the main functionality of Clouds as online data storage solutions to a wide range of cloud service categories including Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Function/Machine Learning as a Service (MLaaS) and more generally Every thing as a Service (*aaS).

The success of Web services to propose a flexible and cost-effective way for distributed application development that meets enterprise interaction requirements, has significantly increased the number of available atomic and composite Web services. This large heap of

available services exponentially grows when it comes to building new composite services over all the composition possibilities. This straightforwardly rises the challenge of identifying relevant services either for single function invocation or for designing composite service for more complex tasks. First generation tools for service discovery based on their syntactic functional and non-functional descriptions fall short to face this challenge [116]. At the same time ontologies and the Semantic Web were gaining a wide interest towards providing machine understandable Web contents [20]. Enhancing Web services using semantic Web standards quickly became a thriving area and gave birth to semantic web services [114] as a new generation of Web services. Existing standards were extended with semantic dimensions describing service capabilities, eg. WSDL-S, and new ones were also proposed as a full semantic language for web services, eg. OWL-S. These languages mainly concern SOAP Web services and rather harden the complexity of service integration while the trend is to promote lightweight service architectures. Semantic handling is then left to an annotation process within service engineering frameworks. Such process relies on external semantic resources to automatically annotate service operations and parameters with external resources [146]. Domain ontologies were largely used for service annotation with abstract semantic concepts. However the domain specific focus of ontologies makes them less relevant when it comes to connecting heterogeneous services with complementary functions from complementary domains to build composite services. Cross-domain ontologies such as DBPedia and then the Linked Open Data (LOD) brought some answers for "domain-independent" semantic service annotation. Recent advances in Machine Learning and Natural Language Processing gave rise to active researches to capture service functionalities and underlying semantics from text descriptions, interaction logs, and user feedbacks [98, 175].

Over the last ten years, I have been particularly interested in the intersection of the two research topics of Web services and the Semantic Web. My aim was to bring innovative sound approaches that leverage all Web service life-cycle steps with native semantic awareness to achieve high level of successful and effective interaction in service-service scenarios within compositions and mashups as well as in user-service scenarios. I was particularly interested in defining declarative models with evolving rules to keep pace with context changes. Within this perspective, my research contributions started with semantic similarity measures proposal for service classification and recommendation, and progressively moved to handle personalized service-service and user-service interactions including privacy and trust awareness. These contributions appealed for the most hot topics in interaction with Web service research fields including (i) ontologies and Linked Open Data (LOD) for semantic similarity measures, (ii) Formal Concept Analysis (FCA), Configuration Theory, First Order Logic and Dynamic Condition Response (DCR) Graphs for declarative frameworks, (iii) Machine Learning for service interaction handling, and (iv) SOLID standards,

Blockchain and Smart Contracts (SC) for privacy and trust in service interactions. These contributions were achieved in strong collaboration with colleagues and PhD students from University of Tours and other universities within international collaborations. They were mostly supported by two scientific projects, Smart Loire and SQVALD, funded by Région Centre Val de Loire. These projects provided application contexts with specific challenges respectively in the Tourism and Healthcare domains. This dissertation will go through these contributions with sufficient details to be self-contained and easily understood. In case of need for in-depth focus the reader can go through the corresponding published research papers which contain fully detailed description of each contribution with related work review, experiments and links on used libraries and datasets.

1.3 Manuscript organization

The manuscript is organized into five parts :

Introduction: This initial part, which contains the current chapter and Chapter 2, presents the general context of my research activities. It raises the identified research challenges, and summarizes key contribution I come up with to answer these challenges.

Semantically-Enabled Service Interaction (Part II): This part is dedicated to our contributions on enabling service interaction via semantic capabilities. Chapter 3 explores our work on ontology-based similarity. Chapter 4 discusses the details of our new LOD-based similarity.

Context-Aware and Personalized Service Interactions (Part III) : This part presents the details of our contributions towards context-aware and personalized service interactions. A service perspective approach is discussed in Chapter 5, while Chapter 6 focuses on the perspective of running devices.

Handling Service Interaction Evolutions (Part IV) : This part highlights the details of our contributions towards handling service interaction. Chapter 7 discusses our work on handling evolutions via leveraging ML models. Chapter 8 describes the exploitation of Blockchain technology to set trustworthy service interactions.

Conclusion and Research Perspectives (Part V) : This final part provides a general conclusion, sets the ongoing research efforts and suggests potential next steps to consider as future research directions.

Chapters 1, 2, and 9 outline a summary of our contributions, highlighting the essential elements of research and providing insights on future perspectives. The remaining chapters elaborate on the deep details of the proposed contributions, providing an in-depth comprehensive exploration of each one.

Chapter 2

Research Problems and Contribution Synthesis

Contents

This chapter introduces the research challenges we identified in service interaction contexts and summarizes the scientific contributions achieved towards dealing with these challenges. In Section 2.1, I discuss three main research challenges: Semantically-enabled service interactions, in Section 2.1.1, Contextualized and user-centric service interactions, in Section 2.1.2, and Dynamic service interactions and evolution, in Section 2.1.3. Then, in Section 2.2, I give an overview of the ways we tackled each challenge respectively. For each contribution, I discuss the main ingredients, the formal models, and the architecture elements to help understanding the main idea and the ways we proceeded to achieve it. I leave detailed presentation to dedicated chapters.

2.1 Research problems

After two decades of intense development and innovation, Web services have established themselves as an essential technology for fast establishing next generation real-world applications that may wrap a wide range of Web-accessible functions, resources, programs, databases, sensors, devices, etc. This unprecedented proliferation of Web services not only accentuated the already known challenges of service discovery and composition, but also raised new ones regarding extending service interactions to user-centric perspectives. This includes the challenge of contextualizing and personalizing service recommendation, handling user feedbacks, preserving privacy and trust while evolving in highly dynamic contexts, etc. For both types of challenges we believe that a successful approach should provide declarative formal models capable of capturing semantic meanings underlying service descriptions, functions, compositions, evolutions, and interaction contexts. Our research activities towards achieving such approach mainly address the following specific research challenges.

2.1.1 Semantically-enabled service interactions

Handling Web services semantics was an evident answer for several efficiency problems faced by first generation Web services. Indeed, service management frameworks that relied only on syntactic descriptions of services were error prone because of the mismatch problem between descriptions of service functionalities and parameters. They were also time consuming because of the lack of abstraction structures of service collections which also leads to low accuracy when it comes to discover relevant services for specific use. Semantics brought new perspectives and challenges towards efficient automation of service life-cycle management with machine understandable and processable services. Too many approaches were proposed since then and covered different facets of enhancing Web service life-cycle

management with semantic level. The main focus was to annotate services with abstract concepts from semantic resources such as domain ontologies. The abstraction semantics are then used to identify services in discovery process, to classify services into classes based on their semantics, and to decide how likely two services can successfully build a new composite one that matches the expected value-added function in composition process or to which end one service can substitute another service to ensure continuous composition running.

These intensive research and development activities in semantic Web service management resulted in significant improvements of Web service management tools and frameworks to achieve high performance and accuracy especially in basic use-cases with optimum conditions of use including: identified specific domain, full accurate descriptions of Web services, rich and expressive domain ontologies, etc. In general, however, such conditions are not always met. Web service composition process may involve heterogeneous services with complementary functions referring to complementary domains rarely covered by one single domain ontology, rising evident need for cross-domain semantic resources. In addition, service descriptions provided by service providers may not be exhaustive as this task requires a high domain expertise to choose the most appropriate description terms. Service descriptions may also result from aggressive advertising and campaigns to promote providers services, and again used terms may not be the appropriate ones to use. Consequently the right meanings for service functions may not be captured by semantic annotation process. It therefore becomes necessary to complete service annotation with data from feedbacks on real service running and interactions.

Once the semantic service annotation challenge is tackled, the next challenge to cope with is twofold. First, defining efficient metrics and measures that exploit semantic annotation together with additional criteria to provide accurate similarity measures for Web services. Second, building dynamic and flexible structures over the similarity measure that makes it easy to quickly and efficiently identify relevant services for a given usage. To this end, typical graph-based representations could bring interesting solutions. However, they require the integration of a semantic layer to support relationships between atomic Web services as well as those within compositions. The built structures should in turn serve as flexible search space for service discovery and for composition building. Efficient algorithms have therefore to be defined over the classification structure to support the discovery process as well as to generate successful compositions through the use of planning tools.

Another major challenge yet to be addressed is to define the underlying formal models for a full round-trip service management framework that enables accurate annotation, flexible classification, efficient discovery, reliable composition generation, and efficient recovery

from service failures. These points among others show that there are still challenges lying ahead of the semantically-enabled service interactions. This dissertation is partly focused on our contribution regarding these challenges.

2.1.2 Contextualized and user-centric service interactions

2.1.2.1 Need for personalized service recommendation

The digital transformation of the society brought new ways and opportunities for service-based application development and consuming. Powerful mobile devices have become a basic must for even trivial daily-life use. They allow access to a wide range of Mobile and Web applications covering professional use such as shared workspaces, calendar scheduling, remote meeting, team managing, etc. as well as personal use such as weather forecast, gaming, home monitoring, online shopping, etc. In many cases these applications are built as mashups of existing services and APIs through the use of mashup tools and platforms such as ProgrammableWeb. These tools have significantly lowered the barriers to mashup development allowing less experienced contributors with few programming skills to assemble their own applications [39, 172] in an intuitive way. This straightforwardly led to evident increase of available services resulting in an unparalleled large scope of choices on selecting services for further mashups. Consequently, the technical facility for mashup building comes up against the challenge of choosing the appropriate services to compose from the plenitude of the available ones.

In this context, semantically enabled frameworks (Section 2.1.1) can bring partial answer as they can be employed as the preliminary selection criteria indeed for coarse ranking. Semantic matchmaking process will likely find a large set of candidate web services with close semantic similarity. The challenge then is to define additional ways to achieve highly relevant service recommendation approaches able to provide mashup builders with the most relevant services to achieve their specific added-value needs.

This challenge of service recommendation fine-tuning has been addressed through considering contextual information characterising the situation for service interaction, which led to the context-aware services discovery research topic defined as the ability to make use of context information to discover the most relevant services for the user [152, 21]. Context-aware service discovery was widely studied from the perspective of ubiquitous computing. In the earlier proposed approaches the context was usually reduced to the user/service location. More advanced ones introduced a wide range of heterogeneous contents relying on predefined ontologies and user context history with the promise to delivering “the right information at the right place and the right time" to users [21]. Service interaction contexts may vary within highly heterogeneous situations making it difficult for any approach

to define a global view with sufficiently relevant criteria to capture service usage interest and therefore provide an accurate service recommendation. This straightforwardly raised the need to consider specific application domains to focus on relevant selection criteria for service recommendation [41]. This is the case of the Tourism domain for example where the need for service recommendation systems have grown as far as the tourists need shifted to more personalized experience.

Although a lot of work has been done in this direction, there is still a gap regarding user contribution within Mashup creation in intuitive and efficient way. The challenge that still needs to be addressed is to achieve efficient framework that first provides inexperienced users with intuitive ways to create abstract mashups that capture user needs and preferences and then rely on efficient formal and declarative models to derive relevant context-aware and domain specific mashup instances that meet the user needs.

This dissertation tackles this twofold challenge from the perspective of Tourism domain. We discuss our contribution towards a visual intuitive language for mashup creation and a configuration-based framework for personalized and user-centric service recommendation in the Tourism domain.

2.1.2.2 Need for fluent user-side service running management

In the same context of digital transformation of the society, end-users use mobile devices to interact with the services and applications discussed so far from the perspective of personalized service recommendation. The most commonly used devices are smartphones and pads. However, depending on the interaction context the mobile device may range in a larger variety including sensors, screens, home-devices, wearable devices, etc. What becomes more common is that the user may interact with the same service from more than one device simultaneously or more usually in sequence to continue an interaction on a second device when it becomes impossible or uncomfortable to use the first one. The related research topic is known as service synchronization and migration in multiple-devices scenarios [78, 170]. The best known contributions that addressed this topic relied on the concepts of Liquid Software [80] to adapt the service interactions to the used device [122, 70].

Although these approaches succeeded to adapt service interactions to the chosen running device, they mostly relied on service-side synchronization. Cloud-based solution indeed brought a wide range of support regarding elasticity and resource management for service execution. On the user-side, however, these approaches do not yet allow to obtain simple data-flow without restarting user-service interaction sessions from scratch each time the user moves to a new device.

Successful approaches towards the challenge of user-side service migration/synchronization should tackle the following issues to provide the user with high quality of experience: First, it should review synchronization and data-flow handling within the REST protocol. Second, it should provide efficient matching algorithms between user-service interaction requirement and devices capabilities as well as efficient device recommendation for service execution. Last, it should rely on sound formal models that guarantee the approach to be generic and extensible to any application scenario. This dissertation discusses our contribution towards these challenges.

2.1.3 Dynamic service interactions and evolution

2.1.3.1 Need for handling dynamic evolution of service interactions

Web services compositions and Mashups designed through the challenges discussed so far are then run to provide the expected results. They usually hold over several interaction sessions. Therefore, the next important challenge to cope with is the dynamic evolution of such mashups and compositions over time and the interaction relationships services may hold within. Basically, the most evident relationship between two services is the *composition* relationship to yield composite added-value services. With the development of mashups and new generation Web services the composition is generalized to *collaboration* relationship defined as the ability of two or more services to work together to satisfy users requests. Classifying Web services based on their similarities yields the second type of relationship between services which is the *substitution* or *replaceability* between the highly similar services to avoid mashup breaking in case of service failure. Substitution is naturally generalized into *competition* between services providing the same functions being provided by concurrent entities.

The quick and phenomenal progress of social networks development and their highly dynamic evolution inspired researchers on the Web services to model Web service interaction relationships as a social network of services [111] giving birth to the concept of social Web services [110]. In such network, services which are the entities identify those peers with which they'd like to work (collaboration), those that can replace them in case of failure (substitution), and those that compete against them for selection (competition). In the same direction, new types of relationships such as *subscription* have been proposed later to capture situations when a service tries to keep updated with the evolution of another [97].

Following the social network metaphor, Web service networks allow comments and feedbacks over services and relationships. This generates relevant new data yet to be considered to keep high quality mashups, or in the in contrast, to prevent the failure of

badly constructed ones. Feedback analysis allows also to evaluate the real quality of service (QoS) of given services that can be set against the QoS given by service providers. This highly competitive network of services forces service providers to regularly update their deployed services in order to continue taking part in the engaged collaborations. They also deploy new services to answer new needs and/or follow new trends. These updates and deployments are also described through additional data, largely free text. Consequently, social Web service computing brought new interesting representation models that capture the continuous service deployment and the dynamic interaction relationships services may partake during their lifetime. It however raised the new challenges of handling the generated data, and analyzing it to ensure and maintain network consistency and evolution while providing accurate relationship recommendations. Although many research works tried to tackle this challenge through the use of Machine Learning approaches for text analysis of descriptions and comments on services and their interactions [99], there is still a need for a more general framework that relies on both semantically-enabled models and machine learning tools.

2.1.3.2 Need for trustworthy and dynamic service execution

Web services are widely used in today's business world as the functional part of larger Business Processes which also integrate other enterprise perspectives including data and control. In its abstract representation, a business process is modeled as a graph of tasks, events and control operators [164]. Service-based business processes rely on Web services to implement their tasks. Business processes are designed to achieve business goal which usually involves the collaboration of many entities in decentralized environments. In such open environments, collaborating entities are also competing for their own business success which leads to untrusted behavior. Therefore, successful collaborations require either a central coordination between involved parties or at least an agreed on reference broker to mediate and control the execution. The first case, mostly known as *orchestration*, faces several issues including central node congestion, additional fees, etc. The second case, known as *choreography*, is more suitable for open collaboration as entities collaborate directly without intermediaries. They however still need to rely on well-defined collaboration rules and policies to trust each other towards successful collaborations. These issues have been widely addressed by state-of-the art approaches that usually rely on formal and transactional models to define collaboration rules and on formal verification to ensure execution soundness [163, 22, 77]. These approaches fail, however, to dispense with the need for independent third party to guarantee compliance with trust policies between engaged entities.

The emergence of Blockchain technologies for decentralized and transactional data shar-

ing across a network of untrusted participants brought the right solution that fits well with the requirements of business process choreographies executions. It didn't take long for the Blockchain technologies to be adopted as infrastructure that handles trustful interactions between business entities without any need of a central authority [88, 169, 117, 64, 36]. These approaches mostly rely on second generation blockchains which allow for complex task running through programs called Smart Contracts that enforce the terms of agreement between the untrusted parties. The proposed approaches focus on execution control flow of business processes while leaving transactional flow to the basic blockchain specification that follows standard ACID model which fails to capture advanced reliability requirements of complex business process executions. There is therefore need to define suitable models to handle the transactional flow of blockchain-based process executions.

Collaborating business entities evolve in highly dynamic environments that appeal for continuous updates and improvements of their internal processes to keep high competition level. These dynamic changes may affect the collaborations in which the enterprise takes part. Changes may also directly concern the global collaboration due to new external business policies that should be taken into account. In both cases, changes must be notified and propagated to the involved partners. Change propagation in process choreographies has also been tackled prior to blockchain-based approaches [65]. In the blockchain-based ones however it is still challenging because of the non-mutable nature of shared data. There is therefore need to bring additional levels to cope with the change propagation within blockchain based choreographies execution frameworks.

This dissertation will discuss our contribution towards transactional flow modeling and change propagation model for blockchain-based choreographies.

2.2 Contribution Synthesis

In this section, I give a summary of our contributions regarding the research problems discussed so far. I will discuss the intuition behind each contribution and its main ingredients including the formal model, the architecture, and the specific application context. Each contribution will then be detailed in a dedicated chapter of the manuscript.

2.2.1 Semantically-enabled service interaction framework based on Ontologies, Formal Concepts Analysis, and Linked Open Data

Our contribution towards the challenge of semantically enabled service interactions discussed in Section 2.1.1 is mainly focused on proposing efficient semantic similarity measures and then designing service life-cycle modules based on these similarities. We first proposed

ontology-based semantic similarity measure within a fully integrated framework for service composition, which appeals for concepts lattices and graphs for service classification and composition generation. Then, following the evolution of semantic Web to Linked Open Data, we proposed a LOD-based semantic similarity measure for service recommendation. The proposed LOD-based similarity measure acts then as the central component of an open source library we developed and proposed to the community as an API and as a Maven dependency. These contributions are summarized in the following subsections.

2.2.1.1 Ontology-based similarity and Formal Concept Analysis for Web service composition

This contribution starts by proposing an Integrated Development Environment for Composite Services Engineering (IDECSE) that considers semantics in all the composition steps: analyzing user query, classification of services in the registry based on their features semantic relatedness, composing services, and verifying the composition process. The proposed architecture has 3 main components: (i) Service matching component which relies on efficient similarity measure computing, (ii) Service classification component that provides a flexible organisation of service registry to support efficient service discovery, and (iii) Composition generation component that allows the establishment of coherent composite services relevant to the desired functionalities.

Semantic similarity measure for Web services The IDECSE Framework deals with semantic Web services features which are annotated through ontological concepts. Web service description files features include functional features such as Inputs, Outputs, Preconditions, and effects, and non-functional ones such as QoS and service invocation constraints. We propose a similarity measure that focuses on functional features and leaves non functional ones to the composition process. The proposed similarity measure is defined as the sum of weighted similarities between pairwise service Inputs, Outputs, Preconditions, and Effects, denoted as follows for any two services S_i and S_j:

$$\sigma(S_i, S_j) = w_1 Sim_{In}(S_i, S_j) + w_2 Sim_{Out}(S_i, S_j) + w_3 Sim_{Prec}(S_i, S_j) + w_4 Sim_{Eff}(S_i, S_j)$$

where $\sum_{k=1}^{4} w_k = 1$. The weights w_i allow modeling the similarity measure to be adapted to different application cases where it may be required to focus on IO only or any other possible combination of the four aspects. In the case where no specific preferences are given, these weights are equally set to 0.25.

Each of the four Sim functions is in turn defined as the sum of two weighted concept-based and data-type based measures as follows for the case of Inputs and analogously for

the three other dimensions:

$$Sim_{In}(S_i, S_j) = u_{In_1} Sim_{Co}(In_i, In_j) + u_{In_2} Sim_{DT}(In_i, In_j)$$

where $u_{In_1} + u_{In_2} = 1$.

Web services features are semantically annotated using concepts from ontologies. Measuring their similarity turns out to measuring the similarity between their annotation concepts in the considered ontologies. Comparative study of the state-of-the-art approaches on semantic similarity measures have shown a better correlation coefficient and higher efficiency rates for the measure proposed in [145] which follows the Tversky model [157], where common taxonomical features tend to increase similarity while the non-common ones decrease it. We therefore adapt the Sanchez formulas to define Sim_{Co}.

In the same way, we reviewed and adapted the best state-of-the-art data-type measures to define Sim_{DT}. Following works in [139] and [154] we propose a practical measure between different data-types that defines abstract groups of compatible data-types and then assigns similarity values to each pair of groups. Considering Sim_{DT} prevents meaningless composite services built upon highly similar services with respect to Sim_{Co} which however fail to match their data-types. Following this intuition, the weight u_{In_1} must be higher than u_{In_2} to focus on semantic similarity while keeping data-type similarity for checking the composition consistency. Conducted experiments confirmed that the combination $u_{In_1} = 0.80$ and $u_{In_2} = 0.20$ gave the highest accuracy.

The proposed Web service semantic similarity measure within the matching component plays the central role in the IDECSE framework as it is the basis for service matching, classification, discovery and composition as it will be discussed hereafter.

Flexible service classification with Formal Concept Analysis The way services are organized within the service registry highly impacts the efficiency of service identification for any given need. We therefore propose to build the classification component of IDECSE upon the well-established formalism called Formal Concept Analysis (FCA) [71] and particularly its derived structures called Concept Lattices.

Based on partially ordered sets theory, FCA allows to structure the data into a hierarchy of partially ordered concepts called the Concept Lattice. Each concept corresponds to a maximal set of objects having in common a maximal set of attributes. The concept lattice hierarchical relation is based on the dual partial order between sets of objects and dually sets of attributes. This structure and its properties motivated the use of concept lattices as indexing structure for information retrieval either by direct querying or by progressive navigation [118, 119].

Following this direction we apply FCA to automatically organize services into a concept

lattice based on the semantic similarity defined so far. FCA applies on binary matrix data called formal context, which is not the case of service data set. We operate data transformation as follows. Services are considered as attributes and objects and encoded within a matrix $SeSimMat$ of service similarities. We define a similarity threshold $\theta \in [0\,,1]$ and generate a service similarity context matrix $SeSimCxt$ from $SeSimMat$. $SeSimCxt$ is in turn used to generate a formal context $SimCxt$ represented as a triple (S,S,R_{Sim_θ}), where S is a set of services, and R_{Sim_θ} is a binary relation indicating whether two services can offer similar functionality or not.

Starting from $SimCxt$ we can build the corresponding concept lattice structuring the services in the registry. We define querying and navigation algorithms upon the service concept lattice: (i) to find equivalent, similar, and composable services to the requested need, (ii) and to select the most relevant services for the purpose of substitution process to maintain a composite Web service application functionality as much as possible.

Depending on the request need, selected relevant services are transmitted to the corresponding component of the framework.

Planning and Graph-based service composition The last component of the IDECSE framework is the composition generator. It takes as input a set of similar services returned by the previously presented components to provide a set of coherent composite services relevant to requested function. This component appeals for AI planning techniques and graphs together with the semantic similarity measure and QoS constraints to ensure successful compositions. The key point is the compatibility between Input-Output interfaces of the services to be connected within the resulting composite ones.

The highest level of compatibility is achieved with identical matching between Input-Output which feats the use of AI Planning. In this case, Web service composition problem is mapped to a planning problem based on transformation rules defined in [82] and using the Planning Domain Definition Language (PDDL) [75]. In this mapping, each service S_i is represented as a state St_i, its inputs and preconditions are represented as the state preconditions $prec(St_i) = S_{i_{Inputs}} \cup S_{i_{Preconditions}}$, and its outputs and effects are represented as the state effects $eff(St_i) = S_{i_{Outputs}} \cup S_{i_{Effects}}$. We apply STRIPS planning algorithms on the resulting mapping problem to generate possible solution plans corresponding to possible service compositions.

In the general case however, there may be no exact Input-Output interface matching. The composition generator relies on the proposed semantic similarity measure to compute approximate matching possibilities. The similarity measure $\sigma(S_i, S_j)$ is redefined to apply on the outputs/effects of S_i and the inputs/preconditions of S_j. The obtained similarity values are mapped to logical match type as defined by [100]. The logical match types are

Fail, *Subsume*, *Plug-in*, and *Match* which respectively correspond to composability values in [0 , 0.3[, [0.3 , 0.6[, [0.6 , 0.8[, and [0.8 , 1].

Services with "*Match*" or with "*Plug-in*" are most likely to lead to successful composition. The next step consists of considering these services to build a directed weighted graph with nodes being services and edges being composability values. We formalize satisfiable compositions as S_{comp} ={ I_{comp}, O_{comp} } where I_{comp} is a set of inputs that present a valid degree of match with the set of requested Inputs $S_{R_{In}}$ and O_{Comp} is a set of outputs that present a valid degree of match with the set of requested Outputs $S_{R_{Out}}$. Then, service composition solutions are those corresponding to paths in the graph between $S_{R_{In}}$ and $S_{R_{Out}}$. The graph search algorithm is enhanced by considering QoS constraints over services to fine-tune composition solutions ranking.

The work described in this subsection was conducted during the PhD of Ahmed Abid defended on July 2017. More details will be given in Part II, Chapter 3.

2.2.1.2 Linked Open Data based similarity for Web service recommendation

This contribution takes advantage of the evolution of Semantic Web to Linked Open Data (LOD) to propose efficient similarity measures that operate for any domain overcoming the shortcuts of ontology-based similarities which usually focus on few specific domains. The contribution is threefold: (i) we propose a new similarity measure, called LODS, that assesses the matching degree between pairs of terms represented with LOD resources, (ii) we apply LODS for context-aware service recommendation, and (iii) we propose an open source LOD-based similarity measure library.

LODS: a Linked Open Data based Similarity Measure LOD initiative [23] aims at making a gigantic interlinked database of interlinked datasets/resources following few basic interlinking principles [19]. LOD have quickly grown and led to an important source of semantic information with high informativeness that can be useful for many applications including semantic similarity calculation [120]. Following these advances, we propose LODS, a LOD-based similarity measure that takes advantage of LOD properties and resources interlinking to compute similarity between any two terms.

DBpedia [102], which realizes LOD vision by structuring Wikipedia content and interlinking it with external datasets, acts as a central kernel of the LOD cloud. We therefore consider it as a starting point to glean initial data to compute similarity degree between compared concepts. Then, we follow interlinks to enrich data from other datasets through a data augmentation process. Among the numerous LOD properties, LODS focuses on the following ones:
Instances: LOD resources are described with various multi-domain ontologies making it

possible for a given LOD concept to have multiple instances in different ontologies with different levels of description richness.
Categories: LOD resources can be arranged into categories with different classification granularities and depths.
Resource properties: LOD resources are described with different properties which distinguish and characterize resources.
Interlinking: LOD resources are usually interlinked to each other using *owl:sameAs*.

Considering these properties we define LODS as the average of 3 sub-measures: Instance-based ($SimI$), Category-based ($SimC$), and Property-based ($SimP$). Formally, for any two LOD resources a and b:

$$LODS(a,b) = avg(SimI(a,b), SimC(a,b), SimP(a,b))$$

SimI exploits the taxonomic structure of ontological concepts used to instantiate resources. It extracts resources features by getting their super-concepts from each annotating ontology. It then applies the Tversky model on the set of features. The final similarity value is the average of values obtained for all the annotating ontologies.
SimC follows the same principle as *SimI* but applies on taxonomic structure of classification schemata and limits the number of extracted super and sub categories while extracting resources features.
SimP exploits ingoing and outgoing resources properties. It applies Partial Information Content (PIC) measure to focus on the discriminant properties rather than the commonly used ones.

The detailed formalization of these sub-measures is given in Chapter 4. The intuition for combining these sub-measures is to take into account the LOD reality where resources are not systematically described with ontological concepts. They are neither always arranged in categories nor well distinguished with properties. The objective is to reduce the negative impact of poorly described resources on the resultant similarity score. Each measure applies on enriched data by following links between different dataset. We define *SimI* and *SimC* as feature-based measures based on the Tversky model [157] instead of distance-based methods. This is mainly because in multi-domain ontologies, taxonomic relations do not necessarily represent uniform distance [145].

Context-aware LOD-based service discovery LODS similarity measure is applied for context-aware service discovery and recommendation. The proposed approach consists of the following steps: (i) We define an annotation model that captures both functional properties for service actions and non-functional properties for contextual information. (ii) We enrich the service model instances with semantic data from LOD. (iii) We propose

matchmaking algorithms based on LODS to identify relevant services for user needs in a given context. (iv) We illustrate the approach effectiveness in the tourism context provided by the SmartLoire project.
(i) Context-aware service annotation model: We represent a service S as a triple $\langle n_s, \mathcal{F}_s, \mathcal{NF}_s \rangle$, where n_s is the service name, $\mathcal{F}_s = \{f_1, f_2, \ldots, f_i\}$ is the set of service functionalities, and $\mathcal{NF}_s = \{P_{1,s}, P_{2,s}, \ldots, P_{n,s}\}$ is the set of service non-functional property sets. Due to the diversity of non-functional information, these sets are generally organized in profiles, where each profile $P_{i,s}$ describes a specific context information domain.
(ii) LOD-based enrichment of service model instances: Functional service features in $\mathcal{F}_s$ are annotated with LOD resources starting with DBPedia and following interlinks as in the previous section. Non-functional features however may be either quantitative or qualitative properties. Only qualitative properties which take string values can be annotated with LOD resources.
(iii) Extending LODS for context-aware service discovery: We define a two-step discovery process. The first step considers functional features and their LOD annotating resources and relies on LODS measure for computing service similarities with respect to requested service. The second step considers non-functional properties with a conditional similarity measure which applies LODS on LOD annotated qualitative properties and adequate numerical similarity measures on quantitative properties. The overall non-functional similarity is a weighted combination of the similarity values of the set of profiles. The weights allow to define importance to a given profile rather than another depending on the context.
(iv) Applying the approach in the Tourism domain: Within the SmartLoire project, we consider a dataset of touristic services provided by ADT37 (Agence Départementale de Tourisme d'Indre-et-Loire) through its platform Tourinsoft (http://www.tourinsoft.com/). The dataset contains 1632 services classified into different touristic offer categories and annotated through functional and non-functional properties. We applied the previously discussed steps and evaluated the discovery relevance by implying more than 20 real users in the query definition and results ratings. Obtained results from different metrics demonstrated the high matchmaking relevance in both functional and non-functional discovery phases.

LDS: LOD-based similarity measures library In order to go further than the LODS similarity measure, we propose LDS (Linked Data Similarity) Library as an open source Java software library of LOD-based semantic similarity measures. LDS defines a generic, reusable, and extensible architecture that integrates the best-known existing LOD-based similarity measures and supports both developers and researchers who intend to use these implemented measures or develop their own new ones.

LDS development follows the following requirements: (i) *Simple design* to facilitate the reuse of the tool and provide developers the ability to extend and use it with their developed semantic measures; (ii) *Efficient LOD querying, caching and indexing* to ensure similarity calculation efficiency and scalability; and (iii) *Extensibility* to offers developers a way to implement rapidly their measures and incorporate the library with their work.

The LDS architecture has 5 main components: (i) *Similarity Engine* which acts as the interaction interface of LDS with external programs. (ii) *LOD-based Similarity* which calculates the similarity values. (iii) *Linked data Manager* which retrieves data for similarity calculation. These two components act as plugability layer allowing to easily integrate new similarity measure as subclass of LOD-based Similarity and the corresponding subclass of Linked data Manager. (vi) *Linked data Indexer* implements a mechanism for data indexing and caching to allow scalability and efficiency of LDS. (v) Finally *Linked data Benchmark* is used for testing and measures evaluations. The full architecture of LDS with detailed description of its components is provided in Part II, Chapter 4.

In its current version the library implements the following LOD-based similarity measures: LODS and its sub-measures SimI, SimC, and SimP discussed earlier, Linked Data Semantic Distance (LDSD) [135] and its extensions [135, 11], Resource Similarity (Resim) [137] and its extensions [11], and Partitioned Information Content Similarity (PICSS) [121].

In addition, to evaluate the ease of usage and extensibility of LDS, we proposed a new LOD-based similarity measure, Extended Partitioned Information Content Similarity (EPICS), which extends PICSS by considering an additional set of features called similar features, which considers the similarity between resources based on shared properties and directions. The evaluation results not only highlighted the potential of EPICS but more importantly the effectiveness of LDS as a core library for composing, implementing, and testing new similarity measures.

The work described in this subsection was conducted during the scientific stay of Nasreddine Cheniki at the University of Tours between 2016 and 2019 and during the PhD of Fouad Komeiha to be defended in 2023. More details will be given in Part II, Chapter 4.

2.2.2 Semantic and configuration-based contextualized and user-centric service interactions

Our contribution towards the challenge of contextualized and user-centric service interactions discussed in Section 2.1.2 is twofold following two user-service interaction perspectives: (i) service perspective and (ii) running device perspective. From the service perspective, we propose an approach based on Configuration Theory [151] for service composition into Mashups. The approach relies also on a set of visual artifacts as Domain

Specific Visual Language to allow end-user express preferences and constraints on the built Mashups. These artifacts are connected with a rule-based declarative language for consistent Mashup generation. From the running device perspective, we propose an architecture following Liquid Software principals [80] to allow for smooth service migration from one device to another from the user-side. We define semantic and rule-based migration context to ensure service interaction continuity while switching to new devices. These two contributions are summarized in the following sections.

2.2.2.1 Personalized and context-aware Mashups: A Configuration-based approach

As discussed in Section 2.1.2, high context-awareness is achieved with domain specific approaches. In this work, we tackle the problem of context-aware service interactions from the Tourism domain perspective and propose a declarative approach for Mashup building based on two main ingredients: (i) A visual intuitive language that allows user build an abstract Mashup and express constraints and preferences and (ii) a configuration-based technique to map the abstract Mashup with the relevant service instances while ensuring compliance with the expressed constraints and preferences. The overall approach is implemented within the CART system, a Web/Mobile framework for personalized trip planning in the Val-de-Loire French region.

Visual abstract mashups We propose a Domain Specific Visual Language containing a set of graphical artifacts with drag-and-drop actions and a set of simplified forms to express user constraints and preferences. The graphical artifacts correspond to the possible attractions, called Points of Interest (POI), that can be done during a touristic trip in the Val-de-Loire, typically Castles and Museum, Gardens and Open spaces, Events, etc. It also includes logistic elements such as Restaurants, Transport (public or car/bike rental). An abstract mashup is then the sequence of artifacts chosen by the user to define her/his trip. Additionally, preferences and constraints can be expressed through intuitive forms built upon the set of service properties. Typical global constraints can be the visit duration as a cumulative value of individual duration and the total budget the tourist should not exceed. Preferences can be expressed over the POI categories such as indoor vs outdoor, guided tour possibilities, etc. Additional contextual constraints such as weather, traffic, etc. are also captured.

The graphical artifacts are mapped to abstract services with corresponding functionalities. The sequence of abstract services is called the *mashup query*. The preferences and constraints are mapped to semantic rules and called the *configuration knowledge*. Both will be considered by the service configuration module to derive consistent Mashup in-

stance that provides relevant trip plans to the user. The resulting plans are also presented through a user friendly visualisation including Maps and additional representations of the trip steps.

Configuring service mashups The key point of the proposed approach is the configuration-based service mashup. We formalize the composition problem through the Configuration theory introduced in [151] under the property-based configuration model. In this model, a final configuration product is obtained through the aggregation of a set of property-value pairs using a set of pre-specified operators that allows maintaining the configuration in a coherent state by proposing possible compositions and pruning inconsistent ones. Following this model, we formalize the service mashup as a *Configuration Problem* $CP = (m, R, CK)$ where m denotes the mashup query, R denotes the set of context-aware rules, and CK denotes the configuration knowledge used to aggregate services.

The configuration knowledge describes how to compose service instances by mean of configuration operators and configuration rules. It relies on service's functionalities, property-value pairs, to build composition called functional connections. These functionalities are aggregated through configuration operators that basically apply cumulative operators (eg. addition) on common service properties and composition operators on distinct properties. Configuration rules are a set of restriction rules that guide the configuration process and limit the composition search space.

Configuration solutions are sequences of service instances corresponding to the abstract mashup, following the configuration knowledge, and satisfying the context-aware rules. Formally, a configuration solution is a pair $CS = (I, Q)$ where I refers to the set of items in the form (n, s) representing that the service s appears n times in the mashup to be configured and Q represents the set of mashup features (called quality) in the form of a pair (f, x) that defines the value x of the feature f. Q is progressively built from configuration knowledge CK and with respect to rules in R. The detailed formalization of the service configuration, the semantic matching between the abstract level and the instances, and the search algorithms will be provided in Chapter 5. There can be more than one valid configuration solution for a configuration problem. In this case it is possible to rank theses solutions based on ordering criteria defined on Q.

The CART framework for Personalized trip planning The proposed approach is implemented into a tool called CART (Configured mAshup Recommender application for personalized Trip planning) for personalized travel mashup. The CART's architecture is structured into 3 layers. The service Layer is in charge of collecting travel data from Web APIs in addition to DATAtourisme already used, eg. Google Maps, Google Direction,

Google Places, and Forsquare. The Mashup Layer implements the configuration approach as a SpringBoot Application. The application layer is implemented as a client application consuming services from the mashup application. It includes UIs allowing users to build a query mashup for the trip plan indicating their preferences and constraints. In addition, CART provides end-users with interactive environment to define mashup query and visualizes visit plan recommendations for the Loire Valley French region.

LOD-based mashup reconfiguration In CART, besides weather forecasting and transportation APIs, we have mainly exploited travel related-data representing places to visit , e.g., castles, museums, parks, events, restaurants and hotels, from Tourinsoft APIs in the context of Loire Valley French region (www.tourinsoft.com). Many other large data sets are however unexploited because they are sitting in isolated data silos. We propose extending the CART system with a semantic layer to constitute an integrated platform bringing together tourists data, locally events data, weather data, etc., from heterogeneous APIs. The extension relies on LODS similarity measure defined in Section 2.2.1.2. Following the LOD-based annotation and data linking, we first enrich the DATAtourisme ontology with contextual concepts from the LOD resources. We then apply LODS similarity measure to generate new service instance alternatives for the mashups already obtained by CART. We call this process a *reconfiguration* process.

The work described in this subsection was conducted during the PhD of Marwa Boulakbech defended on July 2020. More details will be given in Part III, Chapter 5.

2.2.2.2 User-side service migration in multiple-device contexts

In this work we tackle the challenge of providing fluent user-side service running management discussed in Section 2.1.2.2. We propose CUBE, a user-centric architecture following Liquid Software principals [80] to allow for smooth service migration from one device to another from the user-side. We then define semantic and rule-based migration context to ensure service interaction continuity while switching to new devices.

CUBE: a user-centric architecture for fluent service/data/session migration in multiple devices environment Our aim in this work is to provide users with user-centric framework that allows to synchronize user-service interactions over multiple devices and to perform smooth migration of such interactions between devices. Services are hosted and managed on Web servers or Cloud which usually allow for multiple device access. They however require the user to login each time a new device is used. Moreover, in most cases, users have to restart interaction sessions from scratch. Two typical scenarios can be considered to illustrate the need for alternative user-centric solution: a light-coupling

scenario with email service and tight-coupling scenario with YouTube service. Considering server-side synchronization, starting email writing on a device and then moving to a second device requires the login on both devices and saving the written part of the email as a draft from the first device to avoid data loss and to continue from the second device. Similarly, for video streaming on YouTube, user should login from both devices and resume video from saved history session on YouTube.

As alternative that allows time and data saving, we propose the CUBE, a user-centric framework that follows Liquid Software principals and defines a layered architecture built upon REST and RESTful constraints to ensure smooth service synchronization and migration in multiple device contexts from the user perspective. With CUBE, both previous scenarios will be handled locally preserving data and session states without the need for multiple login. The user logs in from the first device then the session can be moved to a second device through the use of RESTful properties allowing to manage data status. The multiple-device user context is virtualized for the server which continues interaction as it was with the first device.

In its abstract representation, the CUBE is seen as a four component model geometrically represented as: a central point which represents the user, surrounded by a first cube, called INNER CUBE, representing the set of user devices, contained in turn in a second cube, called the OUTER CUBE, representing the set of services, and between the two cubes the POOL AREA hosting the user-service interactions through the devices.

Formally, we define the CUBE as $\mathbb{C} = \{u, \mathbb{I}_c, \mathbb{O}_c, \mathbb{P}_a\}$. u denotes the user. $\mathbb{I}_c$ denotes the INNER CUBE defined as $\mathbb{I}_c = \{d_p, \mathbb{D}_u \setminus \{d_p\}, \mathcal{R}_d\}$ where d_p is the device being used by u, $\mathbb{D}_u$ is the set devices owned by u, and $\mathcal{R}_d$ denotes the research mechanism to discover new devices in the user context. $\mathbb{O}_c$ denotes the OUTER CUBE defined as $\mathbb{O}_c = \{s \in \mathbb{S}$ such that $\exists d \in \mathbb{D}_u$ and $d \leftrightarrow s\}$ where $\mathbb{S}$ is the set of services and $d \leftrightarrow s$ means that there is an interaction between u and service s through device d. And finally, $\mathbb{P}_a$ denotes the POOL AREA defined as $\mathbb{P}_a = \{d \leftrightarrow s$ such that $d \in \mathbb{I}_c$ and $s \in \mathbb{O}_c\}$.

The layered CUBE architecture is made of five different layers: (i) Application, (ii) Request, (iii) Conversation, (iv) Data, and (v) Physical. It is based on an innovative approach with strong specification of principals and constraints of the REST and RESTful models to allow data fluidity and content adaptation to devices characteristics. The detailed descriptions of this architecture and each layer are provided in Chapter 6. We will also provide technical details on the CUBE implementation and illustration on real use of the system.

Semantic rule-based device recommendation for service-migration in multiple device environment A successful migration decision, whether it is done automatically

or manually, requires to follow some criteria to know which target device is the most suitable for the service/application that will be migrated. Also, it has to capture the users preferences, timeline and other features related to security, reliability, compatibility of devices, constraints of the services, etc., in order to be able to suggest the right device. Although this issue has been widely covered from the cloud/server perspective as part of cloud resource management, efficient device recommendation is still a challenge from the user-side perspective. It is even more challenging as far as we consider the limited device computation performances and the lack of access to large semantic resources, such as Linked Open Data, for similarity calculation.

To address this challenge, We enhance CUBE with a declarative module that relies first on ontologies and semantic inference rules to ensure device recommendation. The approach is built upon three main contributions: (i) we first model the main concepts describing devices, services, users and contexts into a small OWL ontology that can be held on each device independently. (ii) Second, we define a set of typical semantic inference rules, in Semantic Web Rule Language (SWRL), formalizing the migration requirements based on the concepts of the ontology. (iii) Third, we implement the approach within a prototype that populates the ontology based on APIs and user interactions, applies a reasoning through the semantic rules, and infers device recommendations for service migration.

(i) OWL ontology core concepts: We conceptualize 4 main components: *Device*, *Owner*, *Service*, and *Characteristic*. Each of them contains specific sub-classes to handle service migration context diversity. We also define necessary properties to model the interactions and relationships between theses concepts. For example, the Owner class has two object properties with the device class: a device is *ownedBy* an owner, and the owner *usesDevice*. These two properties are different because a user can own several devices but only uses some of them at a time. Another property is: the user *interactsWith* the service. Furthermore, in order to define the relationship between the devices and their characteristics or the services with their requirement, we add two main properties. *hasCharacteristics* property: links between Device and Characteristic, while the *requireCharacteristics* property: links between Service and Characteristic.

(ii) Formalizing migration rules with SWRL: Based on the defined concepts and properties, we formalise service migration context and conditions through a set of inference SWRL rules. For example,
R1: $Device(?d) \wedge HasBattery(?d, ?b) \wedge swrlb : greaterThan(?b, 10) \rightarrow Alive(?d, true)$
defines the condition to consider a device as alive whereas,
R2: $Device(?d1) \wedge Device(?d2) \wedge HasLocation(?d1, ?l) \wedge HasLocation(?d1, ?l) \rightarrow InProximity(?d1, ?d2)$
expresses under which condition two devices are considered to be in proximity.

Migration rules are usually a combination of other rules. For example R4 which requires R1, R2, and R3 is an example of migration rule expressing conditions to achieve a successful migration.
R3: $Device(?d) \wedge HasScreen(?d, ?c) \wedge WatchingService(?s) \wedge RequiresScreen(?s, ?c) \rightarrow IsCompatiblewith(?d, ?s)$.
R4: $Device(?d1) \wedge Device(?d2) \wedge Service(?s) \wedge Alive(?d2, true) \wedge InProximity(?d1, ?d2) \wedge IsCompatibleWith(?d2, ?s) \rightarrow Migrate(?s, ?d2)$.

(iii) Implementation: The proposed approach is implemented within an API. The ontology is designed using Protégé and the SWRL rule-based reasoning is performed through Apache Jena. The API manages and stores the facts and rules (both defined in Apache Jena) to finally compute the recommendation results. Facts are very dynamic and change depending on the situation and the devices. They are stored in a ".n3" file. Some characteristics are automatically added by the app from device APIs (Battery, Connectivity, etc.) while others are filled in manually by the user. A set of basic common rules is provided by the app for each device. It can then be enriched by the user to handle specific migration scenarios. Additional details on the prototype and the evaluation is provided in Chapter 6.

The work described in this subsection was conducted during the PhD of Clay Palmeira da Silva defended on September 2019 and during the Master 2 Internship of Dimeth Nouicer defended on December 2020. More details will be given in Part III, Chapter 6.

2.2.3 Machine Learning and Blockchain to handle dynamic and trustworthy service interactions and evolution

2.2.3.1 Handling service evolution with Machine Learning and Linked Open Data

This contribution addresses the challenge of handling dynamic evolution of service interactions discussed in Section 2.1.3.1 through an approach combining semantically-enabled models and machine learning tools on social interaction data.

Following the graph-based social network representation of Web services in [111] and [97], we define a three-phase process to deal with service continuous deployment and changes. (1) *The Bootstrapping Phase* performs two tasks: (i) Semantic Annotation associates a profile to each service and annotates it with LOD resources; (ii) Semantic Matchmaking bootstraps relationships for newly deployed and services undergoing updates by running LOD semantic matching of service capabilities. (2) *The Updating Phase* updates service relationships by gathering and analyzing Web service interactions and their user feedback using sentiment analysis. (3) *The Learning phase* leverages Web services inter-

action experience to learn new open data links that will be reused in further semantic matchmaking, hence improving the accuracy of relationship recommendation.

Formally, the multi-relation network modeling Web service interactions is represented as a weighted directed graph $G = (V, R)_t$ where V is a set of services, and $R = \bigcup_{k=1}^{k=3} R_k$ is a set of edges, where $R_k = \{(S_u, S_v) \in V \times V\}$ is the set of edges of the k-th relationship. Major services interaction relationships are: *substitution* (R_1), *composition* (R_2), and *subscription* (R_3). t is the date time when the graph G is observed.

(1) The Bootstrapping Phase: This phase relies on LODS similarity measure and LOD-based service annotation discussed in Section 2.2.1.2. Services profiles are formally represented as pairs of features (F, D) where F is a text description of service functionality and D is a list of description keywords. Following the LOD-based annotation process, service features are linked to a set of LOD resources and LODS similarity measure is applied to compute similarity between pair of services in V. Similarity values are then considered to evaluate how likely two services can yield one of the three defined types of relations and compute a probability for successful interaction. Detailed algorithms are given in Chapter 7.

(2) The Updating Phase: Although semantic matching of service profiles allows building relevant business relationships, it relies however entirely on Web service profile data as isolated components and underlooks their interactive behaviour. The aim of this phase is to consider user rating scores for observed service interactions to update the graph accordingly. User feedbacks are expressed through text comments. We apply *sentiment analysis* to distinguish between positive comments and negative comments observed in a time interval $[t, t + \delta t]$. The updated weight of a relation between two services S_u and S_v is computed as the average of the previous weight and on the ratio of positive feedbacks over all the feedbacks. Formally, $w_{t+\delta t}(S_u, S_v) = avg\left(\frac{|positiveFeedbacks(S_u,S_v)_{t+\delta t}|}{|allFeedbacks(S_u,S_v)_{t+\delta t}|}, w_t(S_u, S_v)\right)$. The higher will be positive user feedbacks on the relationship, the higher will get the updated weight. The weight will decrease on the contrary case.

(3) The Learning phase: This phase exploits the highly positively evaluated relationships to enrich LOD with new corresponding relationships. Updating LOD is triggered when the gap between the updated weight value and the bootstrapped weight value gets above a given threshold for the same relationship. If the updated weight is greater than the bootstrapped one, this means that Web services have a strong record of interactions in practice, while bootstrapping phase has not recommended them as such, which proves the lack of information in LOD. Thus, new relationships among resources describing these interoperable Web services should be added to LOD. On the opposite, if the updated weight is the lowest, this means that Web services struggle to interact. The cause can be either Web service description is inadequate with service real capabilities, or Web service suffer

from QoS limitations. LOD relationships among resources describing such services should be destroyed.

The work described in this subsection was conducted during scientific stay of Hamza Labbaci at the University of Tours between 2018 and 2019. More details will be given in Part IV, Chapter 7.

2.2.3.2 Blockchain and smart contracts for trustworthy execution of business processes in dynamic contexts

The contribution discussed hereafter tackles the challenges discussed in Section 2.1.3.2 regarding the need for transactional flow modeling and change propagation model for blockchain-based choreographies and gives an overview of a threefold declarative approach we propose which includes: (i) a transformation model for mashups into smart-contract choreographies executions, (ii) a flexible transactional business process model for smart-contract business process executions, and (iii) a change propagation model to handle dynamic evolution of business process choreographies.

Mashup to smart contract transformation model. The starting point of this work is to go a step further configuring service mashups in the tourism context discussed in Section 2.2.2.1 to propose a way for personalized trip realization. The deployment and execution of touristic itineraries usually involve many independent entities (tourist, museum, restaurants, hotels, transport, etc.) that cooperate and compete in untrusted distributed context. In order to ensure trust within cooperation to guarantee a successful execution of touristic itinerary, we propose a two steps transformation model: (a) from data mashup itinerary to process choreography (BPMN) and then (b) from BPMN to smart contract. The CART trip planning outputs itineraries following an XML Schema Model with a root element containing a sequence of steps with or without PoIs and with or without payment. It also contains user constraints such as total duration and maximal budget. PoIs may also have additional attributes. The detailed structure of the itinerary generic model will be given in Chapter 8.

The main transformation rules are given in Table 8.1. We first apply a set of transformation rules which associate: a choreography model for each itinerary instance (rule (a)). Within the choreography model we generate a participant pool for every PoI and for the user (rule (b)) as well as a choreography task for each step of the itinerary (rule (c)). Steps without PoI correspond to private tasks (rule (d)). Additional user constraints and PoI attributes are embedded in the model documentation and in the PoI process documentation respectively (rules (e) and (f)).

The resulting choreography BPMN file is in turn used to generate corresponding Smart

Table 2.1: XML to Solidity code transformation rules

Rule	**XML element**	**Choreography element**	**Solidity code**
(a)	Root element: Touristic_Visit	Touristic_Visit choreography model	Touristic_Visit Interaction Smart Contract (TVISC)
(b)	POI	participant (POI) pool	participant address
(c)	Step containing POI	choreography task =send/receive tasks +messages flows	functions in TVISC
(d)	Step with no POI	private task	-
(e)	Tourist's constraints (budget, total visit duration, etc)	embedded in choreography model documentation	Global variable in the TVISC
(f)	POI attributes (fees, opening hours, etc)	embedded in the POI process documentation	Result of the callback performed by Touristic_Visit oracle contract

contracts in Solidity code using the second part of the transformation rules in Table 8.1. Only interaction activities are captured and included in the produced Smart contract which encodes only the public view (rule (a)). Each interaction activity, i.e., choreography task, consists of a send task, receive task and the corresponding message flows. This is encoded as two functions in the Smart contract (rule (c)). We consider Hyperledger Fabric which is a permissioned blockchain where only authorized organizations has the access to the network. In our case, authorized organizations are participant pools in the choreography model (rule (b)). Private tasks are left to participant private process and are therefore not included in the Smart contract (rule (d)). Additional constraints and attributes are encoded as global variables (rule (e)) and callbacks (rule (f)).

Extending Smart contracts with transactional business process model. The generated interaction smart contract implements the business logic of interaction activities in the process choreography, that is the control flow. The transactional semantics of a smart contract follow ACID models and are therefore limited to support transactional requirements of business process executions [22]. In order to ensure execution reliability of the generated smart contracts we extend them with the Transactional Business Processes models [22, 60] which define the transactional flow on top of the control flow of the interaction activities. Transactional Business Processes models define transactional properties of activities: *pivot*, *compensatable*, and *retriable* and the transactional mechanisms which include failure recovery mechanisms. In addition, we enrich the life cycle of a process activ-

ity, *enabled* → *started* → *completed* with four other possible termination states which are *aborted*, *cancelled*, *failed* and *compensated* which we call transactional termination states.

In this contribution we extend the transformation model introduced in the previous section and generate both the control flow and the transactional flow as well as failure handling mechanisms. For each business process activity we also generate functions and variables in the smart contract as a transactional flow. Generated functions include lifecycle activities (*ActivityName_ start()*, *ActivityName()*, *ActivityName_ complete()*, and *ActivityName_ fail()*) and transition activities (*abort()*, *cancel()*, *complete()*, *fail()*, and *retry()*) which allow the activity to move from one state to another. Variables include state vector *StateVector_ ActivityName* to capture the activity state evolution for each activity. The extended transformation model is implemented on top of the Caterpillar tool [107].

Change propagation in Blockchain-enabled business process choreographies. Collaborating business entities evolve in highly dynamic environments that appeal for continuous updates and improvements of their internal processes to keep high competition level. These dynamic changes may affect the collaborations in which the enterprise takes part. Changes may also directly concern the global collaboration due to new external business policies that should be taken into account. In both cases, changes must be notified and propagated to the involved partners. Thus, business process management systems should manage change propagation in a trustworthy fashion. Although change propagation in process choreographies has been tackled in general [65], it is however still challenging for blockchain-based choreographies because of the non-mutable nature of shared data.

In running choreography instances, a change may consist of a simple operation, ADD/REMOVE/UPDATE activity, or a combination of operations. Handling such changes requires the underlying choreography model to be flexible. We therefore consider declarative choreographies modeled through Dynamic-Condition-Response (DCR) graphs [127, 128]. With a DCR graph $G = (E, M, Rel)$, processes are modelled as a set of *events* E, corresponding to the activities in BPMN, linked together with *relations* Rel modeling linking constraints. Relations are of five types (*condition*, *response*, *milestone*, *include*, and *exclude*) [85]. The graph runtime state is captured by a marking matrix M of the events through a triplet of binary values corresponding to (*currently included*, *currently pending*, *previously executed*). In [84], authors propose an approach for trustworthy execution of DCR choreographies. We briefly describe here our contribution which builds on and extend this work with the change management mechanism. Detailed formalization and illustrative scenario will be provided in Chapter 8.

In [84] a DCR choreography is defined as $C(G, I, R)$ where G is a DCR graph, I is a set of *interactions* and R is a set of *roles*. An interaction i is a triple (e, r, r') in which

the event e is initiated by the role r and received by the roles $r' \subset R \setminus \{r\}$. Each partner has a projection of the DCR choreography, called public DCR process, in addition to its private DCR process which refines the public process with local events of the partner. The shared events and interactions are managed through the choreography smart contract on the Blockchain.

We propose to extend this approach with change management through the choreography smart contract. We define the change element G_{Ref} as a refinement of C which occurs on private DCR process and should be propagated to the public DCR process whenever the change impacts common interactions with other partners. G_{Ref} can be either an atomic element (an event or an interaction relation) or a DCR subgraph. When G_{Ref} impacts common interactions, we manage a negotiation phase that leads either to a consensus in which case the public process is updated accordingly or to failure in which case the change is rejected. A change operation can be of three types respectively corresponding to ADD, REMOVE, and UPDATE: $C \oplus G_{Ref}$, $C \ominus G_{Ref}$, and $C[G_{Ref} \mapsto G'_{Ref}]$. We describe hereafter the four-steps approach we propose to manage these changes.

(i) Change proposal. The role initiator defines the change of its private DCR process following the previously introduced possible change types and operations. We review a set of integrity rules [58] to ensure safety and liveness of the updated DCR graph to avoid cycles. Safety means that the DCR graph is deadlock free whereas liveness guarantees the ability of the DCR graph to completion. The integrity rules are of two types. *Allowed* change rules define what can be done on the graph while *Denied* ones define what should be rejected. When the change proposal satisfies these rules the private DCR process is updated. In the case where the change also concerns the public DCR choreography we move to the next steps.

(ii) Change request for public-related changes. The change request is sent the choreography smart contract which stores the list of change requests assigned to process instances. Ongoing process instance changes are recorded with the identification hash of the current process instance. During the change request lifecycle, the request is assigned to a status: *Init* if no change request is ongoing, *BeingProcessed* during the negotiation stage, *Approved* or *Declined* once the change request is processed by all endorsers. The smart contract checks the identity of the change initiator which should belong to the list of partner, checks that no other change request is being processed, and then creates the change request and sends change request notification to the involved partners. The change initiator also sets two response deadlines t1 for change endorsement and t2 for change propagation to be checked by the smart contract. If one of the change endorsers does not reply before deadline t1 during endorsement or t2 during propagation, an alarm clock triggers a smart contract function cancelling the change request. The smart contract function sets the

change request status to cancelled and emits an event notifying partners that the change has been cancelled. In this way we prevent any deadlock that could occur due to one of the partners not responding.

(iii) Change negotiation for public related changes. All partners subscribe to the change request events emitted by the smart contract. Endorsing partners must send their decision request to the smart contract based on the *Allowed/Denied* changes rules. If the change once computed on the endorser's process respects all the rules, then the endorser approves the request. It is otherwise rejected. The smart contract collects all the decisions from the endorsers to lock (or not) the choreography instance and proceed (or not) with the change and update the change status to *Approved* or *Declined*. In case of approval the smart contract locks the instance for change propagation. As it manages the negotiation process, a tamper-proof record of the negotiation is accessible by all partners. This prevents conflicts and eases potential claim resolutions.

(iv) Change propagation. Change propagation is to apply the change effect after the negotiation phase succeeds to (i) the affected partners DCR public processes, (ii) each partner propagates the change effect to its private DCR process. To ensure the correctness of the change propagation, we introduce a correctness property that checks the safety and compatibility of the updated graphs. The smart contract receives all the local projection confirmations and notifies the change initiator. That latter forwards the new DCR choreography to the smart contract which updates the relations and resets the change status to allow for new changes.

The work described in this subsection was conducted during the PhD of Amina Brahem to be defended by the end of 2023. More details will be given in Part IV, Chapter 8.

Part II

Semantically enabled service interactions

Chapter 3

Semantic Web Service Composition Using Semantic Similarities and Formal Concept Analysis

Contents

This chapter describes our first contribution regarding the challenge of semantically-enabled Web service interactions discussed in Section 2.1.1. It provides a detailed view of the approach described in Section 2.2.1.1 which consists in an integrated framework for semantic web services. The main contribution is threefold: a semantic similarity measure for Web services, a flexible FCA-based Web service classification for Web service discovery, and a planning and graph-based composite service generation.

3.1 Introduction

One of the most interesting and relevant properties of services is the possibility to combine a number of existing services to create a more general composite service. Composing services allows for the definition of increasingly complex applications by progressively combining services at increasing levels of abstraction [59]. This composition pattern has received much interest to support business-to-business or enterprise application integration. Several efforts have led to the development of platforms and languages to support composition and deployment of services [134, 17, 149, 103]. However, despite this considerable progress, the composition process still poses limitations and challenges which have yet to be addressed by current technologies and tools for Web service composition. For instance, considerable differences on structural, semantic and technical levels along with the growing number of available Web services makes their discovery a significant challenging task. Achieving a required composite functionality requires the discovery of a collection of Web services out of the enormous service space. Each service must be examined to verify its provided functionality, making the selection task neither efficient nor practical [14].

Moreover, using services without considering their underlying semantics, either functional semantics or quality guarantees can negatively affect the composition process by raising intermittent failures or leading to slow performance [166]. Indeed, semantics play important roles in Web process life-cycle. They specify the detail semantic information about the underlying functions supported by a service. Another important aspect is related to services compatibility as an essential pre-requisite to service composition. Measuring the similarity of services is an important and valuable task to get useful information about their compatibility. Similarity measure can be considered as an optimization step before composing services since it enables to reduce the search time by functionally classifying similar services.

In this context, this chapter presents a practical approach to measure the similarity of Web services. Both semantic and syntactic descriptions are integrated through specific techniques for computing similarity measures between services. Formal Concept Analysis (FCA) is then used to classify Web services into concept lattices, and therefore generate a hierarchy of classes of similar Web services. Following this step, a composition engine takes as inputs the set of similar services and the specification of the required service, and generates the candidate composition plans that realize the goal through Planning-based and graph-based algorithms.

The remainder of this chapter is organized as follows. Section 2 introduces the components of the composition framework and details their components. Section 3.2 provides an overview on the framework architecture. Section 3.3 details the proposed semantic

similarity measure for Web services. Section 3.4 details the FCA-based classification of Web services and the use of related structure for service discovery. Section 3.5 details the planning-based and the graph-based composite service generation. Section 3.6 provides insights on the framework implementation and the conducted experiments. Section 3.7 reviews related work. Finally Section 3.8 concludes the chapter.

3.2 Semantic Web Service Composition Framework overview

In this section we introduce the Integrated Development Environment for Composite Services Engineering (IDECSE) framework. The proposed framework is intended to achieve a high abstraction level allowing the continuous and dynamic improvement of the composition to support and encourage the adoption of SOA technologies while reducing costs and composition time. Based on a layered approach, the main general objective is to develop a well-integrated framework to enable the mastery of complexity and dependability of service compositions by offering tools for specifying, discovering and composing services. All service life-cycle processes in the proposed approach are based on native semantic integration. IDECSE general architecture is depicted in figure 3.1. It consists mainly of three complementary components: the semantic matching engine, the service classification engine, and the composition engine. The service matching process which plays the central role as it underlies and supports service discovery and composition processes through providing successful matching services possibilities among all the services in the service space. The goal here is then to define efficient similarity measures which allow ensuring an accurate matching process. The defined similarity measure allows classifying Web services into sets of highly similar ones. Achieving the previous processes will make it easy to define efficient and optimized algorithms for relevant service discovery. Finally, the composition process which is based on efficient tools for ensuring meaningful compositions and dynamically interacting with the other framework modules towards preserving composition continuity. These components will be detailed in the following sections.

IDECSE framework also provides a user-friendly interface to enable end users to specify the high-level description of their desired service. They express their requirements and preferences that mark the boundary of the solution by following some instructions to build the query. A query parser is integrated to parse and validate the query by checking syntactic correctness and decoupling the functional requirements from the non-functional parameters such as QoS properties.

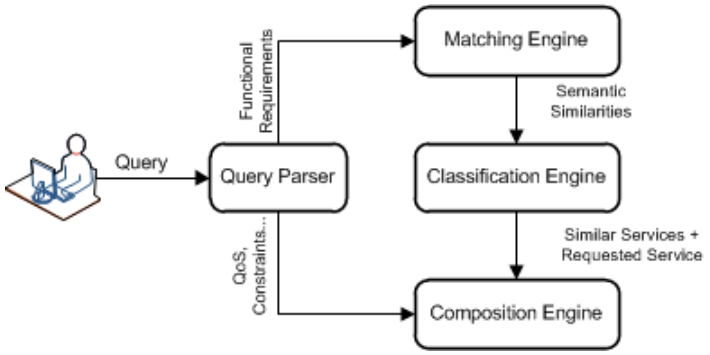


Figure 3.1: Overview of the IDECSE Framework

3.3 Semantic similarity measure for Web services (Service Matching Engine)

The Service Matching Engine is responsible for measuring the similarity between semantically annotated OWL-S services. We define a similarity measure σ over a set S of services as follows:

$$\begin{aligned} &\sigma : S \times S \rightarrow [0\,,1] \\ &\sigma(S_i, S_j) = \alpha_1 Sim_P(S_i, S_j) + \alpha_2 Sim_M(S_i, S_j) + \alpha_3 Sim_G(S_i, S_j) \end{aligned} \tag{3.1}$$

where $\sum_{k=1}^{3} \alpha_k = 1$.

Sim_P, Sim_M and Sim_G correspond to similarity function between Profiles, Models and Groundings, respectively. As Service Model and Service Grounding are used to supplement the initial similarity and specify details of how to access a service, their parameters can be modeled as QoS. In this case, similarity is based on functional features located in the Service Profile of the service OWL-S file.

$$\begin{aligned} \sigma(S_i, S_j) &\approx Sim_P(S_i, S_j) \\ &= w_1 Sim_I(S_i, S_j) + w_2 Sim_O(S_i, S_j) + w_3 Sim_{Pre}(S_i, S_j) + w_4 Sim_{Eff}(S_i, S_j) \end{aligned} \tag{3.2}$$

where $\sum_{k=1}^{4} w_k = 1$.

Sim_I, Sim_O, Sim_{Pre}, Sim_{Eff}, correspond to the similarity function between Inputs, Outputs, Preconditions and Effects of compared services, respectively. The weights w_i allow modeling the similarity measure to be adapted to different application cases where it may be required to focus on IO only or any other possible combination of the four aspects. In the case where no specific preferences are given, these weights are equally set to 0.25.

In the general case, weights are either expressed by some actor (the user for example) to define personalised preferences regarding the application domain or the actor expectations, or empirically deduced through learning techniques over previous computing scenarios.

Each of the four Sim functions is in turn defined as the sum of two weighted concept-based and data-type based measures as follows for the case of Inputs and analogously for the three other dimensions:

$$Sim_{In}(S_i, S_j) = u_{In_1} Sim_{Co}(In_i, In_j) + u_{In_2} Sim_{DT}(In_i, In_j) \tag{3.3}$$

where $u_{In_1} + u_{In_2} = 1$.

Web service features are semantically annotated using concepts from ontologies. Measuring their similarity turns out to measuring the similarity between their annotation concepts in the considered ontologies. Comparative study of the state-of-the-art approaches on semantic similarity measures have shown a better correlation coefficient and higher efficiency rates for the measure proposed in [145] which follows the Tversky model [157], where common taxonomical features tend to increase similarity while the non-common ones decrease it. We therefore adapt the Sanchez formulas to define Sim_{Co}.

Let A and B two concepts, represented by the nodes a and b in a predefined *is-a* semantic taxonomy. Let C be a set of concepts of such a taxonomy; $(\leq)$ is defined as a binary relation $\leq$: C $\times$ C. Having two concepts c_i and c_j, $c_i \leq c_j$ is fulfilled if c_i is a hierarchical specialization of c_j or if $c_i \equiv c_j$ (i.e. same concept). The set of taxonomical features describing the concept a is defined in terms of the relation $\leq$ as follows (Equation 3.4).

$$\phi(a) = \{c \in C | a \leq c\} \tag{3.4}$$

[145] defines the semantic similarity between two concepts as follows:

$$Sim(a,b)_{Co} = 1 - \log(1 + \frac{|\phi(a) \backslash \phi(b)| + |\phi(b) \backslash \phi(a)|}{|\phi(a) \backslash \phi(b)| + |\phi(b) \backslash \phi(a)| + |\phi(a) \cap \phi(b)|}) \tag{3.5}$$

In the same way, we reviewed and adapted the best state-of-the-art data-type measures to define Sim_{DT}. Following works in [139] and [154] we propose a practical measure between different data-types that defines abstract groups of compatible data-types and then assigns similarity values to each pair of groups. Datatype groups similarity defined in [139] is summarized in 3.1.

Considering Sim_{DT} prevents meaningless composite services built upon highly similar services with respect to Sim_{Co} which however fail to match their data-types. Following

Table 3.1: Simple DataType Groups Similarity([139])

	Int.	**Real**	**Str.**	**Date**	**Bol.**
Int.	1.0	0.5	0.3	0.1	0.1
Real	1.0	1.0	0.1	0.0	0.1
Str.	0.7	0.7	1.0	0.8	0.3
Date	0.1	0.0	0.1	1.0	0.0
Bol.	0.1	0.0	0.1	0.0	0.1

this intuition, the weight u_{In_1} must be higher than u_{In_2} to focus on semantic similarity while keeping data-type similarity for checking the composition consistency. Conducted experiments confirmed that the combination $u_{In_1} = 0.80$ and $u_{In_2} = 0.20$ gave the highest accuracy.

Finally, we define our similarity measure as follows (Equation 3.6):

$$\begin{aligned}\sigma(S_i, S_j) = {} & w_1[u_{I1}Sim_{I_{Co}}(S_i, S_j) + u_{I2}Sim_{I_{DT}}(S_i, S_j)] \\ & + w_2[u_{O1}Sim_{O_{Co}}(S_i, S_j) + u_{O2}Sim_{O_{DT}}(S_i, S_j)] \\ & + w_3[u_{Pre1}Sim_{Pre_{Co}}S_i, S_j) + u_{Pre2}Sim_{Pre_{DT}}(S_i, S_j)] \\ & + w_4[u_{Eff1}Sim_{Eff_{Co}}(S_i, S_j) + u_{Eff2}Sim_{Eff_{DT}}(S_i, S_j)] \end{aligned} \tag{3.6}$$

where $\sum_{k=1}^{4} w_k = 1$, $\sum_{j=1}^{2} u_{aj} = 1$ and $a \in \{\text{I,O,Pre,Eff}\}$.

The proposed similarity measure will be used for flexible FCA-based service classification as well as for Graph-based composition generation detailed in the following sections.

3.4 Flexible Web service classification and discovery with FCA (Service Classification Engine)

The way services are organized within the service registry highly impacts the efficiency of service identification for any given need. We therefore propose to build the classification component of IDECSE upon the well-established formalism called Formal Concept Analysis (FCA) [71] and particularly its derived structures called Concept Lattices. We illustrate the FCA-based classification and discovery on a set of semantic services in Table 3.2 extracted from a public dataset [1]. For each service we focus on the operations and the annotation concepts of its inputs and outputs.

Based on partially ordered sets theory, FCA allows to structure the data into a hierarchy of partially ordered concepts called the Concept Lattice. Each concept corresponds to a maximal set of objects having in common a maximal set of attributes. The concept lattice

[1] http://projects.semwebcentral.org/projects/sws-tc

Service	S_{Id}	**Operation**	Op_{Id}	**Input**	**Output**
Author_Finder	S_1	Author_Find(Book)	op_{11}	#Book	#Author
Author_Price _Finder	S_2	Find_Author(Book) Find_Price(Book)	op_{21} op_{22}	#Book #Book	#Author #Price
Search_Book	S_3	Search_Book(Author)	op_{31}	#Author	#Book
Hotel_Cost	S_4	Find_Hotel(Time,Place)	op_{41}	#Date, #Address	#Hotel, #Price
Book_Hotel	S_5	Book_Hotel(Date,Hotel)	op_{51}	#Date #Hotel	#Reservation
Hotel_Reservation	S_6	Hotel_Reservation(Time Hotel)	op_{61}	#Date #Hotel	#Reservation
IP_to_Country	S_7	IPtoCountry(IPaddress)	op_{71}	#IP	#Country
IP_Map	S_8	ExtractPlace(IPaddress)	op_{81}	#IP	#Address

Table 3.2: An example of semantic Web services in the registry

hierarchical relation is based on the dual partial order between sets of objects and dually sets of attributes. This structure and its properties motivated the use of concept lattices as indexing structure for information retrieval either by direct querying or by progressive navigation [118, 119].

Following this direction we apply FCA to automatically organize services into a concept lattice based on the semantic similarity defined so far. We consider services as objects and operations as attributes. We compute the similarity between service operations based on σ in Equation 3.6. Considering the set of services in Table 3.2, this step yields the Service-Operation matrix in Table 3.3. The matrix is symmetric and only the top part is filled in for visibility.

Table 3.3: Service-Operation similarities

Serv. \ Op.	op_{11}	op_{21}	op_{22}	op_{31}	op_{41}	op_{51}	op_{61}	op_{71}	op_{81}
S_1	1	1	0,419	0.353	0.203	0.192	0.152	0.295	0.273
S_2		1	1	0.353	0.203	0.192	0.152	0.295	0.273
S_3				1	0.237	0.238	0.238	0.204	0.280
S_4					1	0.483	0.483	0.175	0.175
S_5						1	0.824	0.142	0.142
S_6							1	0.142	0.142
S_7								1	0.714
S_8									1

In FCA, this representation is called Multi-valued formal context formally represented as a quadruplet (G, M, W, I) where G is a set of objects, M is a set of attributes and $(g, m, w) \in I$ means that attribute m has value w for object g. FCA basic settings,

however, apply on binary contexts formally represented as a triple (G, M, I) where G is a set of objects, M is a set of attributes and $(g, m) \in I$ means that object g has attribute m. We apply the conceptual scaling on the Multi-valued context based on a similarity threshold $\theta \in [0\,, 1]$. Similarity values below the threshold are removed and the remaining ones are replaced with "$\times$", which yield the context in Table 3.4 for $\theta = 0.4$. The corresponding concept lattice is given Figure 3.2 (Left).

Table 3.4: Service-operation formal context for $\theta = 0.4$

Serv. \ Op.	op_{11}	op_{21}	op_{22}	op_{31}	op_{41}	op_{51}	op_{61}	op_{71}	op_{81}
S_1	X	X	X						
S_2	X	X	X						
S_3				X					
S_4					X	X	X		
S_5					X	X	X		
S_6					X	X	X		
S_7								X	X
S_8								X	X

The concept lattice structures the services in the registry into classes of services with similar operations. It can then be used as underlying structure for identifying relevant services for a given need: substitution of similar service or composition of complementary services. Relevant service identification can rely on best-known algorithms for FCA-based querying and navigation [118, 119]. In the case of querying, an illustration is given with a requested service S_R which allows to find an *Author* as *String* given a *Book Title* also as *String*. S_R is modeled as one-operation service, similarity measure σ is applied to compute similarity values of S_R with respect to registry services and the corresponding scaled context line is added to the context. The corresponding concept lattice is given in Figure 3.2 (Right).

Selected relevant services can then be returned as query answer for individual invocation or more generally they can be considered as input of the composition engine for composite service generation. This step will be detailed in the following section.

3.5 Planning and graph-based composite service generation (Service Composition Engine)

The last component of the IDECSE framework is the composition generator. It takes as input a set of similar services returned by the previously presented components to provide a set of coherent composite services relevant to requested function. This component appeals for AI planning techniques and graphs together with the semantic similarity measure

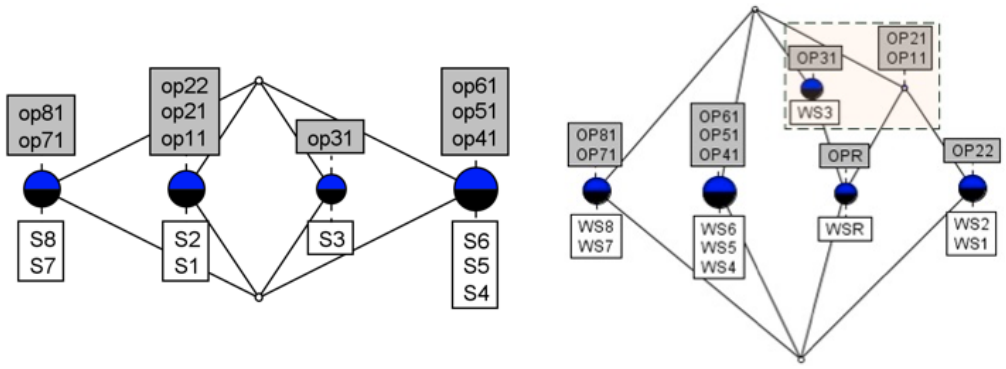


Figure 3.2: Service concept lattice corresponding to the formal context in Table 3.4 (Left) and with the requested service S_R (Right)

and QoS constraints to ensure successful compositions. The key point is the compatibility between Input-Output interfaces of the services to be connected within the resulting composite ones.

3.5.1 Planning-based composite service generation

The highest level of compatibility is achieved with identical matching between Input-Output which feats the use of AI Planning. In this case, Web service composition problem is mapped to a planning problem based on transformation rules defined in [82] and using the Planning Domain Definition Language (PDDL) [75]. In this mapping, each service S_i is represented as a state St_i, its inputs and preconditions are represented as the state preconditions $prec(St_i) = S_{i_{Inputs}} \cup S_{i_{Preconditions}}$, and its outputs and effects are represented as the state effects $eff(St_i) = S_{i_{Outputs}} \cup S_{i_{Effects}}$. We apply STRIPS planning algorithms on the resulting mapping problem to generate possible solution plans corresponding to possible service compositions. In the general case, however, there may be no exact Input-Output interface matching. We therefore propose a graph-based composition generation which relies on the proposed semantic similarity measure to compute approximate matching possibilities.

3.5.2 Semantic Similarity for Graph-based composite service generation

Graph-based composite service generation consists in modeling the set of selected services as a composability graph: a weighted directed graph where nodes correspond to services and edges to their composability similarity values. Composability similarity between two services S_i and S_j is the matching similarity between S_i outputs and S_j inputs to evaluate how likely the two services can be connected. Similarity measure σ in Equation

3.6 is redefined as follows.

$$\sigma_{Composability}(S_i, S_j) = u_1 Sim_{Co}(S_{i_{Out}}, S_{j_{In}}) + u_2 Sim_{DT}(S_{i_{Out}}, S_{j_{In}}) \quad (3.7)$$

where $S_{i_{Out}}$ actually denotes the union of $S_{i_{Outputs}}$ and $S_{i_{effects}}$ and $S_{j_{In}}$ actually denotes the union of $S_{j_{Inputs}}$ and $S_{j_{Preconditions}}$.

The obtained similarity values are mapped logical match type as defined by [100] and illustrated in Table 3.5.

Table 3.5: Semantic Composability Matching

Composability value	$[0, 0.3[$	$[0.3, 0.6[$	$[0.6, 0.8[$	$[0.8, 1]$
Logical Match Type	Fail	Subsume	Plug-in	Match

Services with "*Match*" or with "*Plug-in*" are most likely to lead to successful composition. Composability graph can be reduced to keep only services with these two composability match types. To illustrated the graph building steps we consider the set of selected services in Table corresponding to the use-case of getting access to restaurant menus given the user IP address. The steps of composability graph generation are given in Figure 3.3.

Table 3.6: Example of selected services for composability graph building

Services	S_{Id}	**Input**	**Output**
IP_City	S_1	IP_address	City
IP_Restaurant	S_2	IP_address	Restaurant
Find_Restaurant	S_3	City	Restaurant
Restaurant_Menu	S_4	Restaurant	Menu
Requested_Service	S_{R1}	IP_address	Menu

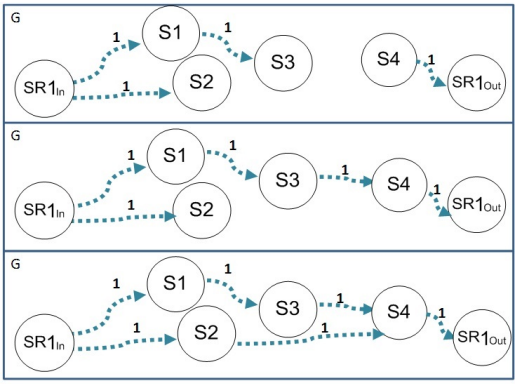


Figure 3.3: Composability graph generation steps

We formalize satisfiable compositions as S_{comp} ={I_{comp}, O_{comp}} where I_{comp} is a set of

inputs that present a valid degree of match with the set of requested Inputs $S_{R_{In}}$ and O_{Comp} is a set of outputs that present a valid degree of match with the set of requested Outputs $S_{R_{Out}}$. Then, service composition solutions are those corresponding to paths in the graph between $S_{R_{In}}$ and $S_{R_{Out}}$. The graph search is enhanced by considering QoS constraints over services to fine-tune composition solutions ranking. In this example, in addition to S_2 as individual relevant answer to the requested service, the composition $S_1 - S_3 - S_4$ is returned as relevant composite answer. A more detailed formalization and illustration of the graph building and search algorithms is given [2].

3.6 Implementation and evaluation

The proposed approach is implemented as a Java application including an OWL-S parser based on Jena [2] and JAXP[3] and FCA algorithm based on ConExp implementation[4]. The prototype is evaluated on two datasets: a set of 250 OWL-S services available online [5], mainly for accuracy, and a set of 1000 randomly generated services, for performance study.

An overview of evaluation results is given hereafter. More detailed experimental study is provided in [2]. We start by evaluating the required time to build the IDECSE registry which is the sum of execution time of three main tasks: parsing service features, measuring similarity, and building lattice of services. Results for the first dataset (250 OWL-S services) are given in Figure 3.4 (Left) and show acceptable performances for this step built before user interactions. Then we moved to requested service matching for which results are given in Figure 3.4 (right). The indicated time includes query processing, S_R similarity measure, and lattice querying to identify the set of relevant services.

The composition generation evaluation results are shown on figure 3.5 (Left) showing also acceptable time to get composition plans including important number of composable services. The generation of a plan with 7 services, 6 dependencies and 14 parameters for each service takes less than one second. Plans with more than 20 services takes less than two seconds to be generated.

Finally, the overall performances of the proposed approach is compared to two other existing prototypes corresponding to state-of-the art approaches: PORSCE presented in [82], and 2P presented in [93] and results are shown on Figure 3.5 (Right). Experiments are run on a randomly generate dataset of 1000 Web services. Services parameters are uniformly distributed and randomly assigned to services. Then, we generate 100 queries with parameters from the set of service parameters to keep the same semantic domain. For

[2]https://jena.apache.org
[3]http://docs.oracle.com/javase/tutorial/jaxp/
[4]https://sourceforge.net/projects/conexp/
[5]http://projects.semwebcentral.org/projects/sws-tc

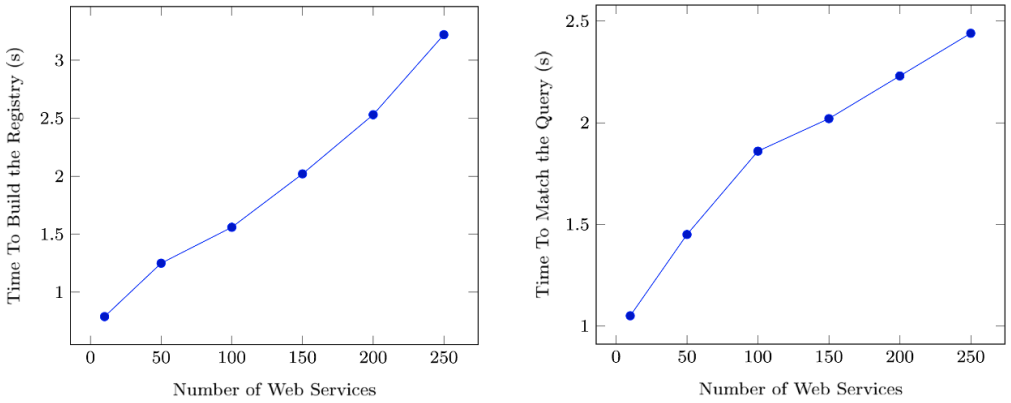


Figure 3.4: (Left) Time to build IDECSE registry. (Right) Time to match a query.

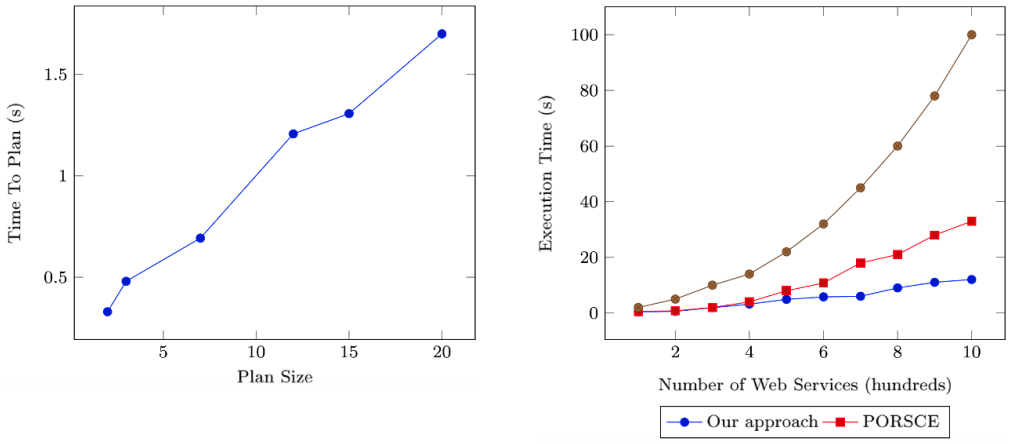


Figure 3.5: (Left) Time to generate composition plans. (Right) Performance comparison with PORSCE and 2P.

composition generation, we consider until 3-service plans. Results show the efficiency of the proposed approach.

3.7 Related Work

In this section we briefly review the state-of-the-art approaches related to our approach in order to position our contribution accordingly. A more detailed and extended review of related work is presented in [2] and [7]. Among the numerous proposed approaches, some surveys in [134] and [103], we will particularly focus on those which perform service clustering and similarity computing for automating service composition.

Web service clustering was discussed in several approaches. [61] perform service clustering based on WSDL extracted contents such as types, messages, ports, and service name.

[171] present a hybrid Web service tag recommendation strategy that employs tag mining and semantic relevance measurement. Web services are clustered according to the similarities of extracted WSDL features. [44] present the service cluster net unit used then to check whether any service in the cluster can satisfy user requirements for further use in service substitution process. One more specific service clustering family groups those based on FCA. [13] use FCA to classify services according to keywords extracted from WSDL service descriptions. [129] proposed a service ranking algorithm for diversifying web service discovery results in order to minimize the redundancy in the search results using FCA.

Some other approaches rely on semantic similarity measures or the definition of semantic queries using SPARQL language for service discovery. [154] consider WSDL descriptions to provide a practical approach for measuring the similarity between Web services by using a set of existing lexical and semantic metrics. Similarly, [105] employ external knowledge to compute the semantic distance between terms from two compared services. While [16] propose an approach based on modeling the available services into RDF views over a mediated ontology. The generated views are then integrated as annotations into WSDL files. User queries are transformed into a mediated ontologies using SPARQL query languages. [126] introduce a SPARQL-driven approach for searching relevant services to be addressed to service repositories for further automatic service composition.

Regarding service composition integrated frameworks, [81] extend their AI-based planning composition framework using semantic metrics to discover and reason about services. Graph-based approaches such as [101] develop an integrated framework for dynamic Web services composition. The framework exploits the semantic Input-Output matchmaking to discover relevant services and perform automatic composition using a graph-based approach, taking into account functional and non-functional properties. [55] support both automatic semantic discovery and composition, among other relevant phases of the composition life-cycle taking into account non-functional properties. [142] develop a framework for discovery and composition which includes optimizations to reduce graph size and an optimal search to extract the best composition from the graph. One of the limitations of the discovery phase is that it does not support fine-grained services retrieval.

Although Web service discovery and composition approaches have presented efficient and performing techniques, they still face challenging tasks, which straightforwardly impact the composition process and the efficiency of composition frameworks. First, providing an accurate service matching requires to consider the maximum number of available features describing services, syntactically and semantically, while defining efficient similarity measures. Second, the discovery process performances and accuracy are highly dependent on how Web services are structured and/or categorised into considered registries. Existing FCA-based approaches suffer from lack of semantics and are limited only to basic common

features between services. Considering semantic similarity at this level reduces the research space of relevant services to a specific cluster or class of services instead of the whole registry. Third, the composition process depends mainly on two elements. The the list of selected services and the composition algorithm. Fourth a successful framework should ensure the matching, classification, discovery, and composition processes. Unfortunately, the literature review stresses the lack of such integrated frameworks. Discovery and composition of Web services have indeed been usually developed separately. Also, matching methods always rely on similarity measures independently defined for abstract concepts, semantically annotating services, regardless services additional features. In this work, we addressed these tasks through the proposed integrated framework considering semantics in all stages of service life-cycle, defining efficient similarity measure that also considers data type compatibility, combining semantic similarity with FCA for service clustering and discovery, and using AI planning and Graph-based efficient algorithms for composition generation.

3.8 Conclusion

In this chapter, we presented a integrated approach for Web services discovery and composition using semantic similarity measure which combines semantic features together with data type compatibility. Formal Concept Analysis was used for building Web service lattices according to functional and operational service similarities. Generated classes of similar services are then used to identify the candidate composition plans that realize the desired goal based efficient planning and graph-based algorithms. The approach is implemented and evaluated over two data sets and compared to state-of-the-art baselines. Results show the approach efficiency and performances which outperform baselines.

The key point of this work is semantic similarity measure for OWL-S services which considers ontological hierarchies. The measure efficiency is therefore domain-dependant, limiting its applicability in general cases where services may refer to complementary domains. The next logical step is therefore to consider cross-domain ontologies. This objective meets the emergence and recent advances in Linked Open Data (LOD). In the next chapter, we take advantage of LOD to define efficient cross-domain similarity measure to be applied for service recommendation.

The work described in this chapter was conducted during the PhD of Ahmed Abid defended on July 2017 [2]. It is mainly published in the following journal and conference papers: [4], [5], [6], [3], [8], and [7].

Chapter 4

Linked Open Data Similarity for Web services

Contents

This chapter describes our second contribution regarding the challenge of semantically-enabled Web service interactions discussed in Section 2.1.1. It provides a detailed view of the approach described in Section 2.2.1.2 which consists in exploiting the Linked Open Data (LOD) for efficient service recommendation. The main contribution is threefold: a LOD-based semantic similarity measure, a context-aware service recommendation, and a Library for LOD-based similarity calculation.

4.1 Introduction

The continuous improvement of mobile network performances together with the significant evolution of hardware and software of mobile devices has deeply transformed the way users interact with information systems. A new era of computing, called "ubiquitous computing", based on these two evolutions, has emerged. Nowadays, a mobile user has an ubiquitous access to multiple resources, in particular services. In order to satisfy user needs in terms of functional and non-functional service requirements, discovery process have to be applied. This process performs at first stage a functional matchmaking to return services having functionalities that user requested. At the second stage, non-functional parameters (contextual information) are taken into consideration to rank services depending on user's task or situation. Incorporating context information during discovery process reduces improves the discovery accuracy by filtering results that are functionally relevant but do not correspond to user context.

Despite the interest of using ontologies in context-aware mobile services discovery, they present in our opinion two major weaknesses. First, they are resources limited and its knowledge covers generally restricted domains. Second, they are generally isolated, i.e., their concepts are not interconnected with concepts in other ontologies, which limits their exploitation. It is not possible for example to enrich concept definition in one ontology from an equivalent one defined in another ontology because of missing links. These two limitations implicitly impact any ontology-based service discovery process.

Researchers over the past few years have addressed this issue in two different ways. Either by replacing domain specific ontologies by multi-domains ontologies such as WordNet or by using textual knowledge sources such as Wikipedia to bootstrap service discovery[21]. However, both directions have drawbacks. Although multi-domains ontologies are relatively rich, they remain non-interconnected. Textual knowledge sources also suffer from the concepts interconnection problem. In addition, they require exhaustive processing to calculate relevance. We are thus convinced that, the use of data from Web of Data, commonly called (Linked Open Data)[23], will make it possible to overcome the previous limitations. These data are mainly intended to be processed by machines, and therefore designed and thought to be open, structured, semantically described and linked to each other.

In this chapter we takes advantage of the evolution of Semantic Web to Linked Open Data (LOD) to propose, in Section 4.2, an efficient similarity measure, LODS, that assesses the matching degree between pairs of terms represented with LOD resources. We then use LODS measure in an approach that addresses context-aware service recommendation, described in Section 4.3. We describe, in Section 4.4, the open source LOD-based similarity measure library we propose on top of the LODS. We provide in Section 4.5 an insight on

the evaluation and results of the proposed approach. We review in Section 4.6 the main related work. Finally we conclude the chapter in Section 7.8.

4.2 LODS: a Linked Open Data Similarity Measure

4.2.1 Background

In this section, we provide the main necessary constructions and characteristics of LOD and DBpedia to be used in the description of our proposed similarity measure.

RDF triples. Consider a set of URIs $\mathcal{U}$ and literals $\mathcal{L}$, an rdf triple is defined as $\langle s, p, o\rangle$, where the subject $s \in \mathcal{U}$, the property $p \in \mathcal{U}$ and the object $o \in \mathcal{U} \cup \mathcal{L}$.

Linked Open Data. A dataset that follows a linked data principles [23] is a graph $G = (\mathcal{R}, L)$, where $\mathcal{R} = \{r_1, r_2, ..., r_{|\mathcal{R}|}\}$ is a set of resources and $L = \{l_1, l_2, ..., l_{|L|}\}$ a set of links. l_i is defined as $l_i = \langle r, p, r'\rangle \vee \langle r, p, v\rangle$, where p is a property that interlinks the resource r with the internal/external resource r' or with a literal attribute v, which is a basic value (string, date, number ...). So, Linked Open Data is a set of interlinked open datasets $LOD = \bigcup_i G_i$.

Ontology. An ontology is a graph of triples that describe domain concepts and their relations. In LOD, it is preferred to use concepts from widely used ontologies to instantiate its resources [83]. This allows efficient data integration and reuse by LOD-based applications. We denote by O all ontologies used to describe the whole LOD resources.

Classification schema. Classification schemata classify resources into categories and may contain cycles in their taxonomic structure. This requires to limit the level of extracted categories to avoid retrieving useless ones. We denote by C all classification schemata used to classify the whole LOD resources.

Triple patterns, Basic Graph Patterns (BGP). Triple patterns are similar to RDF triples except that each of the subject, predicate or object may be a variable (started with '?'). BGP are constructed from a set of triple patterns. We adopt SPARQL BGP[1] to represent queries over RDF datasets.

Properties types. An infinite set of properties could be used to describe LOD resources. In this work we consider the 4 following property types.

- Instantiation properties (IP). An IP $\tau \in \mathcal{U}$ maps a concept c from an ontology $o \in O$ to a particular resource r, we write $\langle r, \tau, c\rangle$. *rdf:type* is an example of IP.
- Classification properties (CP). A CP $\varsigma \in \mathcal{U}$ is used to classify a resource into a particular category in a schema. *dcterm:subject* is an example of CP.

[1] http://www.w3.org/TR/sparql11-query/

- Linking properties (LP). An LP $\xi \in \mathcal{U}$ is used to interconnect two equivalent resources r and r' from distinct LOD datasets, we write $\langle r, \xi, r' \rangle$. *owl:sameAs* is an example of LP.

- Subsumption properties (SP). An SP $\delta \in \mathcal{U}$ is used to define specialization or subclass relation in a given ontology or classification schemata. Given two concepts c_i and c_j from an ontology, $\langle c_i, \delta, c_j \rangle$ denotes that concept c_i is a specialization of c_j. *owl:subClassOf* is an example of SP.

We call the remaining properties characterization properties, denoted by P, since they distinguish LOD resources. Properties that have a resource r as subject, i.e. $\langle r, p, ?o \rangle$, are called outgoing properties and those having r as object, i.e. $\langle ?o, p, r \rangle$, ingoing properties.

Property paths and paths patterns. We adopt SPARQL1.1 property path notations such as *ZeroOrMorePath*, *ZeroOrOnePath*, *OneOrMorePath*, *SequencePath*, *AlternativePath*, denoted respectively $*, +, ?, /$ and $|$. They are used to navigate between resources and reach particular data inside a single or distinct LOD datasets.

Using property paths in a triple that contains variables, expresses a path pattern that retrieves all triples satisfying it. For instance, the pattern : $\langle r, \xi/\tau, ?c \rangle$ retrieves all instantiation concepts c for the resource r from its equivalent resources. It is worth noting here that we consider property paths traversal over different LOD datasets.

DBpedia. DBpedia [102] is the semantic counterpart of Wikipedia that realizes LOD vision by structuring its content and interlinking it with external datasets such as Wikidata and YAGO. DBpedia transforms every Wikipedia article into a resource, annotated with a set of properties extracted from the article Web page. Many LOD datasets are producing data pointing to DBpedia resources making it as a central kernel of LOD cloud[2]. We therefore consider in this work DBpedia as starting point to glean initial data to compute similarity degree between compared concepts. Afterward, follows interlinks we enrich data from other datasets within a data augmentation process.

4.2.2 Similarity measure for Linked Data Resources

In the following we define a similarity measure composed of three sub-measures: $SimI$, $SimC$, and $SimP$. $SimI$ exploits the taxonomic structure of ontological concepts used to instantiate resources. $SimC$ operates on classification schemata used to categorize resources. $SimP$ uses characterization properties of compared resource. The combination of these sub-measures allows to reduce the negative impact of possibly poorly described resources. Each sub-measure applies on enriched data by following links between LOD

[2]http://lod-cloud.net/

dataset. The sub-measures are defined as feature-based similarity measures based on Tversky [157] model to handle multi-domain ontologies as well as multiple annotation ontologies for the same concept [145].

We provide hereafter the formalization of the proposed LOD based similarity measure. A more detailed presentation with illustrations on a running example can be found in [45].

Let $O_r \subseteq O$ be the subset of ontologies containing concepts that instantiate a resource r. We define a function $\phi_o(r)$ that returns all taxonomic features of a resource r, i.e. all concepts and their subsumers in an ontology $o \in O_r$. Formally,

$$\phi_o(r) = \{?c \in o | \langle r, \tau | \delta^* | \tau / \delta^*, ?c \rangle\} \tag{4.1}$$

To enrich taxonomic features or even increasing the size of instantiation ontologies space of a resource. We follow equivalent resources belonging to the same or to different LOD datasets. Formally, we define $\phi^*_o(r)$, an augmented function as follows.

$$\phi^*_o(r) = \phi(r) \cup \{?c' \in o \mid \langle r, \xi, r' \rangle \wedge \langle r', \tau | \delta^* | \tau / \delta^*, ?c' \rangle,\ ?c' \notin \phi(r)\} \tag{4.2}$$

After applying the function $\phi^*_o(r)$, the set of instantiation ontologies of r will be so augmented. We denote this new set by O^*_r.

Definition 1 (SimI). Let $O^*_{a,b} \subseteq O^*_a \cap O^*_b$ be the set of shared augmented ontologies between two resources a and b. The instantiation similarity $SimI^*_o(a, b)$ of two resources described with concepts from an ontology $o_i \in O^*_{a,b}$ is computed based on the cardinalities of differential and common taxonomic features of compared resources as follows.

$$SimI^*_{o_i}(a, b) = \frac{|\phi^*_{o_i}(a) \cap \phi^*_{o_i}(b)|}{|\phi^*_{o_i}(a) \cap \phi^*_{o_i}(b)| + |\phi^*_{o_i}(a) \setminus \phi^*_{o_i}(b)| + |\phi^*_{o_i}(b) \setminus \phi^*_{o_i}(a)|} \tag{4.3}$$

The overall instantiation similarity $SimI^*$ over the set of augmented ontologies $O^*_{a,b}$ is defined as follows.

$$SimI^*_{\forall o_i \in O^*_{a,b}}(a, b) = \frac{\sum_{o_i \in O^*_{a,b}} SimI^*_{o_i}(a, b)}{|O^*_{a,b}|} \tag{4.4}$$

We consider with $SimI^*$ the average of the resulted similarities from all shared augmented ontologies to hold the balance between ontologies having a rich taxonomic structure and those with a poor taxonomic structure.

Now, given a resource r classified within categories following a set of classification schemata $C_r \subseteq C$, we denote $\Delta^\ell_t(r)$ the function that returns, for a resource r, all the classification categories and their super-categories in a classification schema $t \in C_r$. The parameter ℓ limits the deepness of the hierarchical level in which categories are retrieved.

Formally,

$$\Delta_t^{\ell}(r) = \{?c_1, \ldots, ?c_\ell \in t | (\langle r, \varsigma, ?c_1\rangle \ \wedge \langle ?c_1, \delta, ?c_2\rangle) \vee ... \langle ?c_{\ell-1}, \delta, ?c_\ell\rangle\} \tag{4.5}$$

In the opposite, we denote $\nabla_t^{\ell'}(r)$ the function that returns all the classification categories and their sub-categories, for a resource r, in a schema $t \in C_r$, with respect to ℓ' levels Formally,

$$\nabla_t^{\ell'}(r) = \{?c_1, .., ?c_{\ell'} \in t | (\langle r, \varsigma, ?c_1\rangle \wedge \langle ?c_2, \delta, ?c_1\rangle) \vee ... \langle ?c_{\ell'}, \delta, ?c_{\ell'-1}\rangle\} \tag{4.6}$$

We combine, the two functions to obtain all classification features of a resource r as follows.

$$\varphi_t^{\ell,\ell'}(r) = \Delta_t^{\ell}(r) \cup \nabla_t^{\ell'}(r) \tag{4.7}$$

Following the same principle of augmenting the space of instantiation concepts of a resource r, we can define augmented classification features function denoted by ${\varphi_t^*}^{\ell,\ell'}(r)$. We can as well obtain its set of augmented classification schemata, denoted by C_r^*.

Definition 2 (SimC). Let $C_{a,b}^* \subseteq C_a^* \cap C_b^*$ be the set of shared augmented classification schemata between two resources a and b. The classification similarity $SimC_t^{*\ell,\ell'}(a,b)$ of two resources, described with categories from a classification schema $t \in C_{a,b}^*$ with respect to limited super-categories and sub-categories hierarchy levels ℓ and ℓ', is computed based on the cardinalities of differential and common categories features of compared resources as follows.

$$SimC_t^{*\ell,\ell'}(a,b) = \frac{|{\varphi_t^*}^{\ell,\ell'}(a) \cap {\varphi_t^*}^{\ell,\ell'}(b)|}{|{\varphi_t^*}^{\ell,\ell'}(a) \cap {\varphi_t^*}^{\ell,\ell'}(b)| + |{\varphi_t^*}^{\ell,\ell'}(a) \setminus {\varphi_t^*}^{\ell,\ell'}(b)| + |{\varphi_t^*}^{\ell,\ell'}(b) \setminus {\varphi_t^*}^{\ell,\ell'}(a)|} \tag{4.8}$$

The overall similarity for the whole augmented classifications space is then defined as follows.

$$SimC_{\forall t_i \in C_{a,b}^*}^{*\ell,\ell'}(a,b) = \frac{\sum_{t_i \in C_{a,b}^*} SimC_{t_i}^{*\ell,\ell'}(a,b)}{|C_{a,b}^*|} \tag{4.9}$$

As for properties, let the subset $P_r \subseteq P$ be the sub-space of properties that contains characterizing attributes of a resource r. We denote by $\Omega_{P_i}(r)$ (respectively, $\Omega'_{P_i}(r)$) the function that returns all ingoing (respectively, outgoing) characterizing properties of a resource r in a particular space $P_i \in P_r$. Formally,

$$\Omega_{P_i}(r) = \{(?p, IN), ?p \in P_i | \langle ?x, ?p, r\rangle\} \tag{4.10}$$

$$\Omega'_{P_i}(r) = \{(?p, OUT), ?p \in P_i | \langle r, ?p, ?x\rangle\} \tag{4.11}$$

$$\Psi_{P_i}(r) = \Omega_{P_i}(r) \cup \Omega'_{P_i}(r) \tag{4.12}$$

Following interlinked equivalent resources of r, we can also define as in equation 4.2 function $\Psi^*_{P_i}$ that returns the set of augmented properties describing r. The augmented space of properties P^*_r can then be obtained.

Definition 3 (**SimP**). Let $P^*_{a,b} \subseteq P^*_a \cap P^*_b$ be the space of shared augmented characterizing properties of two resources a and b, we define characterizing properties similarity $SimP^*_{P_i}(a,b)$ of a and b as:

$$SimP^*_{P_i}(a,b) \;=\; \frac{\mu(\Psi^*_{P_i}(a) \cap \Psi^*_{P_i}(b))}{\mu(\Psi^*_{P_i}(a) \cap \Psi^*_{P_i}(b)) + \mu(\Psi^*_{P_i}(a) \setminus \Psi^*_{P_i}(b)) + \mu(\Psi^*_{P_i}(b) \setminus \Psi^*_{P_i}(a))} \tag{4.13}$$

where μ is the partial information content of characterizing properties [121]. This function gives more importance to specific properties based on their occurrences in LOD datasets. Formally,

$$\mu(\Psi^*_{P_i}(r)) = \sum_{\forall p \in \Psi^*_{P_i}(r)} -\log\left(\frac{Freq(p)}{N}\right) \tag{4.14}$$

where $Freq$ is a function that counts the occurrence frequency of the property p in the description of LOD resources, and N the total number of resources in the underlying dataset. The overall similarity of all properties space is defined as follows.

$$SimP^*_{\forall P_i \in P^*_{a,b}}(a,b) = \frac{\sum_{P_i \in P^*_{a,b}} SimP^*_{P_i}(a,b)}{|P^*_{a,b}|} \tag{4.15}$$

Based on the previously defined sub-measures, we define our LOD-based similarity measure LODS as follows.

Definition 4 (**LODS**). Given a two resources a and b that are: (i) instantiated with concepts from augmented shared ontologies $o_i \in O^*_{a,b}$, (ii) classified into categories from augmented shared classification schemata $t_i \in C^*_{a,b}$, and (iii) characterized with a set of augmented shared space of properties $P_i \in P^*_{a,b}$. We define the Linked Open Data Similarity ($LODS$) measure between the resources a and b as follows.

$$LODS^{\ell,\ell'}(a,b) \;=\; AVG(simI^*_{\forall o_i \in O^*_{a,b}}(a,b), simC^{*\ell,\ell'}_{\forall t_i \in C^*_{a,b}}(a,b), simP^*_{\forall P_i \in P}(a,b)) \tag{4.16}$$

4.3 LOD-based context-aware service discovery

4.3.1 Context-aware service annotation model

Considering functional and non-functional aspects, a service s can formally be represented as a triplet $\langle n_s, \mathcal{F}_s, \mathcal{NF}_s \rangle$, where: n_s is the service name, $\mathcal{F}_s = \{f_1, f_2, \ldots, f_i\}$ represents the functionalities of s, and $\mathcal{NF}_s = \{P_{1,s}, P_{2,s}, \ldots, P_{n,s}\}$ represents its non-functional properties.

Functionalities in $\mathcal{F}_s$ can be annotated by a subset of resources $A_s \in LOD$ where: $\forall f_i \in \mathcal{F}, \exists r_i \in A_s | \{\langle f_i, \alpha, r_i \rangle\}$. α being an annotation property that attaches a LOD resource to a service functionality.

Non-functional properties in $\mathcal{NF}_s$ may be of diverse types. They are therefore generally organized in profiles, where each profile $P_{i,s}$ describes a specific context information domain. Properties inside profiles can be of two types: quantitative and qualitative. Quantitative properties have generally numerical values while qualitative properties have string type values that can be attached to LOD resources. This annotation is modeled as follows: for a property p with set of values $\{v_1, v_2, \ldots, v_i\}$, a set of resources $r_i \in LOD$ can be attached to them. Formally, $\exists p = \{v_1, v_2, \ldots, v_i\} \in P_{i,s} \quad | \quad \langle v_i, \alpha, r_i \rangle, r_i \in LOD$.

As an illustration example, let's consider an hotel service named n_s="Le Plantagenêt" with only one feature $\mathcal{F}_s = \{Hotel\}$. The annotation set of this service is $A_s = \{dbp : Hotel\}$. It has one DBpedia resource[3] attached to the functionality *Hotel*. This service has a profile that describes service preferences as non-functional properties: $\mathcal{NF}_s = \{P_{preferences,s}\}$, where $P_{preferences,s} = \{TotalCapacity = 33; Ranking = 3, ComfortServices = dbr{:}Internet, SpokenLangues = \{French,English\}\}$.

4.3.2 Functional matching of services

The first step of contextual services discovery considers service functional annotation resources. Considering a user request defined as of a set of terms $R = \{t_1, t_2 \ldots t_i\}$ where each term can be attached to a subset of annotation resources $A_R \in LOD$ such that $\forall t \in R, \exists r_i \in A_R | \langle t_i, \alpha, r_i \rangle$, the similarity between the request R, annotated by A_R, and service s, which functionalities are annotated by A_s is defined based on LODS measure as follows:

$$Sim_{\mathcal{F}}(R, s) = \frac{\sum_{a \in A_R} \sum_{b \in A_s} LODS^{\ell,\ell'}(a, b)}{|A_R| \times |A_s|} \tag{4.17}$$

$Sim_{\mathcal{F}}$ is a classical aggregation measure [79] that allows the comparison of two objects annotated with semantic concepts. It computes the scores obtained from applying LODS

[3] dbp is a prefix that represents the URL https://dbpedia.org/resource/

measure on each combination of the Cartesian product of the two compared sets. Then, it divides the sum by the number of combinations to obtain a similarity value in [0,1].

4.3.3 Non-functional matching of services

At this second matchmaking stage, we first use semantic rules to eliminate services that do not satisfy mandatory user requirements. Then, a profile-based similarity measure is applied to the filtered services in order to measure their relevance to the user profile. Considering the variety of information that can be found in a contextual environment, each type of information is represented in a separate profile structure. For instance, we can find a profile for user preferences, a second for user device specifications, a third for information about surrounding environment, etc. These information can either be explicitly provided by the user/service-provider or implicitly leaned by the system. Formally, for a user u and a service s, the set of profiles describing user/service non-functional properties in a particular context situation is $P_{1,X}, P_{2,X}, ...P_{n,X}$ where $X \in \{u, s\}$. Each profile is described by a set of properties $P_{i,X} = \{x_1, x_2, ..., x_n\}$. We compute the non-functional similarity between u and s profiles as follows:

$$Sim_{\mathcal{NF}}(u, s) = \lambda_1 f(P_{1,u}, P_{1,s}) + \lambda_2 f(P_{2,u}, P_{2,s}) + ... + \lambda_n f(P_{n,u}, P_{n,s}) \tag{4.18}$$

where λ_i is the weight expressing the importance of the profile in this particular context. The function f computes the similarity between two profiles. It is based on measure defined in [18]. We extend the latter to take into account the different types of data present in user/service profiles. We integrate our LODS similarity measure into the extended measure in order to evaluate the quantitative information present in profiles. Formally, consider $P_{i,u} = \{x_1, x_2, ..., x_n\}$ and $P_{i,s} = \{y_1, y_2, ..., y_m\}$ two profiles describing respectively a user u and a service s properties in a particular context situation, we define the similarity $P_{i,u}$ and $P_{i,s}$ as follows:

$$f(P_{i,u}, P_{i,s}) = \frac{a \sum_{i=1}^{i=N} w_i \times ASim(x_i, y_i)}{a \sum_{i=1}^{i=a} w_i + b \sum_{i=1}^{i=b} w_{a+i} + c \sum_{i=1}^{i=c} w_{a+b+i}} \tag{4.19}$$

where, **a** is the number of common properties of X and Y, **b** is the number of properties belonging to X but not to Y, and **c** is the number of properties belonging to Y but not to X. Each property has a weight w_i, where $\sum_{i=1}^{n} w_i = 1$. The atomic measure *ASim* is calculated depending on the property types. It uses LODS for LOD properties and typical data type specific measures for other cases such as *String*, *Numerical*, *intervals*, and *Sets* properties. A more detailed formalization of *ASim* is given in [48].

The proposed LOD-based context-aware service recommendation approach was applied

in the Tourism domain within the Smart Loire[4] research project as described in Section 4.5.

4.4 LDS library

In order to go further than the LODS similarity measure, we propose LDS (Linked Data Similarity) Library as an open source Java software library of LOD-based semantic similarity measures. LDS is available at *https://github.com/FouadKom/lds.git*. LDS defines a generic, reusable, and extensible architecture that integrates the best-known existing LOD-based similarity measures and supports both developers and researchers who intend to use these implemented measures or develop their own new ones.

LDS development follows the following requirements: (i) *Simple design* to facilitate the reuse of the tool and provide developers the ability to extend and use it with their developed semantic measures; (ii) *Efficient LOD querying, caching and indexing* to ensure similarity calculation efficiency and scalability; and (iii) *Extensibility* to offers developers a way to implement rapidly their measures and incorporate the library with their work.

The LDS architecture shown in Figure 4.1 has 5 main components: (i) *Similarity Engine* which acts as the interaction interface of LDS with external programs. (ii) *LOD-based Similarity* which calculates the similarity values. (iii) *Linked data Manager* which retrieves data for similarity calculation. These two components act as plugability layer allowing to easily integrate new similarity measure as subclass of LOD-based Similarity and the corresponding subclass of Linked data Manager. (vi) *Linked data Indexer* implements a mechanism for data indexing and caching to allow scalability and efficiency of LDS. (v) Finally *Linked data Benchmark* is used for testing and measures evaluations. The full architecture of LDS with detailed description of its components is provided in [91].

In its current version the library implements the following LOD-based similarity measures: LODS and its sub-measures SimI, SimC, and SimP discussed earlier, Linked Data Semantic Distance (LDSD) [135] and its extensions [135, 11], Resource Similarity (Resim) [137] and its extensions [11], and Partitioned Information Content Similarity (PICSS) [121].

In addition, to evaluate the ease of usage and extensibility of LDS, we proposed a new LOD-based similarity measure, Extended Partitioned Information Content Similarity (EPICS), which extends PICSS by considering an additional set of features called similar features, which considers the similarity between resources based on shared properties and directions. The evaluation results not only highlighted the potential of EPICS but more importantly the effectiveness of LDS as a core library for composing, implementing, and testing new similarity measures.

[4] https://smartloire.univ-tours.fr

A more detailed presentation of LDS as well as detailed performance study are given in [91]. LDS served as underlying framework for similarity calculation in a Privacy-aware IoT Device Recommendation approach we propose and describe in [92].

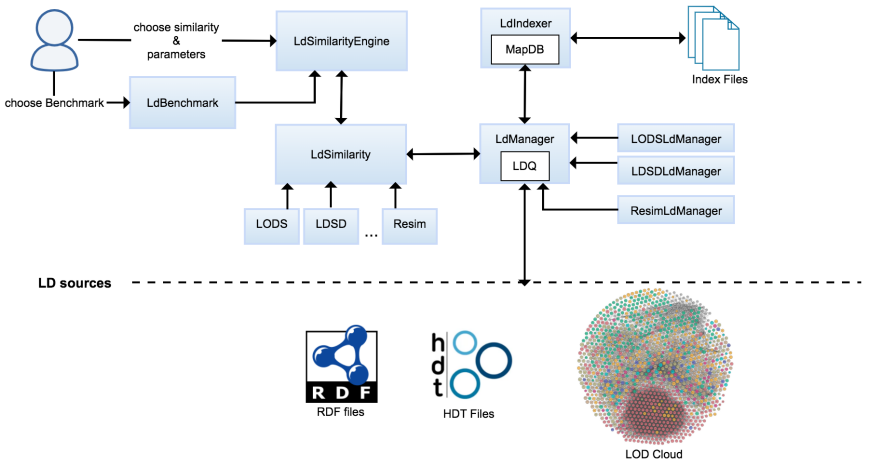


Figure 4.1: Overview of the Architecture of *LDS*.

4.5 Implementation and Evaluation

This section will briefly describe the application of the LOD-based context-aware service recommendation approach in the Tourism domain and the performance study of LODS measure and LDS library. Detailed experiments can be respectively found in [48], [46], and [91].

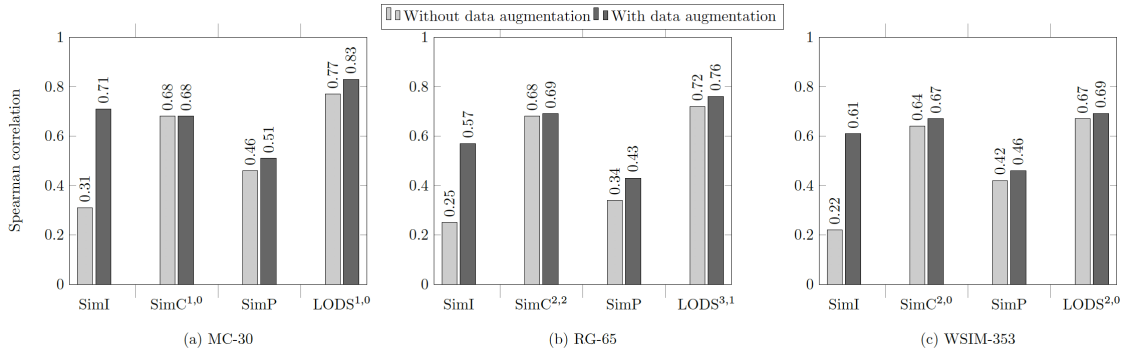
4.5.1 LODS similarity measure evaluation

LODS similarity measure is implemented using Java and Jena framework[5]. We obtain required data directly from provided SPARQL endpoints as much as possible. Unreachable data via SPARQL endpoints or HTTP principles are downloaded and hosted on a local endpoint. We rely on DBpedia knowledge base (DBp) as a primary source of semantic linked data. We then use interlink relationships to navigate through and get more rich data from Wikidata (WD), YAGO knowledge base ($YAGO$), and three different DBpedia chapters. We consider most active chapters: Dutch (DBp_{de}), Italian (DBp_{it}), and French (DBp_{fr}). Therefore the considered LOD dataset is $LOD = \{DBp, WD, YAGO, DBp_{de}, DBp_{it}, DBp_{fr}\}$. The considered ontology space is $O = \{O_{dbo},$

[5]https://jena.apache.org/

$O_{yago}, O_{schema}, O_{umbel}, O_{wd}\}$. The classification schemata is $C = \{C_{DBp}, C_{DBp\_de}, C_{DBp\_it}, C_{DBp\_fr}\}$. And, finally, the property set is $P = \{P_{DBp}, P_{DBp\_de}, P_{DBp\_it}, P_{DBp\_fr}, P_{WD}\}$.

The similarity measure is applied on three well-known benchmarks: MC-30 that contains 30 pairs of concepts [123], RG-65 [143] that contains 65 pairs of concepts, and similarity gold-standard of WSIM-353[6] collection. Results are given in Figure 4.2 and show high accuracy of LODS combined measure on LOD augmented data.



(a) MC-30 (b) RG-65 (c) WSIM-353

Figure 4.2: Evaluation of LODS and sub-measures.

4.5.2 LOD-based context-aware service recommendation in the context of Smart Loire project

Smart Loire[7] is a research project funded by the region Centre Val de Loire with the main objective of cultural tourism promotion in Loire Valley through the development of contextualized and personalized Trip planning framework. The work described in the current chapter is partly supported by the Smart Loire project. We therefore consider to evaluate the proposed approach on the project dataset. The dataset is provided by ADT37 (Agence Départementale de Tourisme d'Indre-et-Loire) through its platform Tourinsoft (http://www.tourinsoft.com/). The dataset contains 1632 services classified into different touristic offer categories and annotated with functional and non-functional properties. A selection of categories is given in Table 4.1 and an example of service description is given in Figure 4.3.

Following the proposed approach steps, we annotated each service with a set of LOD resources considering *Classification properties* as functional properties and the remaining ones, such as *PrestationsEquipements*, *Location*, and *Payment Modes*, as non functional properties. We then proposed to a total of 22 users to express queries about tourist services and collected the most common keywords in Table 4.2.

[6] http://www.cs.cmu.edu/ mfaruqui/word-sim/EN-WS-353-SIM.txt
[7] https://smartloire.univ-tours.fr

Table 4.1: Selected touristic offers

Touristic Offer categories
Hotels and restaurants
Historic Sites and Monuments
Parks and Gardens
Museums
Interpretation centers
Collective accommodations
Leisure and social service activities facilities
Cultural and sporting activities

Table 4.2: A selection of the 9 most frequent queries.

Q	**French keyword**
R_1	Hôtel
R_2	Restaurant
R_3	Hébergements locatifs/collectifs
R_4	Chambres d'hôtes
R_5	Auberges de jeunesse
R_6	Auberge de campagne
R_7	Gîte
R_8	Site et monument historiques
R_9	Château

We the consider these queries to run the functional service discovery algorithm implementing Equation 4.17. For each query we consider the 10 best ranked services. The precision average is 1 for the first 3 services for all the queries, between 0.8 and 0.97 for the 4th to the 7th services and higher than 0.68 for the remaining services. Thes rates show the accuracy of the proposed approach.

Moving to the non functional step, we selected 5 users who were asked to define their profiles in terms of preferences over non-functional service properties and to specify as example of mandatory constraint the maximum geographic distance in which services are retrieved (2km, 5km, 10km, 20km, 60km). Other examples of non-functional properties are given in Table 4.3. We consider the same weight (w_i) for all the properties. We apply non-functional constraints, the algorithm implementing Equation 4.18, on the results in the previous step and measure the context precision as the number of remaining non filtered services divided by the number of services of the previous step. Context precision for each of the 5 users is shown in Figure 4.4 (Left). We also evaluate the overall quality of the discovery compared to user evaluation of the returned results. Figure 4.4 (Right) shows that the context-aware service recommendation process quality is very close to user rating.

Figure 4.3: Example of a touristic service description

```
<service>
    <title>Hostellerie du château de l Isle</title>
    <m:properties>
        <d:SyndicObjectName>Hostellerie du château de l Isle
                </d:SyndicObjectName>
        <d:GmapLatitude>47.3282285</d:GmapLatitude>
        <d:GmapLongitude>1.0458362</d:GmapLongitude>
        <d:ObjectTypeName>Hôtellerie</d:ObjectTypeName>
        <d:ClassementsllPrefectoral>3 étoiles</d:ClassementsllPrefectoral>
        <d:LanguesParlees>Français#Anglais</d:LanguesParlees>
        <d:CapaciteTotale>12</d:CapaciteTotale>
        <d:CodePostal>37150</d:CodePostal>
        <d:Classification>Hôtels - restaurant</d:Classification>
        <d:PrestationsEquipements>Boulodrome#Jardin#Jeux pour enfants
              en extérieur#Jeux pour enfants#Parc#Parking#Parking privé
              #Restaurant#Salle de réception#Salon#Télécopieur#Terrasse
        </d:PrestationsEquipements>
        <d:Web>http://www.chateau-delisle.com</d:Web>
        <d:ModesPaiements>American Express#Carte bleue#Chèques de voyage
              #Chèques bancaires et postaux
        </d:ModesPaiements>
    </m:properties>
</service>
```

Table 4.3: Properties considered in non-functional evaluation

Property	**Type**	μ	**Values Examples**
Spoken language	Set	AtLeast	French, English
Payment method	Set	AtLeast	Blue Card, Cash
Ranking	Number/Interval	-	3 stars, [2-4] stars
Provided services	LOD	Max	dbr:Parking, dbr:Internet

4.6 Related work

In this section we briefly review the main approaches close to our contribution on LOD based similarity measure and its use to context-aware service discovery. A more detailed review of the state-of-the-art is given in [46], [48], and [91].

The emergence of LOD and its rapid expansion through continuous interconnection of new resources motivated the its use as underlying semantic and linked data source to tackle many challenges such as interoperability and recommendation. These challenges usually come to, or at least include, similarity measuring between distinct entities taking advantage of LOD.

In [135] the author proposed a Linked Data Semantic Distance (LDSD) which relies on direct and indirect relationships between two DBpedia resources in the context of music

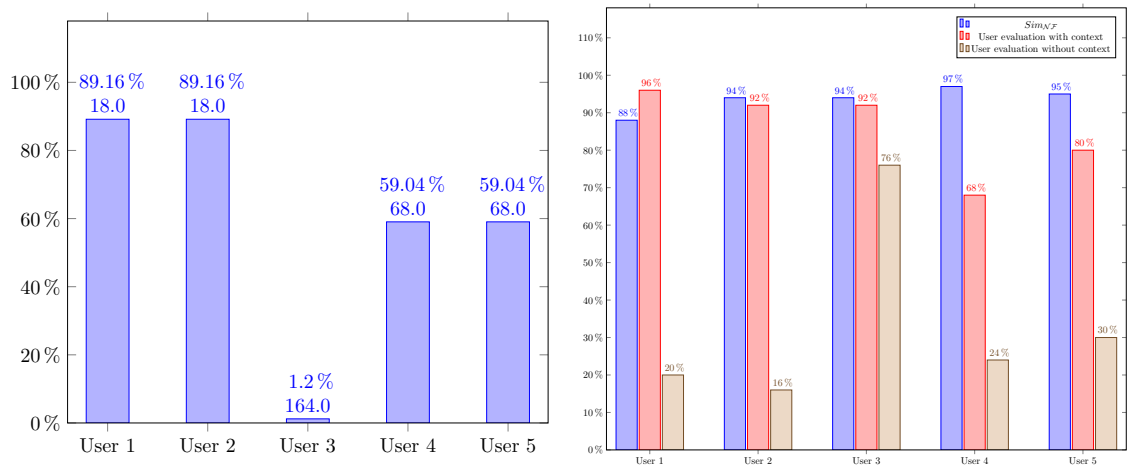


Figure 4.4: (Left) Evaluation of result precision. (Right) Comparing result quality without/without context-awareness of the approach with user evaluation

recommendation. LDSD applies on a cleaned dataset of DBpedia and considers relations between resources and concepts. This limits its accuracy compared to our approach as LDSD does not benefit from external interlinked datasets, it does not consider existing taxonomic structures and it ignores additional characterizing properties.

In [138] authors propose *REWOrD*, a measure based on a Predicate Frequency, Inverse Triple Frequency (*PF/ITF*) model inspired from *TF/IDF* used to compute the informativeness of the paths that connect compared resources. It is also limited in terms of considered LOD dimensions compared to LODS.

Later on [121] proposed Partitioned Information Content Semantic Similarity (PICSS), an information content (IC) based approach to compute the similarity between LOD resources. PICSS uses ingoing and outgoing edges as features and then compares resources based on Tversky model [157]. Although author report efficient performance evaluations, there is no formal presentation to allow generalizing the measure adoption, especially for the resource enriching process. In addition, PICSS has the same shortcomings as the two previously cited measures as it consider less features than LODS.

As for the use of semantic similarities for web services, in particular for context-aware service discovery, there are many proposed approaches following the emergence of ubiquitous computing research era. Going further than the simple use of locations, the proposed approaches model contextual profiles with additional properties related to user, to used devices, and to services [10] [62]. Compared to these works, our approach allows to integrate multiple types of context information requirements which improves the discovery effectiveness.

Context-aware services discovery approaches are usually defined as two steps corre-

sponding to functional and non-functional matchmaking as it is the case for our approach. This is the case for approaches in [155], [130], and [144]. Most of these approaches, however, rely on single ontologies for semantic similarity, and usually for the functional matchmaking only. The non-functional matchmaking is reduced to rule-based verification of contextual constraints [43, 66].

Our approach goes further through the use of LOD instead of single ontologies for similarity measuring. The proposed LODS measure is then used in both functional and non-functional discovery steps. Only mandatory contextual constraints are modeled through rules. The remaining profile properties are considered using LODS.

4.7 Conclusion

Starting from the challenge of providing cross-domain semantically-enabled service recommendation discussed at the end of the previous chapter, we consider Linked Open Data instead of single ontologies as semantic support underlying service frameworks. We propose LODS, a LOD-based similarity measure that exploits (i) the taxonomic structure of ontological concepts, (ii) the classification categories of LOD resources, and (iii) their characterizing properties. Prior to computing the similarity, the approach performs a data augmentation process that takes advantage of LOD interlink relationships with external datasets to enrich data involved in computing similarity measure to reduce the problem of lack of information within a single dataset. We build on top of LODS a context-aware service recommendation approach applied to the tourism domain. Services are modeled through functional and non-functional properties. LOD based annotation process allows to match these properties with corresponding LOD resources. Then adapted matchmaking based on LODS is performed to select the most relevant services for user needs in a given context. In addition, the LODS measure is packaged within a LOD-based similarity library that also implements the best-known state-of-the-art measures to allow their use and integration in any framework. The three contributions were evaluated and prove efficient results.

LODS similarity measure will also be considered in the contribution detailed in next chapters (Chapter 5 and Chapter 7).

The work described in this chapter was conducted during the scientific stay of Nasreddine Cheniki at the University of Tours between 2016 and 2019 as part of his PhD [45] and during the PhD of Fouad Komeiha to be defended in 2023. It is mainly published in the following conference papers: [46], [48], and [91].

Part III

Contextualized and user centric service interactions

Chapter 5

Configuration approach for personalized mashups

Contents

This chapter describes our contribution regarding the challenge of contextualized user-centric service interaction discussed in Section 2.1.2. It provides a detailed view of the approach described in Section 2.2.2.1, which models personalized service mashup building as a configuration problem based on Configuration Theory [151], then identifies the configuration solutions as relevant mashups. The approach relies also on a set of visual artifacts as Domain Specific Visual Language to allow end-user express preferences and constraints on the built Mashups. These artifacts are connected with a rule-based declarative language for consistent Mashup generation.

5.1 Introduction

With the large adoption of Service-Oriented Architecture (SOA) and Cloud Computing, Web services, generally in the form of Web APIs, have grown rapidly both in quantity and diversity. Users are surrounded with these services supporting their lifestyles: services that track their activity through smartphones, enable efficient use of home monitoring, provide the weather forecast or traffic reports, etc. They have become the first-class citizens on the Web and the core functionality of any Web application. However, they can no more be considered as independent/isolated entities, i.e. used individually, but as a set of pairs characterised by relationships that favour the interconnection. Their connected usage has the potential to create new value-added services. This type of service is known as a mashup. In general, a mashup is the process of merging a huge amount of heterogeneous data from different sources providing a fragmented view of the information and transforming these islands of data into an integrated view to fulfill complex needs [57, 42]. Moreover, nowadays end-users occupy an important position in the development of content, it makes sense to involve them in the mashup creation process through transforming end-users to prosumers (producer + consumer). In fact, there is an increasing need by end-users to realise their own ideas and to express their own creativity [161]. So far, systems have been conceived as pre-packaged sets of data, functionality, and visualisations that somebody else (the software developer) builds for us which no longer meets this new need. In addition, mobile devices are considered as the universal gateway between services and end-users.

Nevertheless, services are developed using SOAP-based or REST-based technology, which is difficult to understand for non-technical end-users aiming to compose services based on their specific preferences and/or needs. For example, if a user wants to plan for a trip, she/he will search for a Point Of Interest to visit (POI), restaurants and hotels. She/he would have to interact separately with the different services, handling different URLs, comparing retrieved results, and then aggregate data to respond to a particular travel information need. Even, the recommender system seems to provide a solution to the trip planning problem, they are limited to a list of recommended POI and lack of personalization. The user still needs to guess how much time is needed to visit each single attraction, and to devise a smart strategy to schedule them in order to build a personalized itinerary. A Travel Mashup is an automated solution for this scenario, where travel relevant data is retrieved and aggregated from services considering users' travel information needs, which may be weighted further by their personal preferences, as well as user's context then presented as directly usable information [38]. However, mashup tools require a programming background, so they are not suitable for end-users [12]. The inherent complexity of service composition must be adequately abstracted to give end-users easy-to-use development environments that hide all the underlying technical details. Regarding this, we think

that an ideal mashup platform should offer a lightweight development process promoting user-driven-innovation processes. It should support end-users by means of composition paradigms that abstract from technical details and fit with the particularities of the usage domain dealing with domain-specific paradigms to increase end-user innovation potential.

In this context, this chapter proposes a configuration-based approach for personalised mashup that provides end-users with a transparent service composition. A Domain Specific Visual Language (DSVL) that allows end-users to easily build mashup-based compositions is proposed. Then, the composition problem is modeled as a configuration problem to respond to the mashup query. In order to provide users with a personalised solutions, alternatives are generated by reconfiguration process. In addition, we propose a LOD-based reconfiguration process to improve the recommendation accuracy.

The remainder of this chapter is organised as follows: Section 5.2 introduces an overview of configuration-based approach. Section 5.3 exposes the DSVL for mashup query definition. Section 5.5 presents the configuration problem. Section 5.6 details the reconfiguration process. Section 5.7 provides insights on the implementation and the conducted experiments. Section 5.8 presents the related work. Finally, Section 5.9 concludes the paper.

5.2 Configuration of Personalized Travel Mashup Overview

An overview of our personalized travel mashup framework is shown on Figure 5.1. The framework provides end-users with interactive environment to compose services for their trip planning corresponding to their situational needs through a three-layer architecture:
The Service Layer includes the services developed by professionals developers. Particularly, services are implemented based on SOAP or REST technology.
The Application Layer provides end-users with a Visual Mashup Environment composed of three editors: (i) the mashup editor allows user to compose services, (ii) the rule editor allows to create rules describing situations to adapt the travel mashup, and (iii) a presentation UI to show the execution of the travel mashup in form of itinerary recommendation. This layer interacts with a Mashup Schema to access high-level service representations used by end-users to compose services and create the mashup query.
The Component Layer offers the software artefacts to connect the Application Layer and the Service Layer. It implements the mashup configuration approach relying on the following components. *The Mashup Schema* allows to map service implementations and service high-level representation which hides service technological issues. It describes two facets of services: an invocation facet, used to invoke services; and a semantic facet, used by Visual Mashup Environment. Developers deploy the service to the mashup schema by defining invocation and semantic representation in order to make it available to the mashup

environment. End-users communicate only with the high-level description offered by the semantic facet. The mashup query is analysed by the *Invocation Module*, which connects to the Mashup Schema to obtain the invocation facet of each service of the composition. Rule descriptions representing context-aware situation and user constraints are used by this module for mashup adaptation. Then, this concrete mashup query is executed by the *Configuration Module* to generate suggestions to end-users as itinerary recommendation displayed in the Presentation UI.

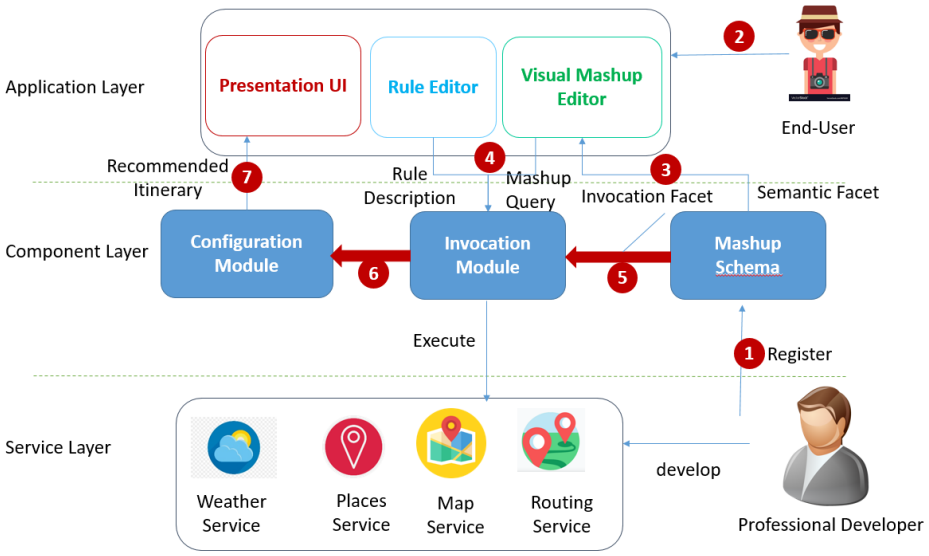


Figure 5.1: Travel Mashup Configuration Architecture

Numbers on Figure 5.1 correspond to possible interaction scenario steps detailed herein.

1. Service providers deploy their services (SOAP/REST) in the Mashup Schema and define the invocation and semantic service facets.

2. End-users interact with the visual environment to create the mashup query that includes the desired services.

3. The Mashup Visual Environment provides end-users with high-level service representation provided by the Mashup Schema through semantic service facet.

4. End-users complete the mashup by creating constraints and context-aware rules using the rule editor and then send it to the Invocation Module.

5. The Invocation Module requests the Mashup Schema to get service invocation data included in the mashup. Considering end-user rules and abstract mashup query, it generates a concrete query and sends it to the configuration module.

6. The configuration module executes the mashup query and generates itinerary suggestions.

7. The Presentation UI displays the recommended itineraries on a map and as a timeline.

5.3 DSVL for Mashup Query Definition

In this section, we present the key concepts of the DSVL which offer to end-users the possibility to compose services using high-level description. More details are given in [31].

An Activity A is a high level representation of a service which is developed by professionals and is registered in the Mashup Schema. $A =< name, description, graphicalicon, P >$ where name, description and graphical icon hide the technological issues that are required to invoke the service for end-users, and P is the set of activity parameters. Each parameter $p \in P$ has a *name*, a *value* and a *source* defining how the value should be provided.

We choose the term activity instead of service because it is closer to end-users' mental model. Thus, end-users have just to indicate the desired activities to perform in a mashup, such as get route from user position to an address, checking weather forecast, and so on. Furthermore, we use the term parameters to represent input and output without distinction to facilitate the mapping task for end-users.

Activities are connected through *Operators* such as *AND*, *OR*, *NEXT*. They can also be filtered through *filter* operators. Different metaphors were evaluated by end-users in order to determine intuitive and most adequate operators for components connection [56]. The puzzle and workflow metaphors were the two most ranked ones. We therefore considered the puzzle metaphors to create the graphical representation of a mashup, as shown in Figure 5.2.

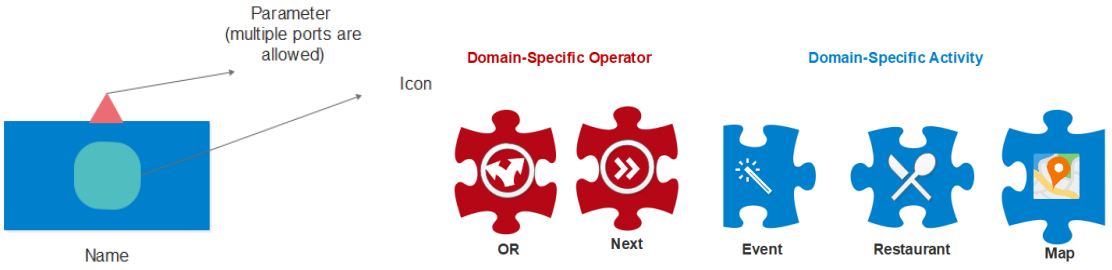


Figure 5.2: (Left) Abstract Activity model. (Right) Example of domain-specific components for travel mashup

The contextual situation is described through a set of user-specified rules using graphical notations to adapt mashups according to user's situational needs. We structure the contextual rule in two parts: Conditions representing the situation for condition activation,

and Actions to be executed. Typically, end-users interpret the "OR" operator as an "exclusive OR". Therefore, we propose a high-level rule description that hides this complexity based on *ALL CONDITIONS* connector and *AT LEAST ONE CONDITION* connector that refers respectively to AND and OR logic operator. Figure 5.3 shows the rule editor interface.

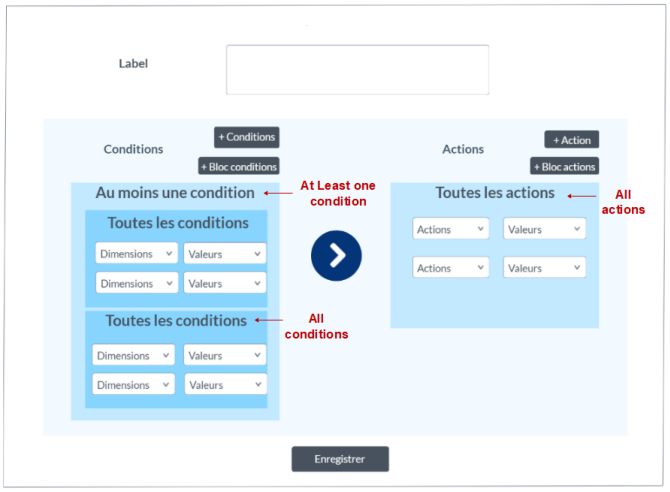


Figure 5.3: Rule editor interface

5.4 Mashup Schema

The mashup Schema acts as a gateway between end-users and service implementations, managing service data types from two facets: *Semantic Facet* and *Invocation Facet*.

Semantic Facet describes the goal of each service in order to be handled by end-users. Once developers register the service into the mashup schema by defining its invocation and semantic facets, it becomes available for end-users. The high-level service representation provided by the semantic facet and corresponding to the activity is used by end-users for mashup creation. It is represented with blue headers in Figure 5.4.

Invocation Facet provides the technological details of the service such as protocol, url, port, parameters, etc. This data is hidden to end-users at composition time. It is used only for service invocation at runtime. It is represented with green headers in Figure 5.4.

The mashup schema is implemented as a Java Web module, and expose a REST API to interact with it through the HTTP protocol based on JSON data format. Figure 5.4 describes the mashup schema composition for each facet by means of UML Diagram.

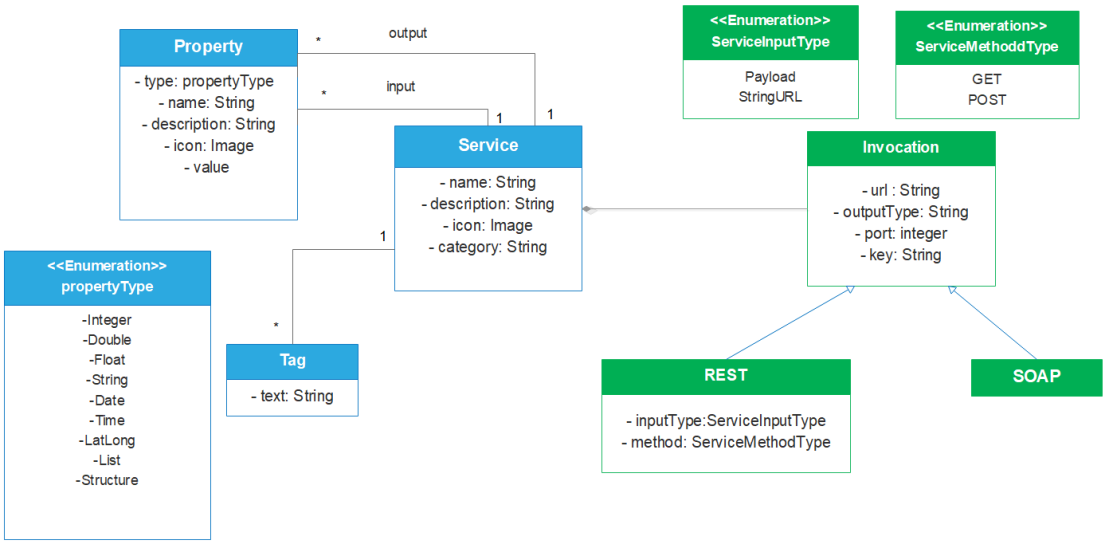


Figure 5.4: Faceted Mashup Schema Data

5.5 Configuration Problem

The service composition is seen as a configuration that consists to put together components following a logical description that respects user constraints to build a composite component responding to the user query. The mashup is designed using functional-based aggregation. Formally, we define the service composition as a *Configuration Problem* $CP = (m, R, CK)$ where m denotes the mashup query, R denotes the set of context-aware rules, and CK represents the Configuration Knowledge used to aggregate services that will be detailed in the following.

5.5.1 Configuration Knowledge

The configuration knowledge describes how to compose data (service instance) by mean of configuration operators and configuration rules that allow to generate a composition graph dependency. It relies on service functionalities, in the form of property-value pairs, to build composition called functional connections. Configuration operators specify how to aggregate services based on service functionalities ($f_i \in F$) to generate composite functionalities ($q_i \in Q$). Formally, let (f, x) and (g, y) be two functionalities and let $op(x, y)$ be a configuration operator. Then, the composite functionality $q = \phi((f, x), (g, y))$ is defined as follows:

$$\phi((f,x),(g,y)) = \begin{cases} (f, op_f(x,y)) & \text{if } f = g \\ (f,x),(g,y) & \text{otherwise.} \end{cases}$$

The composite functionality is a set that contains either a single property if the functionalities are equal, or the two properties if they are not equal. Thus, if two services, which have some properties in common, are included in a set containing the configuration, then it is necessary to compute the values of these properties in some way. This computation is done by the configuration operator. If this operator is not defined for the given value constellation, then these two services cannot occur at the same time in the set of configuration.

5.5.2 Configuration solution

A configuration solution $CS \in CSS(ConfigurationSolutionSet)$ to a configuration problem CP is defined as a pair $CS = (I, Q)$ where:

- I: refers to the set of items in the form (n, s) expressing the fact that service s appears n times in the mashup to be configured.
- Q: is the set of mashup features (called quality) in the form of a pair (f, x) that defines the value of the feature f.

Given a set S of services, a configuration solution CS is built recursively as follows:

1. $CS = (\emptyset, \emptyset)$ is a configuration solution.
2. If $CS = (I, Q)$ is a configuration solution and $s \in S$, then $CS' = (I', Q')$ is a configuration solution if the following conditions hold:

   (i) For each $(f, x) \in Q$ and for each $(g, y) \in Q$, the composite functionality $\phi((f, x), (g, y))$ is defined or $Q = \emptyset$.

   (ii)
   $$I' = \begin{cases} I \backslash (k, c) \cup (k+1, c) & \text{if } \exists (k, o) \in I \\ I \cup (1, c) & \text{otherwise.} \end{cases} \tag{5.1}$$

   (iii)
   $$Q' = \begin{cases} Q \cup (\phi((f, x), (g, y)) | (f, x) \in p_c, (g, y) \in Q) & \text{if } Q \neq \emptyset \\ p_c & \text{otherwise.} \end{cases} \tag{5.2}$$

Condition (i) guarantees that service $s \in S$ will be added to the configuration solution CS only if all the service properties can be combined with all CS qualities. Condition (ii) specifies how a new service can be added to the item set I. Condition (iii) specifies how a new set of qualities will be constructed when a new service is added to the configuration. Although qualities and functionalities are syntactically equal, we distinguish between them

since a functionality is the characteristic of a service, while a quality is the result of the combination of several services having the functionality $f \in F$. A configuration solution has the following characteristics: *validity* and *coherence*. *Validity* refers to user constraint satisfaction. Given a set V of non-negative values, an operator $op : 2^V \rightarrow \mathbb{R}_+$ and a budget threshold B, then $\forall CS_i \in CSS; i, j \in \mathbb{N},\ op(CS_i.I_j, CS_i.I_{j+1}) \leq B$. Budget can simply be the number of resources forming the configuration item solution. *Coherence* denotes context-aware solution.

For example, to deal with a travel mashup that includes *a touristic activity, a restaurant activity, and an event activity* for 1 day visit and 100€ as total budget, we define the following operator set overs services $s_i,\ s_j \in S$: $OP = \{op_{duration}, op_{price}\}$

$$op_{price}(s_i, s_j) = \begin{cases} s_i.price + s_j.price & \text{if } s_i.price + s_j.price < B \\ \perp & \text{otherwise.} \end{cases} \tag{5.3}$$

$$op_{duration}(s_i, s_j) = \begin{cases} s_i.d + s_j.d & \text{if } \ s_i.d + s_j.d + \delta t < D \\ \perp & \text{otherwise.} \end{cases} \tag{5.4}$$

op_{price} calculates the total cost of the visit by the progressive accumulation of service price property limited by the global budget B. Similarly $op_{duration}$ sums the duration of visit of the PoI and considers δt the transition duration to move from one PoI to the next. It keeps only solutions with feasable PoIs within the visit duration D, 1 day in this example. Using these operators and configuration rules, a configuration solution $CS_i \in CSS$ can be:

$$CS_i = \{(I, Q)\}\ where$$

$$I = \{(1, POI)\ ; (1, Restaurant)\ ; (1, Event)\}$$

$$Q = \{(cost, 80)\ ; (duration, 1)\}$$

To generate the configuration solutions, we propose an incremental model that consists in creating compositions by pair patterns, two service instance are composed at the time, applying the configuration operators. An initial configuration solution CS_1 is generated and is extended to incorporate additional instances of candidates forming a new solution CS_2. We call CS_2 *partial configuration*, as it does not yet satisfy the mashup query and must be completed by adding more instances. We repeat this step until reaching a final solution CS_n. The proposed algorithm is detailed in [31].

5.6 Reconfiguration Process

The reconfiguration process provides users with more alternatives to personalise the travel mashup using LODS similarity measure presented in the previous chapter. Figure 5.5 illustrates an overview of the reconfiguration framework.

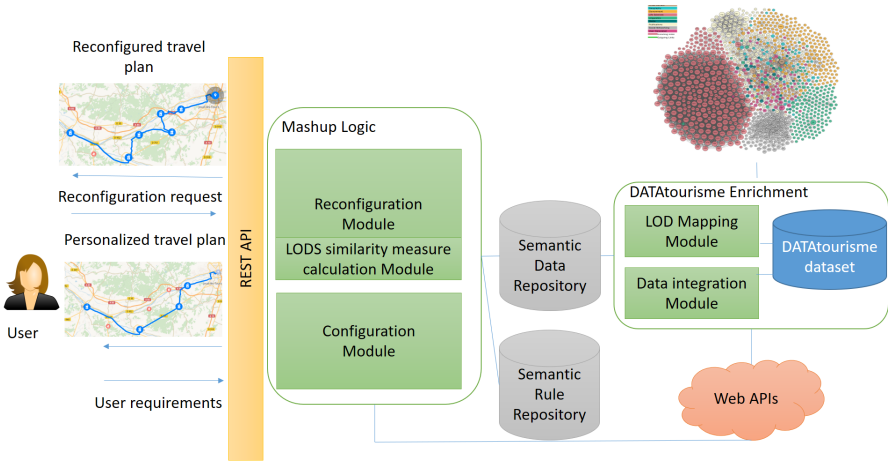


Figure 5.5: Reconfiguration Framework Overview

The DATAtourisme Enrichment module implements a lightweight integration process allowing to enrich the DATAtourisme[1] dataset that aims to centralize, within a national French shared platform, travel information produced by different Tourism Committees. Such information is published as Linked Open Data to facilitate the creation of innovative travel applications. We propose to enrich it by collecting data from travel APIs such as Foursquare, Google Places, Sygic Travel, etc. and map the different pieces of data using the DATAtourisme ontology model. This ensures a unified data representation model and helps then to transform non-RDF data into semantic ones in order to complete the missing information in DATAtourisme dataset. Moreover, to model information that is not covered by this ontology, we can use Schema.org, tourism-domain ontologies or parts from Mio! ontology network [165]. Furthermore, mapping the resulted unified semantic data to existing LOD datasets such as DBpedia[2] allows, in some cases, completing missing information that is not provided by API data sources. The mapping could then be exploited by LOD-based similarity measures such as LODS to propose similar touristic data. The final rich and semantically generated data is stored in a local RDF store.

Generated mashup can be reconfigured by providing users the possibility to personalize their initial plan through adding or removing travel activities as well as adapting properties of a travel activity such as time activity for example. The reconfiguration process exploits

[1] `www.datatourisme.fr/ontologie`
[2] `http://dbpedia.org/`

LODS, a LOD-based similarity measures, to produce similar travel activities as alternatives to the initial ones.

We illustrate in the Figure 5.6 a simplified example of data integration process. Tourism service information retrieved from two APIs (Google Places and Sygic Travel) are transformed (from XML and JSON respectively) into a semantic representation (RDF) before being integrated with DATAtourisme. New properties (rating and visit duration) have then been added to enrich the same service in DATAtourisme dataset.

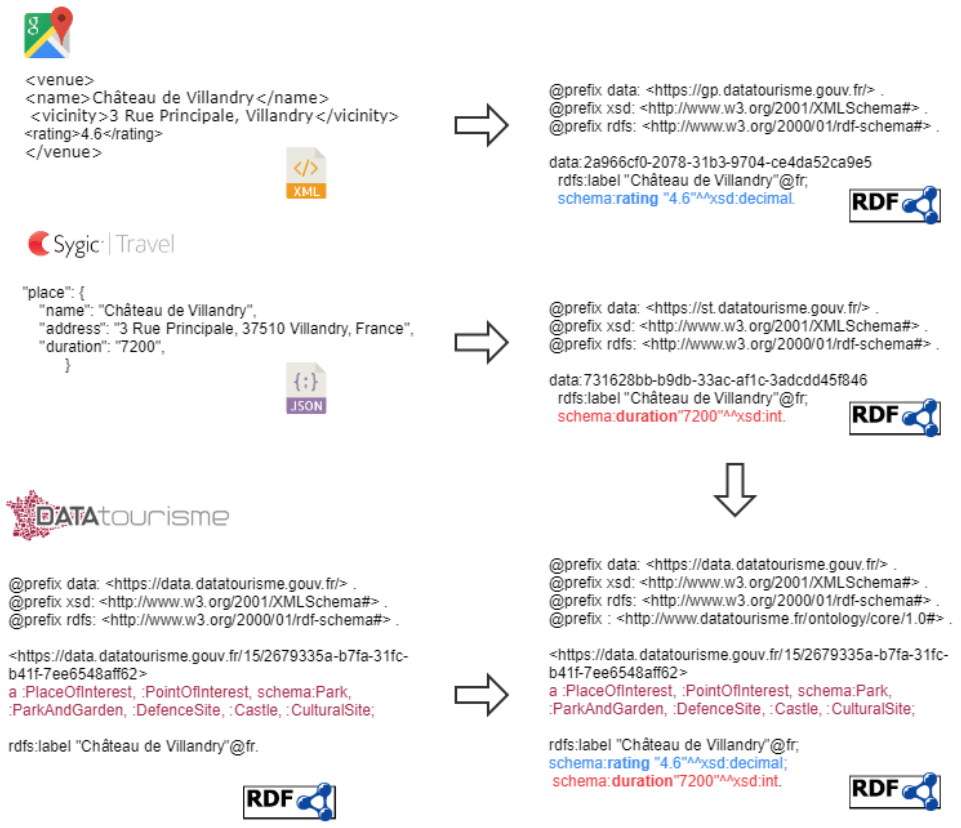


Figure 5.6: Example of tourism data integration

5.7 Implementation and Evaluation

The proposed configuration approach for personalized travel mashup is implemented within a prototype called CART (Configured mAshup Recommender application for personalized Trip planning). CART architecture is structured into 3 layers following the architecture depicted in Figure 5.1. Technically, the service Layer is in charge of collecting travel data from Web APIs eg. Google Maps, Google Direction API, Google Places API,

Forsquare and DATAtourisme. Based on JAXB API, the XML-data are serialized into POJO classes. The Mashup Layer implements the configuration approach as a SpringBoot App. The application layer is implemented as a client app consuming services from the mashup app. It includes UIs allowing users to build a query mashup for the trip plan indicating their preferences and constraints. In addition, CART provides end-users with interactive environment to define mashup query and visualizes visit plan recommendations for the Loire Valley region in France. The considered dataset includes more than 300 monuments, more than 30 remarkable gardens, 6 cities of art and history that crisscross the Loire Valley. Figure 5.7 shows the UIs of CART illustrating recommended itinerary.

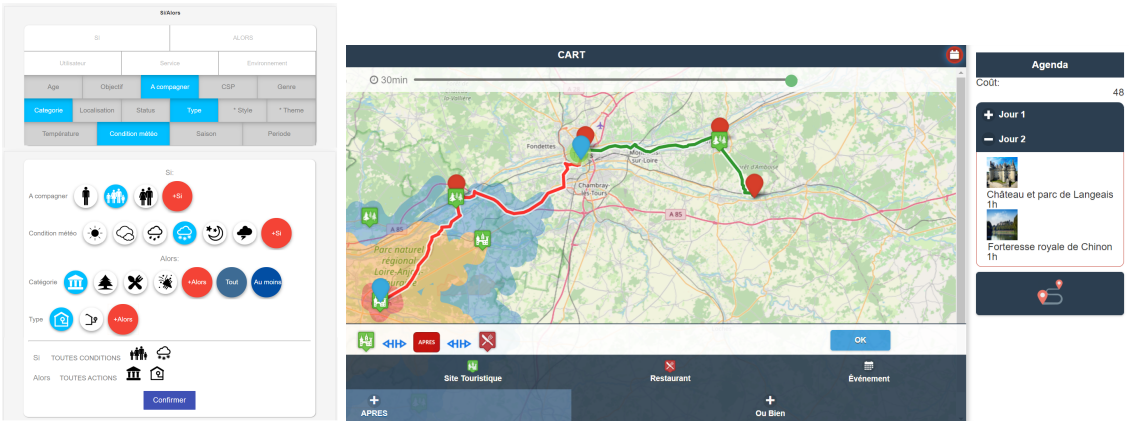


Figure 5.7: An example of visit itinerary generated by CART

We evaluated the proposed approach by 15 end-users (9 male/6 female) with different programming knowledge levels. Our main idea was to propose to participants to explore the mashup environment in order to build the travel mashup and define context rules via the rule editor. Globally, most of the participants felt comfortable using the platform and they mention a good satisfaction level. Detailed scores are given in [31].

The user were asket to answer a UEQ questionnaire which consists in describing two concepts with opposite meanings to be assessed on a Likert scale as shown in figure 5.8. The items constitute six different scales that are aimed at offering results categorized into attractiveness, perspicuity, efficiency, dependability, stimulation, and novelty [147]. Based on this classification, we have extracted a short version of the questionnaire, more appropriate to our platform. We used UEQ to test if the Rule Editor system has sufficient user experience in its current implementation. As can be seen in Figure 5.8, and according to the UEQ methodology, all scales show a positive evaluation, i.e. they are greater than 0.8. In order to interpret these results, we have based on the benchmark graph built by [147], illustrated by the Figure 5.8. We can note that all factors were given a value exceeding the average, which confirms a positive evaluation. The factor with the highest

score is dependability reflecting a strong sense of control over the interaction with the system.

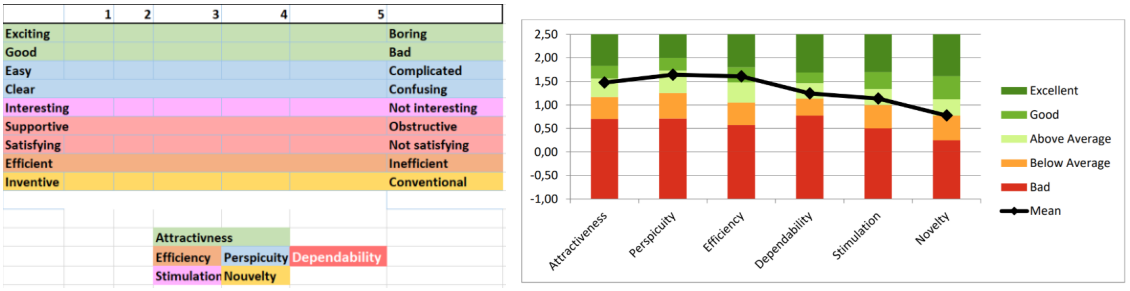


Figure 5.8: (a) UEQ questionnaire result (b) UEQ Benchmark reference

5.8 Related Work

Several mashup platforms: NaturalMash[9]; ResEval[57]; MashupEditor[76]; Puzzle[56]; EUCalip[160] [162], have been proposed to allow end-users to visually compose data and services from different sources, so that they can satisfy their information needs.

For instance, Puzzle[56] is a framework that allows end-users to create mobile applications. It allows the composition of device, smart objects, and Web service functionality using jigsaw metaphor that provides users to put together services without handling technical issues. EUCalipTool [162] offers an intuitive mobile environment that allows end-users to create service compositions based on a domain-specific language. Although theses approaches provide a domain-specific solution, they lack context-awareness.

Various applications for customizing the behaviour of existing applications in mobile devices or Web services have been introduced, such as IFTTT [133]. The idea is to describe the system behaviour based on if-then statements that specify how the system should behave regarding specific situations. They connect apps, devices and services to trigger one or more automation using abstract descriptions of services that mask technology details. EFESTO [12] provides end-users with a smart interactive experience to synchronize the behaviour of smart objects. It implements a visual paradigm for rule creation to support users in smart visit definition for cultural heritage scenario. Block Composer [113], a system for the end-user definition of trigger-action rules, uses the puzzle metaphor to support users in creating such rules. Creating rules is based on visual blocks, implementing a free composition style where users are supported by color and shape coherence of blocks, and integrated with recommendation techniques in order to provide intelligent support during rule creation.

The analysis of the related work illustrates that all the approaches provide high-level

service representation to hide technology issues and support end-users in service composition. However, composing actions and conditions in rule definition for personalised mashup still difficult for end-user as using logic operators is not trivial for them. Therefore, abstract descriptions for rule definition is needed. No approach combines domain-specific, context-aware and propose high-level representation of services and rules to effectively support end-users in personalised service mashup. Our approach fills this gap.

5.9 Conclusion

In this chapter, we describe the configuration-based approach for personalised mashup which is a domain-specific and context-aware mashup approach helping end-users in Tourism domain. The proposed approach is based on three key pillars: (i) Interactive mashup environment that allows end-users to create compositions by means of a DSVL offering high-level service representation. A Mashup Schema plays the role of a gateway between end-users and service implementation. This aspect allows us to achieve the goal of keeping end-users unaware of technological issue related with services. In addition, descriptions of available services are totally decoupled from the end-user tool, facilitating its evolution and maintenance. (ii) Configuration Module that models composition problem using the configuration theory. It aggregates travel related data to generate configuration that matches user constraints and provides context-aware solutions in incremental way. (iii) Reconfiguration process which calculates alternative solutions based on LODS similarity measure.

The work described in this chapter was conducted during the PhD of Marwa Boulakbech defended on July 2020 [25]. It is mainly published in the following journal and conference papers: [33], [34], [27], [28], [26], [29], [47], and [31].

Chapter 6

User-centric device recommendation

Contents

This chapter describes our contribution regarding the challenge of providing fluent user-side service running management discussed in Section 2.1.2.2. It provides a detailed view of the approach described in Section 2.2.2.2, which consists in a user-centric architecture following Liquid Software principals to allow for smooth service migration from one device to another from the user-side. The proposed approach also defines semantic and rule-based migration context to ensure service interaction continuity while switching to new devices.

6.1 Introduction

Nowadays, multiple-device ownership is increasing more and more driven by the high availability of smart objects, their increasing performances and the progress of data management and transfer protocols. It is indeed expected that by 2025, more than 75 billion

smart devices will be in use, according to Statistica[1] and Cisco[2]. It is now common to use simultaneously or in sequence at least 2 smart devices by the same user to perform daily tasks. Especially with the paradigm of Everything-as-a-Service (XaaS), people can access different services anywhere, anytime and from almost any device. In this context, multiple-device ownership raises new challenges when users run services over their different devices. One of the very common challenges is to switch from one device to another while keeping interaction with the same service. Services are hosted and managed on Web servers or Cloud which usually allow for multiple device access. They however require the user to login each time a new device is used. Moreover, in most cases, users have to restart interaction sessions from scratch. Two typical scenarios can be considered to illustrate the need for alternative user-centric solution: a light-coupling scenario with email service and tight-coupling scenario with YouTube service. Considering server-side synchronization, starting email writing on a device and then moving to a second device requires the login on both devices and saving the written part of the email as a draft from the first device to avoid data loss and to continue from the second device. Similarly, for video streaming on YouTube, user should login from both devices and resume video from saved history session on YouTube.

The challenge of fluently moving services through devices in multiplatform environments was addressed through the Liquid Software paradigm [80]. Achieving Liquid Software principals requires to build an entire infrastructure for dynamically moving functionality throughout a server-side or client-side network. Since then, almost all proposed Liquid Software approaches rely on server-side synchronization [52] and still face data and time loss issues. Nevertheless, the shift from SOAP to REST services is a opportunity to achieve user-side liquid software frameworks. REST and RESTfull require services to follow some constraints as described in [136], such as stateless interaction, hypermedia status, and addressability of resources. Following these specific constraints can help achieving interaction session sharing between devices regardless server synchronization. The study in [52] shows, however, that initiatives in this direction are still missing.

Successful migration of service interaction requires to ensure that target device is suitable for hosting the interaction. Also, it has to capture the users preferences, timeline and other features related to security, reliability, compatibility of devices, constraints of the services, etc. This issue has also been widely covered from the cloud/server side, as part of cloud resource management [109] or in the context of IoT device management [92]. From user-side however, efficient device recommendation modules are still to be defined. Such issue is more challenging as far as we consider the limited device computation performances and the lack of access to large semantic resources, such as Linked Open Data, for similarity

[1]https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
[2]https://www.cisco.com/c/en/us/solutions/internet-of-things/future-of-iot.html

calculation.

In this chapter we present our approach to provide users with user-centric framework, CUBE, that allows to for smooth user-service interaction migration over multiple devices from user side. The approach relies on Liquid Software principals [80] and REST/RESTful constraints [67], [136]. We also define a semantic rule-based declarative module to ensure successful interaction continuity on the hist devices. The reminder of the chapter is organized as follows. Section 6.2 presents the CUBE principals and formalization. Section 6.3 presents the framework architecture and main components. Section 6.4 presents the semantic rule-based module to ensure user-service interaction continuity over devices. Section 6.5 provides insights on implemented prototypes and evaluation results. Section 6.6 reviews the main state-of-the-art approaches related to our contribution. Finally, Section 6.7 concludes the chapter.

6.2 CUBE: Approach overview and formal representation

As alternative to server-side synchronization that allows time and data saving, we propose the CUBE, a user-centric framework that follows Liquid Software principals and defines a layered architecture built upon REST and RESTful constraints to ensure smooth user-service interaction migration in multiple device contexts from the user-side. With CUBE, both tight-coupling and light-coupling scenarios introduced previously will be handled locally preserving data and session states without the need for multiple login. The user logs in from the first device then the session can be moved to a second device through the use of RESTful properties allowing to manage data status. The multiple-device user context is virtualized for the server which continues interaction as with the first device.

In its abstract representation, the CUBE is seen as a four component model geometrically represented as: a central point which represents the user, surrounded by a first cube, called INNER CUBE, representing the set of user devices, contained in turn in a second cube, called the OUTER CUBE, representing the set of services, and between the two cubes the POOL AREA hosting the user-service interactions through the devices. Figure 6.1 shows a geometric representation of CUBE abstraction.

Formally, we define the CUBE as $\mathbb{C} = \{u, \mathbb{I}_c, \mathbb{O}_c, \mathbb{P}_a\}$. u denotes the user. $\mathbb{I}_c$ denotes the INNER CUBE defined as $\mathbb{I}_c = \{d_p, \mathbb{D}_u \setminus \{d_p\}, \mathcal{R}_d\}$ where d_p is the device being used by u, $\mathbb{D}_u$ is the set devices owned by u, and $\mathcal{R}_d$ denotes the research mechanism to discover new devices in the user context. $\mathbb{O}_c$ denotes the OUTER CUBE defined as $\mathbb{O}_c = \{s \in \mathbb{S}$ such that $\exists d \in \mathbb{D}_u$ and $d \leftrightarrow s\}$ where $\mathbb{S}$ is the set of services and $d \leftrightarrow s$ means that there is an interaction between u and service s through device d. And finally, $\mathbb{P}_a$ denotes the POOL AREA defined as $\mathbb{P}_a = \{d \leftrightarrow s$ such that $d \in \mathbb{I}_c$ and $s \in \mathbb{O}_c\}$.

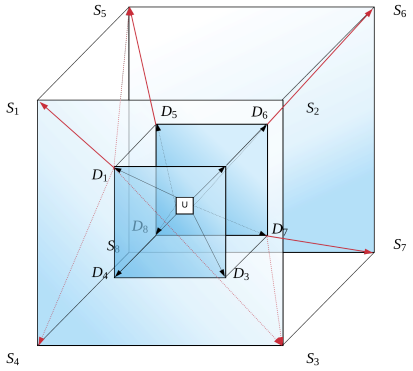


Figure 6.1: CUBE abstract representation

Defined this way, the CUBE is an adaptation of User Centric Network (UCN) in Liquid Software formalization, detailed [80]. The CUBE restricts the UCN definition to a single user instead of multiple users in UCN. In our first proposal of CUBE framework, the device discovery $\mathcal{R}_d$ in the user context relies on graph representation and graph search algorithms as detailed in [49]. Later on we enhance this module with semantic rules discussed in Section 6.4.

6.3 CUBE: Architecture and technical considerations

6.3.1 CUBE architecture

The framework general architecture is depicted in Figure 6.2. The content management system *101* includes two application portions *103* and *105*, two application program interface (API) portions *102* and *106*, and a database portion *104*. *102* and *106* respectively correspond to the INNER CUBE, $\mathbb{I}_c$, and OUTER CUBE, $\mathbb{O}_c$. *103*, *104*, and *105* are part of the the POOL AREA $\mathbb{P}_a$.

The content management system *101* will interact on two fronts, user device front, *100*, and service front, *107*. First, at user device front *100*, the user specifies the first device in use, d_p. Its choice is made from the available devices owned by the same user, following $\mathcal{R}_d$ discovery results. d_p is added to *102*. Other user devices discovered through $\mathcal{R}_d$ are also added to *102*. Then, the user selects the service, s to interact with through service front *107*. The service discovery process in out of scope of this work. s is added to *106*. The CUBE will be able to manage interactions between d_p and s through receiving and sending different types of requests made by the *100* through the conversation layer *105*.

The INNER CUBE *102* interacts with the user's devices, allowing to switch between

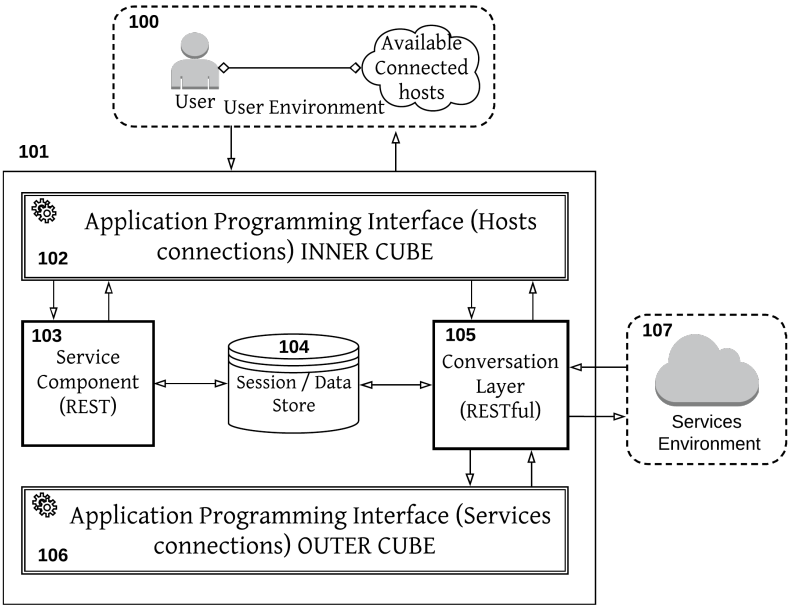


Figure 6.2: The CUBE architecture and its internal modules.

them during the service execution. While services and devices change, all the information and sessions retrieved are stored at *104* via module *103*. Thus, when any change of device is required, the user does not need to start from scratch in order to retrieve or fill in her/his information. The conversation layer, *105*, is also an API built under RESTful principles and is responsible for changing information across the multi-OS environment. Moreover, this part of the module provides a secure connection for the user devices through a token created in the INNER CUBE *102* by the join of User ID from devices and services which are provided by the OUTER CUBE *106*. This type of addressability through a resource identification allows the pool area to ensure a stateless interaction between services. Portion 107 relies on pool area to request external services.

6.3.2 CUBE layered representation and technical considerations

In order to achieve the expected functionalities of the CUBE framework, we follow a layered representation which holds a parallel with TCP/IP layers while proposing additional improvements on each layer to meet the interaction migration requirements. Figure 6.3 shows CUBE layers and their correspondence with TCP/IP. The layered CUBE architecture is made of five different layers: (i) Application, (ii) Request, (iii) Conversation, (iv) Data, and (v) Physical that we will briefly describe hereafter. More detailed descriptions and technical implementation details are provided [49].

Our model is based on an innovative approach with a strong attention to REST/REST-

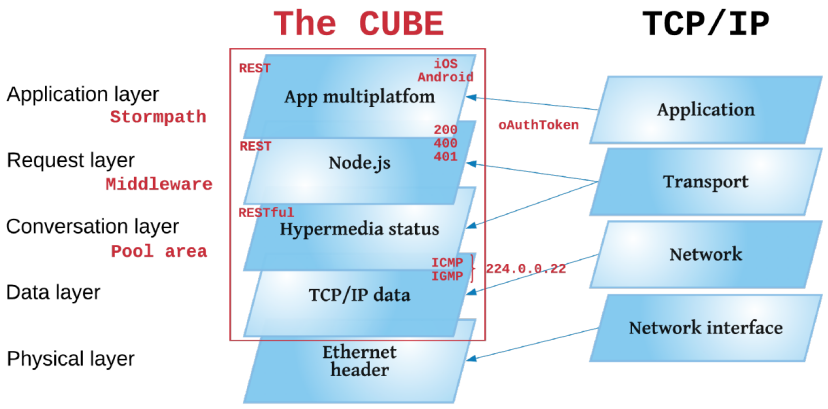


Figure 6.3: CUBE layers and main used technologies and protocols.

ful principals and constraints [136]. We therefore revisited the definition of the layers and propose separated layers for REST and RESTful approaches. The *Application* and *Request* layers work together to allow for the multiplatform, where a service can be started for example on an Android device and then moved to an iOS device. In order to achieve this kind of fluidity, we chose JavaScript (JS) as the language for the two firsts layers.

The Application Layer makes use of the Stormpath[3] for user authentication and security management and respects the principals of the REST model. This layer will be only one of the model layers to appear to the user to allow verifying all the synchronized multi-devices on the user-centric network.

The Request Layer acts as a Middleware, allowing for a connection between Application and Conversation Layers through an oAuth Token method. Basically, this layer works by sending the HTTP responses across the model. It is implemented in Node.js.

The Conversation Layer is, in our perspective, the gap required for the Liquid Software approach working independently of the platform [124, 70, 122], and in our model, it is the domain between the INNER and OUTER CUBES. The main challenge here is to change the data status, if required for any method, allowing for the content to be displayed correctly in the following device. Therefore, this layer is represented as RESTful.

The Data Layer can be modeled in two perspectives: a centralized model with a node-host or a sharing environment without central-node based respectively on IP Multicasting and Peer-to-Peer principals.

The Physical Layer allows for fluidity over all types of connection. It uses the Ethernet header structure and encapsulate data over IP datagrams for sending and receiving.

[3]https://github.com/stormpath/stormpath-sdk-php

6.4 Semantic rule-based device recommendation for service-migration in multiple device environment

A successful migration decision, whether it is done automatically or manually, requires to follow some criteria to know which target device is the most suitable for the service/application that will be migrated. Also, it has to capture the users preferences, timeline and other features related to security, reliability, compatibility of devices, constraints of the services, etc., in order to be able to suggest the right device. Although this issue has been widely covered from the cloud/server perspective as part of cloud resource management, efficient device recommendation is still a challenge from the user-side perspective. It is even more challenging as far as we consider the limited device computation performances and the lack of access to large semantic resources, such as Linked Open Data, for similarity calculation. To address this challenge, We enhance CUBE with a declarative module that relies on ontologies and semantic inference rules, discussed hereafter, to ensure device recommendation.

(i) OWL ontology core concepts: We conceptualize 4 main components: *Device*, *Owner*, *Service*, and *Characteristic*. Each of them contains specific sub-classes to handle service migration context diversity. We also define necessary properties to model the interactions and relationships between theses concepts. For example, the Owner class has two object properties with the device class: a device is *ownedBy* an owner, and the owner *usesDevice*. These two properties are different because a user can own several devices but only uses some of them at a time. Another property is: the user *interactsWith* the service. Furthermore, in order to define the relationship between the devices and their characteristics or the services with their requirement, we add two main properties. *hasCharacteristics* property: links between Device and Characteristic, while the *requireCharacteristics* property: links between Service and Characteristic. Figure 6.4 shows a screenshot of the concepts and properties on Protégé.

After designing the ontology classes and properties, we need to fill the instances with the users devices and context information. The optimal way to do this would be to get APIs do the job, as we should keep users intervention as minimal as possible to facilitate the process. For instance, Battery API can give us: the battery percentage of devices, whether it's charging or not, etc. Ontology-update-process starts either when the user manually triggers a migration request or when predefined callback methods are automatically triggered through specific events related to service or device life-cycle. Both methods can be used depending on the context and on the user preferences. However, APIs cannot get all the information needed; that is why we might need users to enter/complete them manually.

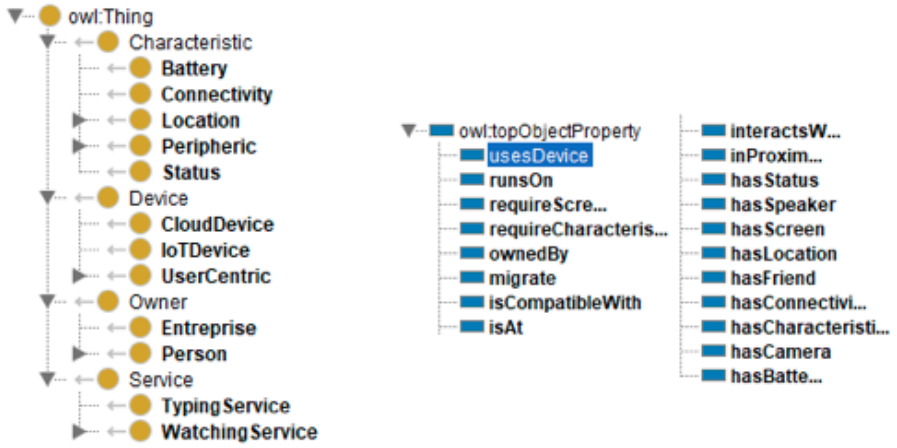


Figure 6.4: Core concepts and properties

(ii) Formalizing migration rules with SWRL: Based on the defined concepts and properties, we formalise service migration context and conditions through a set of inference SWRL rules. For example,
R1: $Device(?d) \wedge HasBattery(?d, ?b) \wedge swrlb : greaterThan(?b, 10) \rightarrow Alive(?d, true)$
defines the condition to consider a device as alive whereas,
R2: $Device(?d1) \wedge Device(?d2) \wedge HasLocation(?d1, ?l) \wedge HasLocation(?d1, ?l) \rightarrow InProximity(?d1, ?d2)$
expresses under which condition two devices are considered to be in proximity.

Migration rules are usually a combination of other rules. For example R4 which requires R1, R2, and R3 is an example of migration rule expressing conditions to achieve a successful migration.
R3: $Device(?d) \wedge HasScreen(?d, ?c) \wedge WatchingService(?s) \wedge RequiresScreen(?s, ?c) \rightarrow IsCompatiblewith(?d, ?s)$.
R4: $Device(?d1) \wedge Device(?d2) \wedge Service(?s) \wedge Alive(?d2, true) \wedge InProximity(?d1, ?d2) \wedge IsCompatibleWith(?d2, ?s) \rightarrow Migrate(?s, ?d2)$.

These inference rules can be very flexible and they give us efficient semantic meanings to achieve relevant device recommendation for migration. They provide typical migration situation. More specific ones related to user context and situation for migration can be defined by the user in the same way.

6.5 Implementation and evaluation

The CUBE prototype is implemented as a hybrid application based on JavaScript, Node, JAVA, HTML, PHP, and Cordova. Were also considered SQL and NoSQL models

to authentication and synchronization, which are deployed as a callback procedure for performance. Regarding the Internet connection, we manage a local sign up/log in, and allow synchronizing this authentication step when a connection is available. In addition, we get IMEI and MAC addresses of the device and associate them to the user login.

As a first feasibility tests we considered both light-coupling and tight-coupling scenarios for email and video session migration respectively. The email scenario was developed based on Gmail API [53] and the video scenario on YouTube [54]. Both scenario were successfully run while moving over multiple devices with different OS. A demo video is available for video session migration scenario (`https://youtu.be/-TYPRXtC7Lw`). In a more recent , the CUBE was extended to support migration of any stateless interaction and was tested for videoconference meeting migration [50].

The semantic rule-based device recommendation extension is implemented within an API outside the CUBE framework. The ontology is designed using Protégé and the SWRL rule-based reasoning is performed through Apache Jena. The API manages and stores the facts and rules (both defined in Apache Jena) to finally compute the recommendation results. Facts are very dynamic and change depending on the situation and the devices. They are stored in a ".n3" file. Some characteristics are automatically added by the app from device APIs (Battery, Connectivity, etc.) while others are filled in manually by the user. A set of basic common rules is provided by the app for each device. It can then be enriched by the user to handle specific migration scenarios.

Evaluation of CUBE implementations was performed with the focus on time and data saving as well as user satisfaction and quality of experience. Selected users were provided a set of devices and asked to reproduce predefined interaction scenarios. Results are detailed in [53], [54], and [50]. They show interesting achievements regarding time and data saving as well as high satisfactory user quality of experience.

The semantic rule-based device recommendation API was also run over migration scenarios to measure the accuracy of the approach towards recommending devices that allow successful service running after migration. The results, detailed in [132] show also high accuracy as well as high user satisfaction.

6.6 Related work

Service interaction synchronization is widely covered by companies such as Apple for cloud services running iOS and Google for Google Docs and other services running on Android. In addition, other approaches have also been proposed in literature for cross-device service synchronization such as [131] and [170] for services running on Android devices. As discussed in the Introduction, these solutions do not allow for fluent service

migration as each device has to connect aside to get synchronized service interaction. In the same category of server-side synchronization, [70] propose the most advanced approach that realizes Liquid Software principals, called LiquidJS. More than service synchronization, LiquidJS allows for service migration over devices but again it handles the migration relying on a server through an API or a browser. The approach also does not meet multi-platform migration. A detailed discussion including a comparative study of the the best known proposed approaches is given in [49] and [54]. From user-side, to the best of our knowledge, the CUBE is the only approach that allows service migration in both multiple device and multiple OS.

As for semantic device recommendation, approaches exist form server perspective under the topic of cloud resource management and allocation. Among them we can cite the mOSAIC ontology [125] for cloud resources annotation and management intended to ease multi-cloud oriented applications development and [109] which deals with job allocation in cloud systems using OWL ontology and SWRL rules. Both approaches use large ontologies defined to cover as much as possible the concepts related to cloud systems. In our work, we rather focus on a small ontology with few semantic rules that can be run by devices locally without need for high computing resources.

6.7 Conclusion

Despite all efforts, there is still a gap before achieving both a multi-device and services synchronization over multi-OS from user-side, described herein. In order to address this challenge, we have presented our CUBE Model under the principles of Liquid Software. Throughout a user-centric approach and combining principles of REST and RESTful models, our proposition, allows to move user-service interactions over multiple devices with heterogeneous environments. The CUBE realization applies the best practices of different technologies fulfilling their principals and constraints regarding the issues of security, stateless interactions and providing a uniform interface shared by all resources, across separated layers for the REST and RESTful applications. Evaluation over typical user-service interaction scenarios show satisfying results in terms of performances and quality of experience.

We also proposed a semantic rule based complementary module for recommending the most suitable device to migrate to. The module is also implemented and evaluated showing high recommendation accuracy.

The work described in this chapter was conducted during the PhD of Clay Palmeira da Silva defended on September 2019 [49] and during the Master 2 Internship of Dimeth Nouicer defended on December 2020. It is published in the following conference papers [52], [51], [53], [54], [132], and [50].

Part IV

Dynamic service interaction handling with Machine Learning and Blockchain

Chapter 7

Handling dynamic service evolution with LOD and ML

Contents

This chapter details our first contribution, summarized in Section 2.2.3.1, towards handling dynamic evolution of service interactions discussed in Section 2.1.3.1. The proposed approach relies on a graph representation modeling a multi-relation network of services and Linked Open Data annotation and similarity measure introduced in Chapter 4 to propose a three-phase process for service continuous deployment and changes handling. Bootstraping phase for service annotation and matchmaking, Update phase for service network updating from interaction feedbacks, and Learning phase for LOD interlinks updating.

7.1 Introduction

Recent breakthroughs in service oriented computing have encouraged companies and organizations to export their functionalities as Web services to gain in visibility and market opportunities. Web services do not operate in silos; they usually interact through several

relationships such as *Composition* and *Substitution*. In dynamic environments, new Web services are continuously deployed. Moreover, existing Web services are updated regularly. Such updates are most of time associated with modifications to service description expressed in natural language. These updates affect operation results, and hence may lead to breach of contract between service providers and clients [69]. Therefore, it is mandatory to handle Web service changes that occur at run-time and accurately communicate these changes to the concerned peers in the network. Handling Web service continuous deployment and updates offers several advantages: First, it avoids brutal contract violation between providers and customers. Second, it speeds-up the exposure of new functionalities in the network, helps them gain more visibility, and hence improves the overall availability of new services. Third, it helps setting new partnerships among Web services.

However, managing the evolution of Web services is challenging for the following reasons. When deployed or updated, Web service capabilities are described by means of keywords in natural language. What is needed is a semantic approach to match service capabilities by matching their describing keywords. Moreover, Web service provided description may not match with service real capabilities. What is also needed is a technique to update service relationships according to service interaction experience and user feedbacks. This rises the following research questions respectively: (1) How to bootstrap relationships for newly deployed/updated services and figure out the most suited relationship type to link two services ? (2) How to update service relationships and ascertain whether two Web services are interoperable in practice ? (3) And finally, How to take advantage of service interaction experience to help learning new links in LOD and in turn improve relationships recommendation accuracy with the newly learned links ?

In this chapter, we describe our approach based on Linked Open Data (LOD) and Machine Learning (ML) for handling Web services deployment and evolution. Service interactions are modeled as a social network of service through a multi-relation graph where nodes correspond to services and edges to interaction relationships. The approach follows a three-step process: LOD based service annotation and matchmaking, ML-based service network update using interaction feedbacks, and LOD links updates using service network relationships. These phases rely on efficient algorithms. The overall approach is implemented and evaluated on service interactions from ProgramableWeb between 2007 and 2017.

The remainder of this chapter is organized as follows. Section 7.2 provides an overview of the proposed approach and a general architecture highlighting the main components. Section 7.3 presents bootstrapping relationships for newly deployed and updated services. Section 7.4 introduces our technique to update relationships according to service interaction with each other and with user. Section 7.5 describes our technique to learn relationships in

LOD according to service interactions and feedbacks. Section 7.6 discusses the experimental study. Section 4.6 reviews related works. Finally Section 7.8 concludes the chapter.

7.2 Approach architecture and overview

The overall architecture of the proposed approach is depicted in Figure 7.1. The approach runs a three-phase process to handle Web service deployment and evolution. The process is triggered when a service provider deploys the interface of his service, and provides a natural language description of service functionality and application domain. First, *Bootstrapping Phase* invokes the LOD-based Annotation module to associate a profile to the service: a set of keywords describing service inputs, outputs, functionality, and application domain. The keywords are annotated with LOD resources. After that, *Bootstrapping Phase* invokes LOD-based matchmaking module to match annotated service profiles and derive composition, substitution, and subscription relationships. This results in new weighted edges for newly deployed service and allows bootstrapping service relationships. This same process will also be run when a Web service undergoes an update. Annotation module will annotate service new features in interface and description with LOD resources. Matchmaking module will match service new features with existing network features. Second, *Updating Phase* updates service relationships according to service interactions with each other and with users. As services record successful interactions, they receive high rating scores, this increases the weight of their relationships. In the same way, weights are decreased for negative feedbacks. Third, *Learning phase* improves bootstrapping relationships accuracy by learning new relationships in LOD. When the *Updating phase* returns a higher weight than the *Boostraping phase* weight for the same relationship, this would mean new relationships should be added to LOD and viceversa.

These phases will be formalized and detailed in the following sections. The formalization relies on multi-relation graph representation to model service network introduced in [97]. Formally, a multi-relation network of Web services is a directed graph $G = (V, R)$ where V is a set of services, and $R = \bigcup_{k=1}^{k=3} R_k$ is a set of edges, where $R_k = \{(S_u, S_v) \in V \times V\}$ is the set of edges of the k-th relationship. Major services interaction relationships are: substitution (R_1), composition (R_2), and subscription (R_3). An edge $(S_i, S_j) \in R_1$ indicates that there is a relationship of type substitution between Web services S_i and S_j, precisely it indicates that S_j is a substitute of S_i and clients can substitute the invocation of S_i by S_j if S_i is unavailable. Similarly, a composition edge $(S_i, S_j) \in R_2$ indicates that S_j can be composed with S_i and clients can invoke S_j and S_i in the same composite application. A subscription edge $(S_i, S_j) \in R_3$ indicates that S_i can subscribe to the news feed of a collaborator/competitor S_j to stay aware about its latest updates.

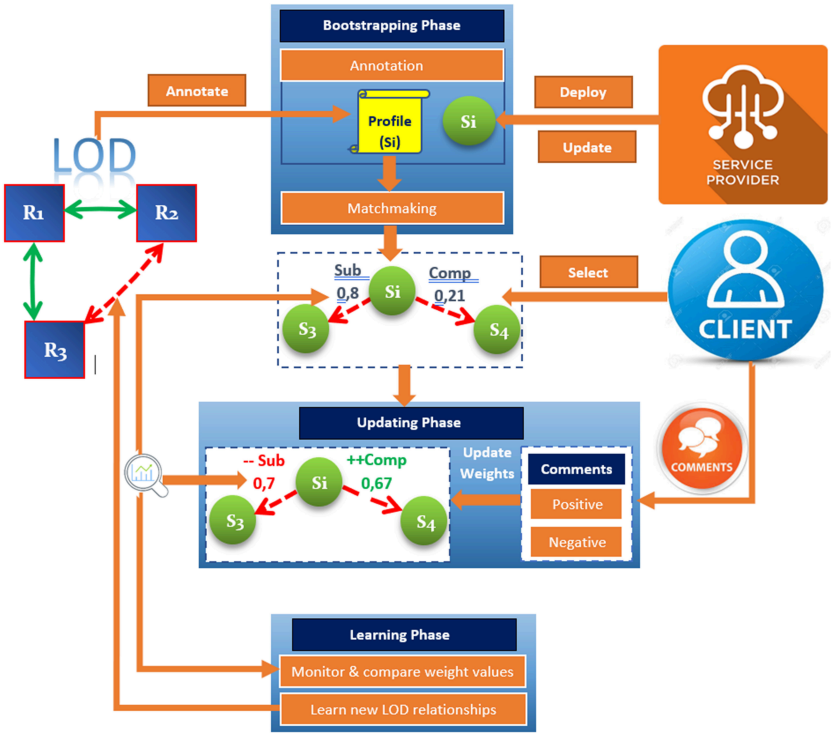


Figure 7.1: Architecture of LOD Management of Service Deployment and Evolution

7.3 Bootstrapping Phase

LOD-based service annotation and matchmaking. This step relies on the contribution detailed in Chapter 4. It applies LOD-based annotation process on service features and uses LODS measure within the service matchmaking. Here, service profile is formally modeled as a pair (F, D) where F is a description in natural language of the service functionality and D is a list of keywords that describe the service application domain. Text in F is pre-processed to keep meaningful keywords describing service functionality. The LOD-based annotation maps these keywords to LOD resources. Then we apply service matchmaking. to infer potential composition and substitution relationships. Semantic matching is invoked in two cases: (a) When a Web service joins the network. (b) When a Web service gets updated. It matches Web service capabilities (either the initial or the updated ones) to recommend the most appropriate relationship to link with two services. It also predicts the probability value to interact through the recommended relationship.

Given two service features $feat_u$, $feat_v$ described each with a set of keywords $K_u = \{k_{u1}, k_{u2} \dots k_{ui}\}$ and $K_v = \{k_{v1}, k_{v2} \dots k_{vj}\}$ respectively, where each keyword can be attached to a subset of annotation resources $A_R \in LOD$, such that:
$\forall k_i, k_j \in K_u \times K_v, \exists\, r_i, r_j \in A_u|\langle k_i, r_i\rangle \times A_v|\langle k_j, r_j\rangle$. The similarity between features $feat_u$

and $feat_v$, annotated by a set of LOD resources A_u and A_v respectively is computed in the same way as in Equation 4.17:

$$Sim_{LOD}(feat_u, feat_v) = \frac{\sum_{a \in A_u} \sum_{b \in A_v} LODS^{\ell,\ell'}(a,b)}{|A_u| \cdot |A_v|} \tag{7.1}$$

Bootstrapping Composition Relationship This phase assumes that two Web services S_u and S_v are composable and can be linked by a composition edge in the network if and only if: (i) S_u and S_v appear together in at least one mashup, and (ii) the similarity values between S_u and S_v functional descriptions and application domains are higher than the following thresholds α_F and α_D respectively.

Algorithm 1 evaluates how likely are two services S_u and S_v to be part of the same composition. The algorithm computes (1) the similarity between their functional descriptions. And (2) the similarity between their application domains. If computed similarities are above given thresholds, then the algorithm computes a composability likelihood $Comp$ between Web services S_u and S_v. If the composition relationship (S_u, S_v) exists, the algorithm updates the composition likelihood as an average of the computed composition probability and the relationship current weight. Finally the algorithm adds a new composition edge (S_u, S_v) to the network of services with the composition likelihood as weight.

Algorithm 1 Composition Matching Algorithm

Input: $G(V, R)$: Graph of services. $S_u, S_v \in V$.
α_F, α_D : similarity thresholds for functional descriptions and application domains
$Coef_F$, $Coef_D$: coefficient values for similarities between functional descriptions, and application domains respectively.

Output: $G(V, R)$: Updated graph of services.

Begin

1: $Sim_F \leftarrow Sim_{LOD}(S_u.F, S_v.F)$;
2: $Sim_D \leftarrow Sim_{LOD}(S_u.A, S_v.A)$;
3: **if** ($Sim_F >= \alpha_F$ and $Sim_D >= \alpha_D$) **then**
4: $Comp(S_u, S_v) \leftarrow (Coef_F * Sim_F + Coef_D * Sim_D)/(Coef_F + Coef_D)$;
5: **if** $((S_u, S_v) \in R_1)$ **then**
6: $Comp(S_u, S_v) \leftarrow AVG(Comp(S_u, S_v), Weight(S_u, S_v))$
7: **end if**
8: $R_1 \leftarrow R_1 \cup (S_u, S_v, "Weight"=Comp(S_u, S_v))$
9: **end if**=0

Bootstrapping Substitution Relationship This phase assumes that two Web services S_u and S_v are candidates for substitution and can be linked through a substitution edge in the network if and only if: (i) they do not appear together as parts of the same mashup, and (ii) the similarities between S_u and S_u functional descriptions and application domains are higher than given thresholds α_F and α_D respectively. Substitution Algorithm differs from Algorithm 1 in that it uses specific threshold and coefficient values. For instance, functional description similarity threshold should be higher for substitution than it is for composition.

The current phase allows bootstrapping relationships for, newly deployed services and those undergoing updates, based on LOD similarity measure. However, relying only on service profile data may lead to inconsistencies between service real capabilities and described ones in profiles. The next section proposes to continuously update service relationships according to user feedbacks.

7.4 Updating Phase

Although semantic matching of service profiles allows building relevant business relationships, it relies however entirely on Web service profile data. Such an approach considers Web services as isolated components and underlooks their interactive behaviour. In reality, Web services interact through various relationships such as composition, substitution, and subscription. *Bootstrapping Phase* may suggest that two services are composable while they struggle to interoperate in practice. Their record of successful interactions may be weak. Users will then give bad rating scores. The reason may be QoS limitations of some services that make them incapable to interoperate with network services. This may also be due to an incoherent service description with service real capabilities that led to biased semantic matching results. Gathering and analyzing Web service interactions and their user feedbacks allow assessing the real value/relevance of bootstrapped relationships. *The intuition is to update relationships weight values returned by semantic layer according to user rating scores.* A relationship (S_u, S_v) that keeps a high weight value after many service interactions means that S_u and S_v have collected a strong record of successful interactions. These interactions have been appreciated by users and consequently have been well rated. On the opposite, if a relationship will see its weight decreasing as services interact, this would mean that services struggle to interoperate with each other. The relation did not get high rating scores and so even after semantic matchmaking returned a high weight value. This section catches Web service interactions and their user ratings and updates relation weight values accordingly. This allows keeping the network updated and eases identifying interoperable services and computing service reputation scores.

We extend the measure proposed in [99] to leverage user comments for updating relationship weights as follows:

$$WE_{t+\delta t}(S_u, S_v) \leftarrow \ AVG\left[\frac{|positiveFeedbacks(S_u, S_v)_{t+\delta t}|}{|allFeedbacks(S_u, S_v)_{t+\delta t}|}, WE_t(S_u, S_v)\right] \tag{7.2}$$

$WE_t(S_u, S_v)$ and $WE_{t+\delta t}(S_u, S_v)$ are the weights of the relation (S_u, S_v) observed at t and $t + \delta t$ respectively.

$positiveFeedbacks(S_u, S_v)_{t+\delta t}$ and $allFeedbacks(S_u, S_v)_{t+\delta t}$ designate the number of *positive comments* and *all comments* respectively that have been commented on the relation (S_u, S_v) in the time interval $[t, t + \delta t]$. We apply *sentiment analysis* to distinguish between positive comments and negative comments observed in a time interval $[t, t + \delta t]$

The updated weight depends on the previous weight (i.e., bootstrapped weight) and on user feedbacks on the relationship. The higher will be positive user feedbacks on the relationship, the higher will get the updated weight. The weight will decrease on the contrary case.

7.5 Learning Phase

This phase consists in proposing new links in LOD according to Web service interaction experience. Keeping LOD up to date allows running accurate semantic matching of Web service capabilities, hence accurately identifying potential business relationships among Web services. Updating LOD is triggered when the gap between the updated weight value and the bootstrapped weight value gets above a given threshold for the same relationship. If the updated weight is greater than the bootstrapped one, this means that Web services are interoperable (i.e., have a strong record of interactions) in practice, while bootstrapping phase has not recommended them as such. This proves the lack of information in LOD. Thus, new relationships among resources describing these interoperable Web services should be added to LOD. On the opposite, if the updated weight is the lowest, this means that Web services struggle to interact. The cause can be either (i) Web service description is inadequate with service real capabilities, or (ii) Web service suffer from QoS limitations. For the first case, LOD relationships among resources describing such services should be destroyed. For the second case, LOD ontology structure will be preserved.

Algorithm 2 details how to learn new LOD relationships or to break existing ones. It derives (LearnLOD) and breaks (BreakLOD) links among resources describing services according to a "Normal probability distribution". Such probability distribution is the most adequate to model phenomenon that consider the sum of many independent variables. In

our case, LOD similarity measure of two service features is computed as the summation of atomic similarities of their LOD resources. By analogy, we learn or destroy new LOD relationships according to a "Normal probability distribution". As it is hard to overwrite existing LOD, we save new learned and broken relationships in local database. Such learning will serve further bootstrapping relationships that relies mainly on semantic matching of service features. Hence, future relationship bootstrapping will be more accurate. This allows a fast exploitation of new functionalities in the network and enhances their overall availability.

Algorithm 2 Learning New Relationships in LOD

Input: $G(V,R)_t$: Graph of services observed at t. $G(V,R)_{t+\delta t}$: Graph of services observed at $t+\delta t$. S_u, $S_v \in V$, *gap*: Learning gap.

Output: Updated LOD

Begin

$weightdiff \leftarrow weight(S_u,S_v)_{t+\delta t}$ - $weight(S_u,S_v)_t$;

1: **if** ($weightdiff$ - gap >0) **then**
2: $LearnLOD(S_u.F, S_v.F)$
3: $LearnLOD(S_u.A, S_v.A)$
4: **else if** ($weightdiff$ - $gap \leq 0$) **then**
5: $BreakLOD(S_u.F, S_v.F)$
6: $BreakLOD(S_u.A, S_v.A)$
7: **end if**=0

7.6 Experimental Study

We evaluate the proposed approach on a dataset 600 mashups of APIs and their attributes from ProgrammableWeb[1]. We used DBPediaSpotlight[2] for annotating Web service descriptions and categories. We study Web service evolution over several time periods $[T_i, T_j]$ where $T_i \geq 2007$, $T_j \leq 2017$, $[T_j - T_j] = 2$ years. We first identify newly deployed services and those undergoing updates at T_i, after which we run our relationships recommendation. Then we observe each recommended relationship $[S_u, S_v]$ interaction history during the time interval $[T_i, Tj]$. Each mashup containing (S_u, S_v) that is deployed within $[T_i, Tj]$ means a successful composition interaction feedback on the relation (S_u, S_v). On the opposite, if no mashup containing S_u and S_v) is deployed within $[T_i, Tj]$, this is interpreted as a negative interaction feedback. Weight(S_u, S_v) is updated according to col-

[1]www.ProgrammableWeb.com
[2]DBPediaSpotlight

lected interaction feedbacks with the use of Google Sentiment Analysis API[3]. We simulate interactions for substitution relationships.

Figure 7.2 shows the recall and precision rates of the relationship recommendation. Recall, shown in Fig 7.2(a), refers to the the percentage of relationships that are successfully recommended within $[T_i, Tj]$. It is shown that up to 70% of service interactions are successfully recommended by the proposed approach, which is a good result. The justification is that proposed relationships recommendation leverages both service semantic and service interaction history. Leveraging service interaction history shed the light on some relationships that should exist between LOD resources and that do not exist on the original LOD version. Considering those new relationships improves the relationship bootstrapping.

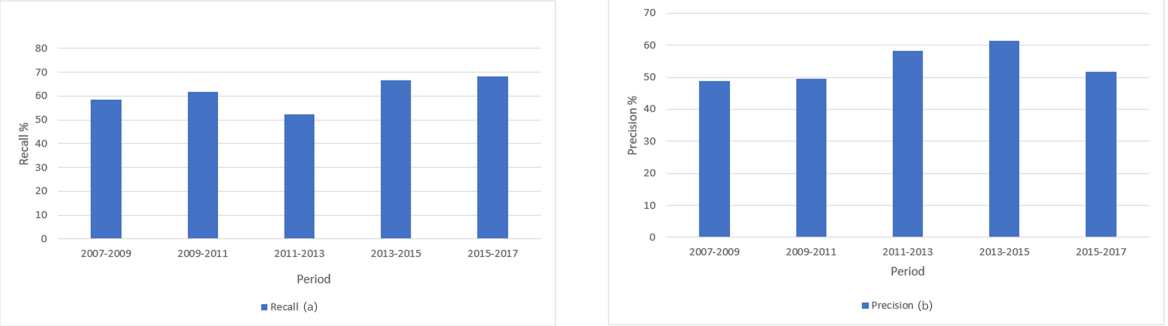


Figure 7.2: Recall and Precision

Precision, shown in Figure 7.2(b), checks the accuracy with which relationship are recommended for newly deployed and updated services. It compares the recommended relationship weights at T_i with real relationship weight at T_j. If the weight difference is less than a threshold value (0.2 in our case), the relationship is assumed to be precisely recommended. Our approach recommends relationships with up to 65% precision. The justification is that weight varies depending on service interaction feedbacks. As such, it gets challenging to predict with high accuracy future relationships weights.

7.7 Related Work

Our work is at the cross-road of three main research fields, namely the Web services social computing, the Web services functionalities evolution management, and linked open data semantic web. The new trend of the research community in service-computing is to exploit social technologies for a better discovery mechanism of services. In this trend, works in [110], [108], [96] propose to build a network of Web services with the purpose of

[3]https://cloud.google.com/natural-language

addressing issues like Web services discovery. Although these approaches handled social interactions of Web services, few efforts have been devoted however to manage Web service evolution that occurs and may affect Web service interaction. Works in [72], [174], and [167] aimed to address Web service functionality evolution. [174] studied the evolution of Amazon Web services to support developers in their tasks of Web service selection and maintenance. [72] proposed an approach for managing services evolution by analyzing the co-occurences of their topics. A topic evolution graph is built according to the analysis of composition patterns in mashups. Resulting graph is used to recommend services for composition. [167] proposed a service evolution model to identify the changes that a Web service is subject to. They classified them into two major categories: Changes that concern Web service Operations and changes that concern Web Service DataTypes.

In our work, we enrich service profiles with LOD-based annotation process which improves service matchmaking accuracy and consequently relationship recommendation accuracy. We also take advantage from user feedbacks to evaluate recommended relationships and reorganize LOD relationships.

Web services evolution has been also at the center of interest of related fields such as IoT and service-based business processes. [156] proposed a framework to automatically capture and communicate services changes in IoT environment. The framework manages syntactic and semantic changes in IoT services capabilities described as RESTful services. In contrast to our work, ontologies have been used to add semantics to services definitions. Ontologies are generally domain-specific which limits management of services functional evolution. We rely instead on LOD that offers more resources to semantically annotated and matchmake services.

7.8 Conclusion

In this chapter, we proposed a LOD approach for managing service deployment and evolution. The proposed approach runs a three phase process. First phase recommends new relationships by matching service capabilities based on LOD. Second phase updates service relationships according to their interaction with each other and with users. Third phase increases recommendation accuracy by learning new relationships in LOD. Managing Web service evolution offers several advantages such as setting long-lasting relationships between service providers and users, and increasing the visibility of newly added functionalities. Experiments conducted on real data returned promising results.

The work described in this chapter was conducted during the scientific stay of Hamza Labbaci in the University of Tours 2018 and 2019 as part of his Phd defended in 2020 [94]. It is published in [95].

Chapter 8

Trustworthy service interactions with Blockchain

Contents

This chapter details our contribution, summarized in Section 2.2.3.2, towards the challenge of handling trustworthy service interactions in dynamic contexts, discussed in Section 2.1.3.2. We present the proposed threefold declarative approach for blockchain-based choreography executions which includes: (i) a transformation model for mashups into smart-contract choreographies executions, (ii) a flexible transactional business process model for smart-contract business process executions, and (iii) a change propagation model to handle dynamic evolution of business process choreographies.

8.1 Introduction

Service-based business processes rely on Web services to implement their tasks. Business processes are designed to achieve business goal which usually involves the collaboration of many entities in decentralized environments. In such open environments, collaborating entities are also competing for their own business success which leads to untrusted behavior. This is the case for business process choreographies which are suitable for open collaboration as entities collaborate directly without intermediaries. They however still need to rely on well-defined collaboration rules and policies to trust each other. These issues have been widely addressed by state-of-the art approaches that usually rely on formal and transactional models to define collaboration rules and on formal verification to ensure execution soundness [163, 22, 77]. These approaches fail, however, to dispense with the need for independent third party to guarantee compliance with trust policies between engaged entities.

The emergence of Blockchain technologies for decentralized and transactional data sharing across a network of untrusted participants brought the right solution that fits well with the requirements of business process choreographies executions. It didn't take long for the Blockchain technologies to be adopted as infrastructure that handles trustful interactions between business entities without any need of any central authority [88, 169, 117, 64, 36]. These approaches mostly rely on second generation blockchains which allow for complex task running through programs called Smart Contracts that enforce the terms of agreement between the untrusted parties. The proposed approaches focus on execution control flow of business processes while leaving transactional flow to the basic blockchain specification that follows standard ACID model which fails to capture advanced reliability requirements of complex business process executions. There is therefore need to define suitable models to handle the transactional flow of blockchain-based process executions.

Collaborating business entities evolve, on the other hand, in highly dynamic environments that appeal for continuous updates and improvements of their internal processes to keep high competition level. These dynamic changes may affect the collaborations in which the enterprise takes part. Changes may also directly concern the global collaboration due to new external business policies that should be taken into account. In both cases, changes must be notified and propagated to the involved partners. Change propagation in process choreographies has also been tackled prior to blockchain-based approaches [65]. In the blockchain-based ones, however, it is still challenging because of the non-mutable nature of shared data. There is therefore need to bring additional levels to cope with the change propagation within blockchain based choreographies execution frameworks.

In this Chapter, we will address the above listed challenges and propose a solution for first trustworthy execution of business process choreographies in Section 8.3. Then, in

Section 8.4, we will extend the proposed approach with transactional mechanisms in order to bring reliability to the blockchain-based business process choreographies. Finally, in Section 8.5, we will render these choreographies adaptive to dynamic changes via proposing the needed change propagation mechanisms. We demonstrate the feasibility of the approach via implemented prototypes and evaluate it through a set of experiments. We use scenarios from the tourism domain with itineraries generated by the CART prototype described in Chapter 5.

8.2 Background

8.2.1 Business Process Choreographies

Business process management (BPM) includes methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes [153]. BPM is ensured and automated by Business process management systems (BPMSs) especially at the intra-organizational level. This first case, mostly known as *orchestration*, faces several issues including central node congestion, additional fees, etc. The second case, known as *choreography*, is more suitable for open collaboration as entities collaborate directly without intermediaries. Choreographies are a distributed way for the composition and control of business processes in an inter-organizational level where the control is not enacted by one single entity and where many parties, that generally do not know neither trust each other, conduct each a certain piece of work and maintain an internal state. The global state of the choreography is obtained through the interactions and message exchanges between the different involved parties.

8.2.2 Transactional Business Processes

Transactional business processes present an evolution of "traditional" business processes to support large flow-based applications with complex control structure. [148, 22]. The transactional approach initially emerged in the context of database management systems with ACID models. Transactional models permit to relax atomicity property to define instead the "failure atomicity" or partial failure of a process activity. In addition, they relax isolation property to allow high inter-process concurrency handling and compensation as backward recovery.

8.2.3 Blockchain and Smart contracts

A blockchain [158] is a distributed ledger where data is organized in sequence of blocks, the containers of transactions. The blockchain network is maintained by independent com-

puters referred to as nodes or peers that do not know or trust each other but can connect and cooperate to validate transactions executed on the blockchain. After validation, peers record, share and synchronize transactions in their respective electronic ledgers. The first generation of blockchain allows only sending and receiving monetary values (e.g., Bitcoin). In the second generation, known as the Smart contracts blockchain (S.C. for short), meanwhile, a transaction allows more complex operations such as the creation of a S.C. and stores the results of function calls in S.C.. A S.C. can be defined as a program enforcing the terms of the agreement between untrusted parties about a valid care sale or a loan assessment or voting or health care tracking, etc. One could consider a contract as a class in object oriented concepts and each deployment of the contract could be considered as an instance of the object. A contract may be deployed to a network multiple times, and each instance would have a distinct address, a secure identifier used to interact with the smart contract.

8.3 Mashup to smart contract transformation model

The starting point of this work is to go a step further configuring service mashups in the tourism context discussed in Chapter 5 to propose a way for personalized trip realization. The deployment and execution of touristic itineraries usually involve many independent entities (tourist, museum, restaurants, hotels, transport, etc.) that cooperate and compete in untrusted distributed context. In order to ensure trust within cooperation and guarantee a successful execution of touristic itinerary, we propose a two-steps transformation model: (a) from data mashup itinerary to process choreography (BPMN) and then (b) from BPMN to smart contract. The CART trip planning outputs itineraries following an XML Schema Model with a root element containing a sequence of steps with or without PoIs and with or without payment. It also contains user constraints such as total duration and maximal budget. PoIs may also have additional attributes. Figure 8.1 shows the detailed structure of the itinerary generic model.

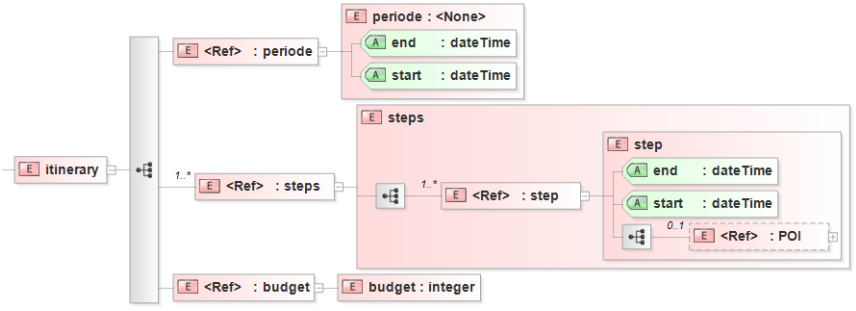


Figure 8.1: A touristic itinerary XML schema

The main transformation rules are given in Table 8.1. We first apply a set of transformation rules which associate: a choreography model for each itinerary instance (rule (a)). Within the choreography model we generate a participant pool for every PoI and for the user (rule (b)) as well as a choreography task for each step of the itinerary (rule (c)). Steps without PoI correspond to private tasks (rule (d)). Additional user constraints and PoI attributes are embedded in the model documentation and in the PoI process documentation respectively (rules (e) and (f)).

Table 8.1: XML to Solidity code transformation rules

Rule	**XML element**	**Choreography element**	**Solidity code**
(a)	Root element: Touristic_Visit	Touristic_Visit choreography model	Touristic_Visit Interaction Smart Contract (TVISC)
(b)	POI	participant (POI) pool	participant address
(c)	Step containing POI	choreography task =send/receive tasks +messages flows	functions in TVISC
(d)	Step with no POI	private task	-
(e)	Tourist's constraints (budget, total visit duration, etc)	embedded in choreography model documentation	Global variable in the TVISC
(f)	POI attributes (fees, opening hours, etc)	embedded in the POI process documentation	Result of the callback performed by Touristic_Visit oracle contract

The resulting choreography BPMN file is in turn used to generate corresponding Smart contracts in Solidity code using the second part of the transformation rules in Table 8.1. Only interaction activities are captured and included in the produced Smart contract which encodes only the public view (rule (a)). Each interaction activity, i.e., choreography task, consists of a send task, receive task and the corresponding message flows. This is encoded as two functions in the Smart contract (rule (c)). We consider Hyperledger Fabric which is a permissioned blockchain where only authorized organizations has the access to the network. In our case, authorized organizations are participant pools in the choreography model (rule (b)). Private tasks are left to participant private process and are therefore not included in the Smart contract (rule (d)). Additional constraints and attributes are encoded as global variables (rule (e)) and callbacks (rule (f)).

In the following, we give a scenario of application of the aforementioned approach. Listing 8.1 presents an excerpt of the XML instance corresponding to the scenario of a tourist, Alice, who wants to visit some attraction in Tours and have dinner in a French

restaurant. This itinerary is transformed into the choreography model presented in figure 8.2. This process choreography model elements are then transformed to their respective Solidity code similarly to the rules in Table 8.1. The output is a smart contract whose generic structure is given in Listing 8.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<itinerary>
    <days>
        <etapes>
            <periode>
                <DebutPeriode>2018-06-17T12:00</DebutPeriode>
                <FinPeriode>2018-06-17T14:00</FinPeriode>
            </periode>
            <poi id="PCU41AASOR100039">
                <Name>Chateau de Tours</Name>
                <Position>
                <Latitude>47.22</Latitude>
                <Longitude>0.40</Longitude>
                <Adresse>25 Avenue Andre ....</Adresse>
                </Position>
                <Type>Point of Interest</Type>
                <Style>Medieval</Style>
        </etapes>

</days>
</itinerary>
```

Listing 8.1: An excerpt of the XML instance corresponding to the motivating scenario

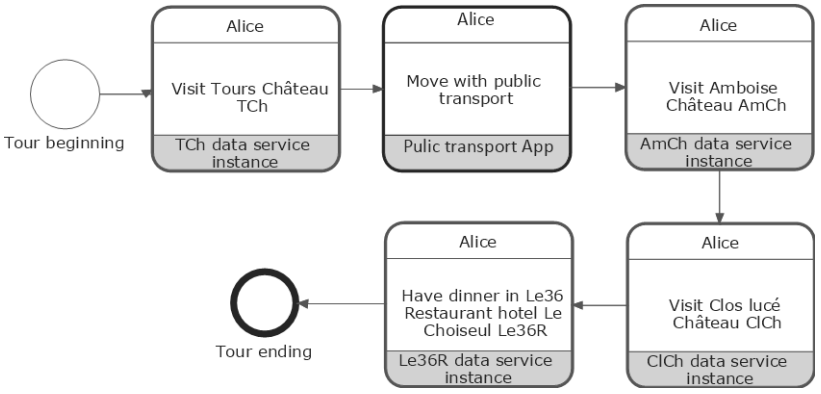


Figure 8.2: Process choreography model of a touristic visit

```
contract TouristicVisit_Oracle_Abstract_contract{
    function Visit_TCh_query_service(uint, uint, bytes32) external returns (uint
        ));
...}

contract TouristicVisit_contract{
    //gloabal variables
    uint public marking = uint(4);
    uint public startedActivities = 0;
    address internal oracleAddress;
    mapping(uint => address) oracleAddresses;
    uint cumulatedFee = 0;
    uint cumulatedVisitDuration = 0;
    uint age = 0;
    bytes32 particularSituation;
    ....
    function TouristicVisit_contract(){
```

```
17          //TouristicVisit_Oracle_contract instance address
18          oracleAddresses[2] = 0x98c9c18f0e3e7c8deec7b2dc9ac5500e2f1fb62c;
19          oracleAddress = oracleAddresses[2];
20      ....}....
21      function Visit_TCh_callback(uint serviceIndex, uint _fee, uint
            _visitDuration, uint _openingHour, uint _closureHour) external returns (
            bool){
22          var(tmpMarking, tmpStartedActivities)= (marking, startedActivities);
23          if (serviceIndex == uint(1)){
24              //TouristicVisit_Oracle_contract instance performs the callback to
                    the corresponding service
25              require(msg.sender == oracleAddress && tmpStartedActivities & uint
                    (2) != 0);
26              {//code to execute like updating cumulatedFee and
                    cumulatedVisitDuration
27              }
28              step(tmpMarking | uint(8), tmpStartedActivities & uint(~2));
29          ...}}  ....
30      function step(uint tmpMarking, uint tmpStartedActivities) internal{
31          while(true){
32              if (tmpMarking & uint(4) != 0){
33                  uint reqId=Visit_TCh_query_service(1, age, particularSituation);
34                  tmpMarking &= uint(4);
35                  tmpStartedActivities |= uint(2);
36              }.....}}
37  }
```

Listing 8.2: TouristicVisit contract

The transformation model is implemented on top of the Caterpillar tool [107]. The reader can find the detailed description of the approach in [36].

8.4 Extending Smart contracts with transactional business process model

The generated interaction smart contract implements the business logic of interaction activities in the process choreography, that is the control flow. The transactional semantics of a smart contract follow ACID models and are therefore limited to support transactional requirements of business process executions [22]. In order to ensure execution reliability of the generated smart contracts we extend them with the Transactional Business Processes models [22, 60] which define the transactional flow on top of the control flow of the interaction activities. Precisely, the transactional flow follows the Flexible Transactional Model principles [148]. The Flexible Transactional Model define transactional properties of activities: *pivot*, *compensatable*, and *retriable* and the transactional mechanisms which include alternatives as forward recovery and compensation as backward recovery. Therefore, we first assign one or many transactional properties to each of input process model activities. Then, we enrich the life-cycle of a process activity, *enabled* → *started* → *completed* with four other possible termination states which are *aborted*, *cancelled*, *failed* and *compensated* which we call transactional termination states.

In this contribution we extend the transformation model introduced in the previous

section and generate both the control flow and the transactional flow as well as failure handling mechanisms. For each business process activity we also generate functions and variables in the smart contract as a transactional flow. Generated functions include life-cycle activities (*ActivityName_start()*, *ActivityName()*, *ActivityName_complete()*, and *ActivityName_fail()*) and transition activities (*abort()*, *cancel()*, *complete()*, *fail()*, and *retry()*) which allow the activity to move from one state to another. Variables include state vector *StateVector_ActivityName* to capture the activity state evolution for each activity.

Following a task failure and depending on the activity's transactional properties, the Solidity code generation templates produce the code of the corresponding functions. In Listing 8.3, we present an excerpt of the BPMN-to-Solidity template corresponding to the code generated in case of a pivot activity failure. The proposed approach is implemented on top of the Caterpillar tool [107]. Further details of the proposed approach can be found in [37].

```
1  //state vectors
2  <%  groupedIds = getWorkItemsGroupByParameters(false);
3      groupedIds.forEach(idGroup => { -%>
4       uint public StateVector_<%= nodeName(idGroup[0]) %> = 0 ;
5  <%  }) -%>
6  ...
7  //for all pivot activities
8  <%  groupedIds = getWorkItemsGroupByParameters(false);
9      groupedIds.forEach(idGroup => {
10       idGroup.forEach(nodeId => {
11        let node = nodeMap.get(nodeId),
12          transactionalProperty = nodeTransactionalProperty(nodeId),
13          nodePreMarking = preMarking(nodeId),
14          AllPivot = [];
15     if ( transactionalProperty  == "pivot" ) {-%>
16
17     function <%= nodeName(idGroup[0]) %>_fail(uint elementIndex<%=
           concatParameters(idGroup[0], false,true, true) %>) external {
18         var (tmpMarking, tmpStateVector) = (marking, StateVector_<%= nodeName(
               idGroup[0]) %>);
19         if(elementIndex == uint(<%= nodeRealIndex(nodeId) %>)) {
20             require(msg.sender == oracle && tmpStateVector & uint(4) != 0);
21             StateVector_<%= nodeName(nodeId) %> = fail_task(tmpStateVector);
22             step(tmpMarking | uint(<%= postMarking(nodeId) << 1 %>  ),
                   StateVector_<%= nodeName(nodeId) %>);
23             return;
24             }
25 <%      AllPivot.push(nodeRealIndex(nodeId)); -%>
26         }
27 <%  } -%>
28 <% }) }) -%>
29 ...
30 //code to generate in function step if the current pivot fails
31  function step(uint tmpMarking, uint tmpStateVector) internal {
32     while (true) {
33     ...
34         if (tmpMarking & uint(<%= nodePostMarking << 1 %>) != 0 &&
               tmpStateVector == uint(64) ) {
35             //remove token
36             tmpMarking &= uint(~<%= nodePostMarking << 1  %>);
37             //compensate the work already done
38 <%          let index = allpivot.indexOf(nodeRealIndex(nodeId));%>
39 <%          let PrevNodeList= []; //PrevNodeList: a list containing ids of
       previous compensatable activities until the previous pivot
40             for (let i = nodeRealIndex(nodeId); i >= indice; --i) {
41                 if (nodeTransactionalProperty(nodeList[i]) == "compensatable" ){
42                  PrevNodeList.push(nodeList[i]); -%>
```

```
43 <%          }}%>
44             //for each node in PrevNodeList update StateVector and call
                   compensation activity
45 <%          for (let i = 0; i <PrevNodeList.length; i++) {%>
46                 StateVector_<%= nodeName(PrevNodeList[i]) %>= compensate_task(
                       StateVector_<%= nodeName(PrevNodeList[i]) %>);
47 <%=               nodeName(processId()) %>_Oracle.<%= nodeName(PrevNodeList[i]) %>
       _cp_start(<%= nodeImaginaryIndex(PrevNodeList[i]) %><%= concatParameters(
       nodeId, true, false, true) %>);
48 <%          }%>
49             ...
50 <%          var node = nodeMap.get(nodeList[indice]) -%>
51 <%          if (is(node, 'bpmn:ExclusiveGateway')) { -%>
52             //case there is an alternative --> start it
53 <%          }else{
54             //case of no alternatives --> abort non started activities
55             for (let i = nodeRealIndex(nodeId) ; i< nodeList.length ; i++) {
56                 if(allActivity.includes(nodeList[i])){ %>
57                     StateVector_<%= nodeName(nodeList[i]) %>= abort_task(
                           StateVector_<%= nodeName(nodeList[i]) %>);
58 <%              } } } -%>
59         continue;
60         }
61     }
62 }
```

Listing 8.3: Failure handling code generation for pivot activity

For example, when a pivot activity fails, a function called *ActivityName_fail()* (line 17 in Listing. 8.3) is generated. This function: (i) updates the state vector of the current task to *failed* (line 21); (ii) updates the value of the marking variable; and (iii) calls the step function to execute the actions to take (line 22). The latter are:

- compensate the set of activities situated between the current and the precedent pivot (each activity between two pivots or before the first pivot must be compensatable) (lines 38-48 in Listing 8.3);
- start the next alternative (lines 51-52 in Listing. 8.3);
- else abort activities that are at the state initial when no alternative exists after the previous pivot (lines 55-58 in Listing. 8.3).

Up to this level of the contributions, we use imperative paradigm (BPMN) to model and implement business process choreographies. In fact, imperative business process models are largely used in the scientific community. The principles of imperative models are aligned with the transactional approach we implemented. And, we consider only the shared view of the choreography. In the next Section, we migrate to declarative business process models called Dynamic-Condition-Response (DCR) graphs. Declarative models are more flexible and light-weight process descriptions than imperative models. Moreover, the declarative paradigm is aligned with the dynamic nature of business process choreographies and will be used to better model and execute adaptive business process choreographies.

8.5 Change propagation in Blockchain-enabled business process choreographies

8.5.1 Declarative choreography modeling and change definition

Collaborating business entities evolve in highly dynamic environments that appeal for continuous updates and improvements of their internal processes to keep high competition level. These dynamic changes may affect the collaborations in which the enterprise takes part. Changes may also directly concern the global collaboration due to new external business policies that should be taken into account. In both cases, changes must be notified and propagated to the involved partners. Thus, business process management systems should manage change propagation in a trustworthy fashion. Although change propagation in process choreographies has been tackled in general [65], it is however still challenging for blockchain-based choreographies because of the non-mutable nature of shared data.

In running choreography instances, a change may consist of a simple operation, ADD/REMOVE/UPDATE activity, or a combination of operations. Handling such changes requires the underlying choreography model to be flexible. We therefore consider declarative choreographies modeled through Dynamic-Condition-Response (DCR) graphs [127, 128]. With a **DCR graph** $G = (E, M, Rel)$, processes are modelled as a set of *events* E, corresponding to the activities in BPMN, linked together with *relations* Rel modeling linking constraints. Relations are of five types (*condition*, *response*, *milestone*, *include*, and *exclude*) [85]. The graph runtime state is captured by a marking matrix M of the events through a triplet of binary values corresponding to (*currently included*, *currently pending*, *previously executed*). In [84], authors propose an approach for trustworthy execution of DCR choreographies. We briefly describe here our contribution which builds on and extends this work with the change management mechanism.

In [84] a **DCR choreography** is defined as $C(G, I, R)$ where G is a DCR graph, I is a set of *interactions* and R is a set of *roles*. An interaction i is a triple (e, r, r') in which the event e is initiated by the role r and received by the roles $r' \subset R \setminus \{r\}$. Each partner has a projection of the DCR choreography, called public DCR process, in addition to its private DCR process which refines the public process with local events of the partner. The shared events and interactions are managed through the choreography smart contract on the Blockchain.

We propose to extend this approach with change management through the choreography smart contract. We define the **change element** G_{Ref} as a refinement of C which occurs on private DCR process and should be propagated to the public DCR process whenever the change impacts common interactions with other partners. G_{Ref} can be either an atomic

element (an event or an interaction relation) or a DCR subgraph. When G_{Ref} impacts common interactions, we manage a negotiation phase that leads either to a consensus in which case the public process is updated accordingly or to failure in which case the change is rejected. A **change operation** can be of three types respectively corresponding to ADD, REMOVE, and UPDATE: $C \oplus G_{Ref}$, $C \ominus G_{Ref}$, and $C[G_{Ref} \mapsto G'_{Ref}]$.

Figure 8.3 presents the trip planning scenario that was initially presented in the introduction (c.f. Section 7.1) translated into DCR: Figure 8.3(a) presents the DCR choreography and Figure 8.3(b) presents the tourist private DCR process. The choreography process is managed in the different partners processes, namely Tourist, TouristOfficer, Hotel, and CastleAdmin, to ensure a separation of concerns. *PayPass (p3)* is an internal event of the role Tourist, managed off-chain to preserve the privacy of Tourist. *PurchasePass (e1)* is a choreography interaction sent by Tourist and received by TouristOfficer. It is managed on-chain (c.f. [84]). To execute the send event, Tourist triggers the SC from its private DCR process. Table 8.2 shows the choreography markings of Figure 8.3 during a run. Each column stands for the events of the choreography. Rows indicate markings changes as events on the left are triggered. For example, initially no event is executed nor pending and the event *e1* is included. Thus, its marking is (1,0,0). Once Tourist executes *e1*, the marking becomes (1,0,1). Partners have control over the set of internal and choreography interactions they are involved in. This set of events, hereafter referred to as a partner private DCR process, is illustrated by the tourist's one in Figure 8.3(b).

Table 8.2: Evolution of the markings (included, pending, executed) of the DCR choreography process in Figure 8.3 (before changes)

	Markings			
	e1	e2	e3	e4
(init)	(1,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
e1	(1,0,1)	(1,1,0)	(0,0,0)	(0,0,0)
e2	(1,0,1)	(1,0,1)	(1,1,0)	(1,1,0)

Both choreography and private DCR processes in Figure 8.3 are susceptible to changes. A change is composed of a set of change elements and a combination of change operations. We define in the following these two concepts. Change operations are presented in colors in Figure 8.3. An example of an UPDATE operation is change #1 in red in Figure 8.3. Here, the pass purchased by the Tourist in the event $e1$ undergoes a change: it will let the Tourist to have a dinner in NewRestaurant instead of having it in the Hotel. Hence, the TouristOfficer, who manages the pass and can add new participants, establishes a new convention with NewRestaurant. Consequently, the change operations to make are: (i) add the partner NewRestaurant, (ii) an UPDATE operation where the interaction *e4* is replaced with the DCR fragment$\{e5,e6\}$.

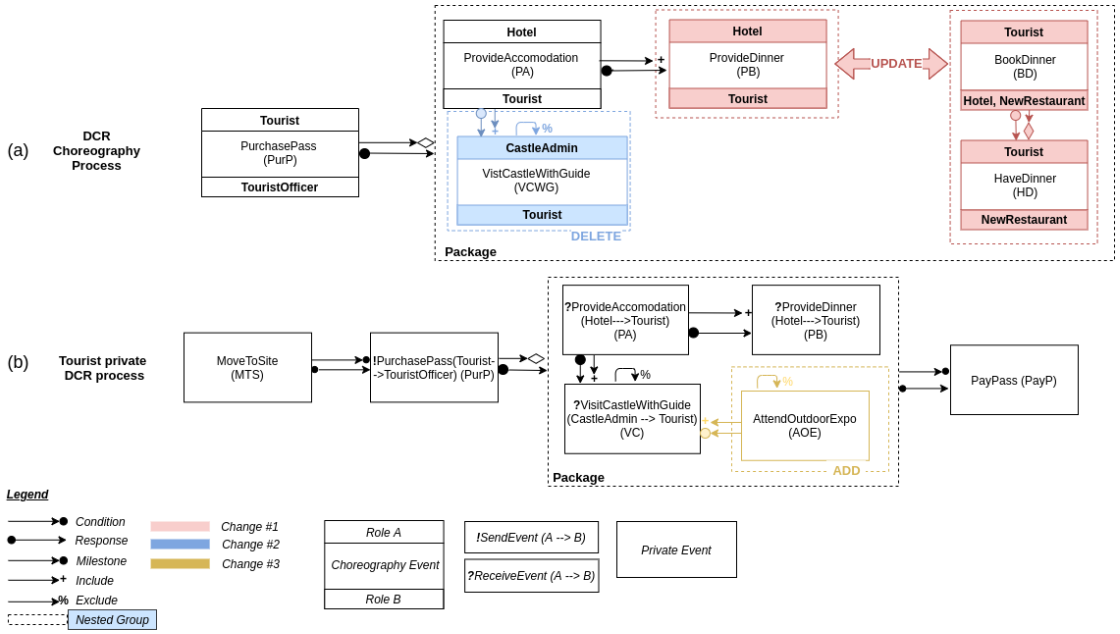


Figure 8.3: DCR choreography process and tourist private DCR process of the trip e-booking process

8.5.2 Approach description

We describe hereafter the four-steps approach we propose to manage these changes.

(i) Change proposal. The role initiator defines the change of its private DCR process following the previously introduced possible change types and operations. We review a set of integrity rules [58] to ensure safety and liveness of the updated DCR graph to avoid cycles. Safety means that the DCR graph is deadlock free whereas liveness guarantees the ability of the DCR graph to completion. The integrity rules are of two types and enumerated in Table 8.3. *Allowed* change rules define what can be done on the graph while *Denied* ones define what should be rejected. When the change proposal satisfies these rules the private DCR process is updated. In the case where the change also concerns the public DCR choreography we move to the next steps.

Table 8.3: Proposed allowed and denied changes for a DCR process

Type	Rule
AR1	Change condition / response / milestone relations
DR1	Inclusion of an excluded event
DR2	Exclusion of an included event
AR2	Block temporarily/ permanently an included event

Algorithm 3 Request change smart contract function

Data: *changeRequests* the list of change requests, E the list of endorser addresses, h_{curr} the current ipfs workflow hash, h_{req} the ipfs hash of requested change description, $t1$ the deadline timestamp for change endorsement, and $t2$ the deadline timestamp for change propagation

Result: emits change request notifications to endorsers

1 **Function** `requestChange`(h_{curr}, h_{req},$E, t1, t2$)**:**
2 require *msg.sender* belongs to the list of business partners
3 **if** $changeRequests[h_{curr}].status == Init$ **then**
4 set $changeRequests[\text{h}_{curr}].\text{h}_{req} \leftarrow h_{req}$
 set $changeRequests[\text{h}_{curr}]$.status $\leftarrow$ "$BeingProcessed$"
 set $changeRequests[\text{h}_{curr}]$.initiator $\leftarrow$ *msg.sender*
 set $changeRequests[\text{h}_{curr}]$.endorsers $\leftarrow E$
 set $changeRequests[\text{h}_{curr}]$.t1 $\leftarrow t1$
 set $changeRequests[\text{h}_{curr}]$.t2 $\leftarrow t2$
 emit RequestChange(h_{curr}, h_{req}, E, *msg.sender*);
5 **else**
6 emit Error; `// an ongoing change request is being processed`
7 **end**
8 **End Function**

(ii) Change request for public-related changes. The change request is sent to the choreography smart contract which stores the list of change requests assigned to process instances. Algorithm 3 presents the SC function registering a change request. Ongoing process instance changes are recorded with the identification hash of the current process instance. During the change request lifecycle, the request is assigned to a status: *Init* if no change request is ongoing, *BeingProcessed* during the negotiation stage, *Approved* or *Declined* once the change request is processed by all endorsers. The smart contract checks the identity of the change initiator which should belong to the list of partners, checks that no other change request is being processed, and then creates the change request and sends change request notification to the involved partners. The change initiator also sets two response deadlines t1 for change endorsement and t2 for change propagation to be checked by the smart contract. If one of the change endorsers does not reply before deadline t1 during endorsement or t2 during propagation, an alarm clock triggers a smart contract function cancelling the change request. The smart contract function sets the change request status to cancelled and emits an event notifying partners that the change has been cancelled. In this way we prevent any deadlock that could occur due to one of the partners not responding.

(iii) Change negotiation for public related changes. Algorithm 4 presents the SC function receiving one endorser's decision. All partners subscribe to the change request

Algorithm 4 Endorser decision management smart contract function

Data: *changeRequests* the list of change requests, e_s the endorser address, E the list of registered endorsers, h_{curr} the hash of the current workflow, h_{req} the hash of the desired workflow, rsp the endorser response $\in \{0, 1\}$

```
9  Function endorserRSP(h_curr, e_s, rsp):
10   require(block.timestamp <= changeRequests[h_curr].t1)
     require(e_s ∈ E)
     require(changeRequests[h_curr].changeEndorsement[e_s] != 1)
     require(changeRequests[h_curr].status == "BeingProcessed")
11   if rsp == 1 then
12     set changeRequests[h_curr].changeEndorsement[e_s] ⟵ 1
13     emit AcceptChange(h_req, e_s)
14     lockInstanceChecker(h_curr)
15   else if rsp == 0 then
       // declineapprovalOutcomes
16     set changeRequests[h_curr].status ← "Declined"
       emit DeclineChange(h_req, e_s)
17   else
18     emit Error(h_req, e_s)
19   end
20 End Function
```

events emitted by the smart contract. Endorsing partners must send their decision request to the smart contract based on the *Allowed*/*Denied* changes rules. If the change once computed on the endorser's process respects all the rules, then the endorser approves the request. It is otherwise rejected. The smart contract collects all the decisions from the endorsers to lock (or not) the choreography instance and proceed (or not) with the change and update the change status to *Approved* or *Declined*. In case of approval the smart contract locks the instance for change propagation. As it manages the negotiation process, a tamper-proof record of the negotiation is accessible by all partners. This prevents conflicts and eases potential claim resolutions.

(iv) Change propagation. Change propagation after successful negotiation phase consists in (i) applying the change effect to the affected partners DCR public processes and (ii) propagating the change effect by each partner to its private DCR process. Algorithm 5 presents the function triggered by partners to confirm the projection to the smart contract. The smart contract receives all the local projection confirmations and notifies the change initiator (*didPropagate* is filled with ones). The change initiator then retrieves the new DCR choreography that was saved into IPFS using h_{req} and forwards it to the smart contract. The SC updates the relations and markings stored into the process instance and resets the change status of the workflow instance: a new change request can be processed.

Algorithm 5 Confirm change propagation smart contract function

Data: *changeRequests* the list of change requests, e_s the sender address, E the list of registered endorsers, h_{curr} the hash of the current workflow

Result: manages the record of projections of the new public view

Function `confirmProjection(`h_{curr}, e_s`):`

  require($e_s \in E$) require($block.timestamp <= changeRequests[h_{curr}].t2$)
  require(changeRequests[h_{curr}].didPropagate[id] != 1)
  set changeRequests[h_{curr}].didPropagate[id] $\longleftarrow$ 1
  emit LogWorkflowProjection(h_{curr})

End Function

8.5.3 Change propagation correctness

To ensure the correctness of the change propagation, we introduce two correctness properties that check (i) the compatibility between the DCR public processes of the participants and (ii) the consistency between one partner's private and public process.

Compatibility between participants public processes. By relying on the rules in Table 8.3 and to ensure the compatibility between public processes, we introduce the following property where: c is a change, r, E are respectively change initiator and endorsers, G_r, $G_{r'}$ are respectively the public DCR process of r and r' where $r' \in E \setminus r$, $effect(c)_{\|G_r}$ the effect of the change c over r, and $effect(c)_{\|G_r} :\Rightarrow G_r$ refers to computing the new state of the graph G_r after applying the effect of c over r, then the proposed property stands as follows:

Property 1. *if* $effect(c)_{\|G_r} : \Rightarrow G_r$ is correct-by-construction and $\forall r' \in E$, $effect(c)_{\|G_{r'}} : \Rightarrow G_{r'}$ is correct-by-construction *then* G_r and $G_{r'}$ are compatible $\forall r \in E$, $\forall r' \in E \setminus r$.

Consistency between participants public and private processes We define the private DCR process P_r of a role r as a subclass of the public DCR process G_r. This leads us to **Property 2.**:

Property 2. $\forall r \in E$, $P_r = G_r + N$ where N is the new behavior. P_r is a subclass of the super class G_r and $G_r = P_r + \tau_\epsilon(N)$, i.e., P_r and G_r are behaviorally equivalent.

A detailed description including proofs of the previous properties is provided in [35].

8.5.4 Implementation and experimental evaluation

An implemented prototype of the aforementioned approach is realized . One smart contract is deployed per choreography process using Ganache testnet. Each partner can edit the running instance via the panel manager, a tool to update DCR graph descriptions. Users can add private and choreography interactions, as well as condition, response, include, exclude, and milestone relations. They can also use the panel manager to remove and

update events and relations. The panel manager implements integrity rules presented in Table 8.3: the panel verifies the soundness of a desired change operation. Hence, we obtain a redesigned DCR graph that is correct-by-construction.

We evaluate the smart contract deployment cost as well as the transactions costs related to change negotiation and propagation operations. Table8.4 presents the gas usage induced by the execution of the SC during the negotiation and propagation stages. Regarding the negotiation phase, the transaction fees for placing the change request of an UPDATE operation are 0,00639582 ETH, and are the highest fees of the negotiation stage. Indeed, the fee to be paid to decline or accept a role is worth around 0.0015 ETH. Nonetheless, all fees are of the same order of magnitude (0.001 ETH). Regarding the propagation phase, many transactions fees have to be considered like for example the cost of sending notifications of the local update which is is worth 0,00201296 ETH and 0,0020115 ETH for both endorsers (around 5$ per local projection). We analyze the costs and find that propagation transaction fees are higher than the negotiation ones, given the fact that the propagation phase is longer and comprises many atomic transactions, mainly comprises the cost of the DCR choreography update.

Table 8.4: SC change propagation gas costs and gas fees

Stage	**Step**	**Partner**	**Gas**	**Cost(ETH)**	**Cost($)**
Nego.	LaunchNego	NewRestaurant	213194	0,00639582	16,513
	Case Decline	TouristOfficer	46773	0,00140318	3,623
	Case Accept	TouristOfficer	78999	0,00157998	4,079
		Tourist	86428	0,00181256	4,68
Propag.	Upd. projection	TouristOfficer	96448	0,00201296	5,197
	Upd. projection	Tourist	96375	0,0020115	5,193
	Upd. projection	NewRestaurant	87648	0,00175296	4,526
	Upd. SC instance	NewRestaurant	1321496	0,02642992	68,238

The reader can find further details of the proposed approach can be found in [35].

8.6 Related Work

Regarding the monitoring and execution of business process collaborations using the blockchain, authors in [73, 169], propose an approach translating a BPMN model to a minimized Petri-net and compiling the latter to a smart contract. A derived tool called Caterpillar implements this approach [107]. However, participant processes are modelled as lanes belonging to the same pool and not as process choreographies. In [140], a runtime verification mechanism is developed on top of the Bitcoin blockchain to verify the correct execution of process choreography instances. In contrast, they make some assumptions. A

particular participant called the process owner who initiates the business process execution hands over the control to a first suitable partner to have a specific process activity executed. Then, the selected partner passes the execution to another choreography participant to perform the next task and so on.

Transactional reliability has been tackled prior to blockchain-based approaches. Although they bring many opportunities to business process execution, transactional models have been applied to orchestration schemes only [22, 60]. They guarantee process execution reliability, that is, the process reaches its objectives and complete successfully or it properly undo all undesirable effects of partial execution. Such features can be used to add transactional reliability to smart contract related transactions, very close to ACID models. Looking to proposed approaches dealing with business process monitoring and execution using the blockchain technologies, they handle only the control flow execution of business processes without taking into consideration its transactional flow. Thus, they do not offer transactional reliability in case of task failures and subsequent behaviour of the business process in such cases.

Finally, in regards to approaches dealing with changes of business process collaborations when they evolve in dynamic environments, they have been also proposed prior to blockchain-based ones. In [65, 89], authors propose change propagation algorithms across business process choreographies. However, authors do not propose any mechanism to ensure that all partners have trustfully applied the change, and no blockchain is used to deal with this problem. Change management has also been studied in DCR processes, but only in orchestrations schemes. In [128], authors propose add/remove/update operations to DCR graphs relations and events. In [58], authors use a set of rules ensuring the correctness of new instances of DCR orchestrations by design. In [106], authors propose an approach that allows collaborative decisions in blockchain-enabled collaborations. Their approach offers flexibility operations such as: (1) late binding and un-binding of actors to roles in blockchain-based collaborative processes, (2) late binding of subprocesses, and (3) choosing a path after a complex gateway. Authors, though, do not consider evolutionary changes like we do. Moreover, their flexibility operations currently push the burden of checking the transitive effect of new changes onto the new parties. This checking, likely done in a manual way, which can lead to errors.

8.7 Conclusion

In this Chapter, first, we proposed a solution for trustworthy execution of business process choreographies modeled in BPMN. Process designers apply a set of transformation rules in order to transform business process choreographies into smart contracts. Then, a

smart contract called Interaction SC monitors on-chain the execution of the shared view of the choreography.

We extended afterwards the Interaction SC with transactional mechanisms. These mechanisms include alternatives as forward recovery and compensation as backward recovery. This ensures that the process choreography execution ends in a consistent way even in case of activities failures.

Finally, we presented a protocol enabling adaptiveness to declarative business process choreographies modeled as DCR graphs. This protocol allows (i) partners to first negotiate the change on-chain, (ii) then to dynamically update the choreography process instance managed by the smart contract with the new process change information, and (iii) finally propagate this information across partners processes affected by the change. We demonstrate as well the correctness of the change propagation across participants processes. To do so, we proposed two correctness criteria: (i) the compatibility between the DCR public processes of the participants, and (ii) consistency between one partner's private and public process.

The work described in this chapter was conducted during the PhD of Amina Brahem to be defended in 2023. It is mainly published in the following conference papers: [36], [37], and [35].

Part V

Conclusion and Perspectives

Chapter 9

Conclusion and perspectives

Contents

This chapter wraps up the main matter of this document by presenting directions for my future research starting by the ongoing work as first step toward them. My future work shares the same general objectives as my previous activities namely semantically-enabled, context-aware, user-centric and trustworthy service interactions. My ongoing research activities mainly tackles the challenge of enhancing personalization and context-awareness in user-centric service recommendation through a combination of Attention Mechanism with Machine Learning and Deep Learning models. My future work extends my previous and current work into the following two research dimensions: Internet of Things services [40] and Distributed Computing Continuum Systems [24].

9.1 Conclusion

This dissertation draws the contributions I achieved with my colleagues and PhD/-Master students during the last ten years. Our main focus was to bring innovative and sound approaches to support service-based solutions ensuring successful interactions in service-service and user-service interactions within most general perspectives of Business-to-Business and Business-to-Customer contexts. We considered typical research issues related to service oriented computing with a specific focus on service discovery and recommendation. Our methodology consists in bringing successful advances in complementary topics such as semantic Web and Machine Learning as well as sound formal frameworks to tackle service interaction issues. We address service mismatch issue through the definition of semantic similarity measure first based on ontologies and then extended to Linked Open Data (Part 2: Chapter 3 and Chapter 4). We address context-aware service recommendation through a Configuration based approach together with LOD based semantic similarity (Part 3: Chapter 5). We tackle the issue of service migration in user-centric contexts through a review of Liquid Software and RESTful principals (Part 2: Chapter 6). We handle service evolutions through the use of Machine Learning models combined with LOD based similarity and Service Social Network graph modeling (Part 3: Chapter 7). And, finally, we address trustworthy dynamic service interactions through Blockchain and Smart contracts (Part 3: Chapter 8).

While our work is in line with current research issues in the field of Web services, it also responds to real needs expressed particularly in the tourism domain within the Smart Loire research project. Although the contributions described in this dissertation was achieved separately, they complement each other to provide a coherent solution for personalized trip recommendation and realisation to meet the Smart Loire project main objectives. The Smart Loire project allowed in turn to evaluate the proposed approaches on real-world datasets and by real users, in addition to publicly available test datasets and baselines.

Service interaction research topic is highly dynamic and continuously evolving. Our contributions described in this dissertation are therefore perfectible in several directions. Some of them are being investigated as ongoing and short-term perspectives. We give an insight in Section 9.2. Some others are longer term following visionary roadmap elements in service oriented computing. We describe our vision to tackle them in Section 9.3.

9.2 Ongoing research activities and short-term perspectives

Our most elaborated ongoing work tackles the challenge of enhancing service recommendation with Attention Mechanisms in order to achieve high level of personalization

and context-awareness. The following subsections give insights on the general idea of our undergoing contributions. We also address privacy concerns in user-centric IoT device recommendation.

9.2.1 Attentive knowledge based service recommendation

Attention Mechanisms (AM) [15] was first proposed as an optimisation technique for machine learning models to make them selectively focus on the most relevant parts of the input data when making a prediction, which helps achieving accurate predictions while running more efficiently. AM have then been quickly adopted and integrated in many AI-based service recommendation approaches which have to process an overwhelming number of available services [168] [104]. AM fit well more particularly context-aware recommendation approaches where the context helps defining important features relevant services should match [159].

However, the use of AM for AI-based context-aware service recommendation rises two major questions. First, which features should be considered as important within the attention mechanism and how to model and learn them ? And, second, how to combine important feature selection with the recommendation learning model ?

9.2.1.1 Attention Mechanisms and Knowledge Graphs

We try to answer the previous questions in the Tourism context and we rely on our previous work on LOD-based data augmentation applied on DataTourism ontology (Chapter 5, Section 5.6). We propose to model AM through Knowledge Graphs (KG). KG provide an efficient form of knowledge representation that captures the semantics of Web objects based on different entities and their relationships. In addition to AM support, KG allow for recommendation interpretation. Indeed, by reasoning over a KG in a node-by-node manner, the connectivity between entities can be discovered as paths that not only reveal the semantics of entities and relations, but also serve as an explanation reflected through a simulated decision-making process which provides explicit semantics for the explanations.

Starting from LOD-augmented DataTourism ontology, we construct first a travel-domain specific knowledge graph (TKG) gathering fine-granular attraction types that allows a deep convergence of all the data and full considerations of their semantic relations. We apply a path extraction process to extract the set of candidate paths. A travel path models a set of entities connected through semantic relationships. Each travel path corresponds a possible mashup of services corresponding to Point of Interest (PoI) entities in the path. Additional -non PoI- entities and semantic relationships will be considered for the mashup explainability. Then, we model the set of travel paths using a recurrent network based

on Gated Recurrent Unit (GRU) technique. User preferences are encored as a Map of weighted features as part of the attention mechanism which also includes the attention function. We consider as attention function the dot product applied to user preferences and travel paths to provide an attention score for each travel path. This allows to keep only the highly ranked paths as relevant recommendations. The first results of our work in this direction are interesting and have been published in [30].

9.2.1.2 Attention Mechanisms and Deep Learning

The previously discussed research direction is based on a unique layer that encodes AM based on user preferences and a single attention function applied as final selection step for mashup recommendation. As a second direction, we consider to spread the attention mechanism over multiple layers of a deep learning process for personalized service recommendation. The idea is apply attention mechanism at three different levels: (i) functional service feature mining, (ii) non-functional service feature mining, (iii) user preference mining, and (iv) overall attention aggregation. The idea is applied on programmableweb dataset and consists in the following steps. We first discover mashups with similar functionalities. Afterward, we build a composite service network that learns function-related features of services based on tags and description information. In parallel, based on user feedback, we learn the non function-related features of services. Then, the recommendation module uses attention aggregation to couple the two service representations and generate the final recommendation result. The first results of our work in this direction are interesting and have been published in [32].

9.2.1.3 Attention Mechanisms and LSTM for Long-term Service QoS Forecasting

Our previous approach recommends accurately potential service collaborations. The proposed recommendation focused the matching/learning on the functional behaviour of web services. In practice, the success and longevity of such collaborations depends on the ability of the collaborating parties to deliver high Quality of Service (QoS) standards in the long-run. Indeed, web services with comparable functionalities maybe differentiated according to their QoS attribute values such as *response-time* and *throughput*. This can be attributed to the differences of their service providers, leading to variability in their functional behaviour. That is, besides accurately recommending functionally composable services, it is crucial to figure out services that are likely to deliver comparable QoS levels over time.

Ensuring long-term success of service interactions and long-run service interoperability

is still challenging. Furthermore, existing techniques predominantly focus on static QoS values observed during composition time. We believe that sequential Long Short Term Memory (LSTM) networks can bring reliable solutions for foreseeing long-term QoS fluctuation over time, hence enhancing the overall service composition. LSTM is a deep learning model adapted to sequential data [86]. It allows to capture information dependency contained in sequential data for a long period of time and uses this information to enhance prediction accuracy. LSTM have been successfully combined with Attention Mechanisms on functional and contextual features for Web service recommendation in mashup creation [150]. Our aim is therefore to extend the LSTM networks predictive ability by integrating attention mechanisms for Long-term Service QoS Forecasting.

Handling attention for time series forecasting [1] assigns different weightage to different parts of the input sequence while predicting each element of the output sequence, allowing the model to refer back to the most relevant parts of the input and weigh them more heavily. Attention mechanisms augment the ability of LSTM to manage long sequences, enabling the model to capture long-term dependencies more effectively by focusing on the most relevant parts of the input sequence. When it comes to Web services, multiple QoS attributes such as bandwidth, latency, throughput, availability, and response time are crucial, and each can have a significant influence on the overall quality of the service. Since these attributes can be interrelated, employing a multivariate LSTM model can be beneficial to capture the latent relationships and dependencies between different QoS attributes. The integration of attention mechanisms into LSTMs enables the model to dynamically allocate varying levels of importance to different time steps of each QoS attribute, depending on their relevance to the prediction task at hand. Then the model will be able to focus on crucial portions of the input sequences (i.e., historical data of QoS attributes like bandwidth) that are more indicative of future values, thus improving the forecast's precision.

9.2.2 Privacy-aware user centric IoT device recommendation

This perspective extends our research interest to the Internet of Things (IoT) related topics. IoT as a paradigm, aims to connect different pervasive devices and smart objects to the internet and provides means for data communication between them. Today, the IoT includes an unprecedented number of connections between things as well as between things and people [68]. According to Statistica[1] the number of connected IoT devices exceeds 15 billion in 2023 and could reach 30 billion by 2030. This rapid evolution is supported by the proliferation of devices of multitude of types, protocols, computing and storage capacities, etc. ranging from the basic sensors to the most sophisticated smartphones and virtual reality headsets. One of the fundamentals of IoT is the ability to create networks of devices

[1] https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

working together to accomplish certain goals in different domains such as smart-homes and healthcare. Ready-made solutions including complementary devices and dedicated management systems may exist for common use. However, when it comes to more personalized use or new goal, identifying relevant devices among the sea of existing ones is still challenging [90]. Moreover, usually acting as data generator, IoT devices naturally face privacy and trust challenges. These challenges are even accentuated regarding (i) the limited device capacity making it possible to hold and manage security mechanisms, (ii) the dynamic contexts in which devices evolve, (iii) and the heterogeneity of collaborating/interacting devices [24].

We tackle both challenges from a user-centric perspective and following W3C recent recommendations towards a standardised and interoperable IoT, the Web of Things (WoT)[2] and Social Linked Data (SOLID)[3] architecture for privacy management. WoT intended to enhance IoT device interoperability by using existing Web standards such as HTTP and REST for data communication. It also includes Thing Description model for homogeneous description of devices and their capabilities. SOLID is a personal data management architecture intended to grant data owners full control over their personal data [112]. It is based on two main components: (1) the Personal Online Store (POD) server where data is being stored, semantically annotated and accessed by third parties according to the data owner's permissions and preferences, and (2) the SOLID application that allows users to register, add, view, and edit data on PODs.

Starting from these elements, we propose a privacy-aware user-centric architecture which relies on the following three main steps. (i) First, we extend LOD-based annotation and similarity calculation discussed in the previous chapters to apply on devices described following Thing Description model. (ii) Second, we implement SOLID architecture to allow for privacy management. (iii) And third, we propose a recommendation algorithm that considers the LOD-based similarity as well as a local user-user social relation network. A first achievement in this direction is presented in [92]. However, there is still a lot of work to be done to achieve high level privacy and interoperability within such a framework. We consider to improve it in two directions. First through extending the privacy management SOLID architecture with a more dynamic permission manager. This will allow for dynamic and evolving privacy. Second the recommendation algorithm should consider permissions prior to recommending any device to ensure its usefulness. One more long-term perspective consists in considering IoT services [24] as functional abstraction of devices within a more general user-centric architecture. We will discuss this idea with other interesting long-term perspectives in the next section.

[2] `https://w3c.github.io/wot-architecture/`
[3] `https://solidproject.org/`

9.3 Mid-term and long-term perspectives

9.3.1 Handling IoT services: a user-side perspective for interactions, privacy, and trust

Transitioning from traditional web services to IoT service computing is fundamentally motivated by massive deployment of new devices for various domains, and how do such devices change the way services are provided and received. This crucial shift to IoT enables the delivery of more responsive and intelligent services, unveiling opportunities in various fields such as enhanced data gathering and analysis, real-time monitoring and adaptive user experiences. Such transition will open up key opportunities in various domains; particularly in trust, liquid software and federated learning.

IoT services defined as the transformation of IoT devices that should allow to address two major IoT challenges from the prism of service computing : communication with things and management of things [24]. At first glance, such a transformation allows to benefit from advances in service computing for a straight transposition into IoT service frameworks. However, the reality of IoT as a cosmopolitan world of heterogeneous things makes the shift challenging at several levels including IoT service annotation and discovery, IoT service running and synchronisation, ensuring privacy and trust, etc. Next, we highlight our vision about directions to tackle these challenges while connecting them to our previous contributions.

A user centric architecture with locally managed IoT service interactions. IoT services abstract IoT devices that are usually deployed for contextualized use. Their interactions, with each other as well as with end-user, should allow for a near real-time response delay. A user-centric architecture is more likely to efficiently handle such interactions. IoT services can take part of IoT Mashups to achieve more global common goal. Successful architecture could rely on the concept of *IoT Big Service* we proposed in [34] as a key point for IoT service composition. The proposed *multi-rooted tree architecture* allows for IoT services to be part of multiple IoT Big Services and any Big service can in turn take part to more general composition. Local and contextualized management of such services will allow to dynamically bring personalized solutions for end-user. Two technical considerations could be helpful to make such architecture feasible. First, the Web of Things (WoT) standardising initiative will make it easy to define LOD-based annotation and similarity calculation aver IoT services. This will allow to go further defining efficient discovery and recommendation techniques. Second, the use of REST/RESTful and HTTP for WoT device communication will allow extending Liquid Software principals for local management of IoT service interaction synchronisation.

A multi-layered architecture for multi-level privacy and trust management. IoT is one of the primary data generators [68], including personal data as well as sensible one. Besides, IoT device networks are by nature vulnerable [24]. This emphasises the need for efficient and reliable privacy and trust management mechanisms. In the previously introduced user centric architecture, this could be achieved through a multi-layered architecture. SOLID-based privacy management can be extended with local privacy control nodes for the data generated locally and then combined with higher level privacy control nodes acting as integrators. The role of privacy control nodes can be affected to intermediary big service within a global IoT Mashup.

Similarly, highest level of trust could be achieved combining hybrid Blockchain application [173] with on-chain/off-chain collaboration [64, 63]. To this end, a trustworthy interaction scenario could consist in a locally managed smart contract for near-immediate transaction state managing, then *relayer smart contracts* propagate transactional full logs or simply final state commitments to higher level Blockchains. This collaboration model extends the two-level ones in literature [173]. Achieving this is however challenging as it requires to think about light-weight Blockchains for the local and lower levels of the architecture.

9.3.2 Beyond conventional cloud computing: The rise of Distributed Computing Continuum (DCC)

Distributed Computing Continuum (DCC) [40] is a new emerging paradigm proposed to overcome both cloud and edge computing shortcomings by optimizing application-specific computational resources. Concurrently, the trend towards serverless computing in the cloud realm necessitates its adaptation at the edge to manage the heightened complexity, especially with AI/ML components. Edge computing presents unique challenges, categorized as core, specific, and derived characteristics, essential for developing a self-adaptive distributed system. Next, we go through main challenges that are still unsolved and their potential solutions in the realm of DCC.

First, the geographic distribution of available web resources may be problematic, indeed service and cloud computing models face issues related to data latency, especially when resources are scattered across vast geographic locations. By integrating resources from the edge to the cloud, DCC can optimize data processing closer to the data source, reducing latency and ensuring more efficient resource distribution. Second, scalability is a challenge in traditional cloud models, especially during peak demands, which can lead to service disruptions or degraded performance. DCC allows for dynamic scaling by leveraging resources across the computing spectrum (from edge devices to cloud centers) according to the demand, ensuring consistent performance. Last but not least, DCCs can provide a stan-

dardized interface ensuring that all devices can cohesively interact with each other despite the heterogeneity of their manufacturing standards, hence achieving high interoperability standards.

We believe that user-centric multilayered architecture for IoT services previously discussed is a step in the right direction to meet DCC requirement at the application level. Managing resources locally and at the Edge level can take advantage of rule-based approaches [132], recent standards for resource state management such as GraphQL [50], and on-the-fly reasoning techniques [115].

9.3.3 Application to Healthcare recommendation in SQVALD research project: A fusion of technologies perspective

Major recent research advances in service computing are being achieved through the convergence with other emerging techniques and methods such as AI, Blockchain, IoT, and DCC. Despite the independent evolution of these technologies, they are becoming more intertwined [68] for two main reasons: First, the novelty brought in one technology quickly finds echo in the other ones. This is for example the case of recent advances in AI deep learning used to bring solutions for service interactions as well as for IoT data "understanding". Second, the combined usage of these technologies allows to propose multi-facet solutions to reach new opportunities in several sensitive environments and domains such as Healthcare [68].

In healthcare, harnessing IoT/IoMT (Internet of Medical Things), AI, Edge Computing, and Blockchain effectively, to tackle challenges like patients privacy fragility, along with RGPD compliance, sensor data integration and monitoring, makes it necessary to propose approaches that leverage wisely all the previous technologies: IoT/IoMT will be in charge of collecting real-time data from various sources and devices, ingesting a rich and continuous flow of medical information within the healthcare framework. AI will take care of analyzing datasets that have been gathered previously to extract and interpret critical insights, thus making accurate predictions and proposing relevant recommendations to patient. In order to process data from IoT/IoMT devices, Edge Computing will run the processing close to the source, facilitating real-time analytics by AI, which is crucial for immediate response scenarios in healthcare, like critical care monitoring or danger recognition. Blockchain ensures the secure and immutable storage of the data generated by IoT/IoMT and analyzed by AI, providing a transparent and immutable ledger for transactions and interactions within the healthcare framework. It also ensures devices, users, and service providers authentication as well as their interactions within the framework [87].

In this context, we are running the SQVALD project with the main objective to assist patients during their recovery period after extensive therapeutic treatment for Cancer.

The idea is not to focus on monitoring various types of sensors and wearable devices continuously measuring and communicating values to integration level which alerts when something goes wrong. We rather aim at helping patient feel and live better in spite of the disease through: (i) the recommendation of adapted complementary activities outside hospitals, (ii) the logistic assistance to schedule and execute/practice these activities, and (iii) the evaluation of the impacts/benefits of the recommended activities on the quality of life of the patient. To this end, SQVALD framework has to successfully integrate the following functions. First, it should securely manage patient data, locally generated as well as accessed on the medical register, through privacy preserving and trustworthy ways. Second, it should be able to "understand" the patient need and capture/learn her/his preferences. Third, it should be able to recommend relevant personalized and context-aware activities. Fourth, it should define and/or choose the best adapted interaction services for a better quality of experience for the patient, ensuring her/his active participation. Fifth, it should include quality of life evaluation metrics and help patient following the right direction.

In order to meet these challenging requirements, it will be necessary to follow a fusion of technologies perspective: IoT, AI, DCC/Edge-Cloud, and Blockchain [68], within a user-centric framework. The already achieved contributions as well as research directions discussed in this dissertation will be considered together towards a successful outcome for the SQVALD project.

Bibliography

[1] H. Abbasimehr and R. Paki. Improving time series forecasting using LSTM and attention models. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–19, 2022.

[2] A. Abid. *Improvement of web service composition using semantic similarities and formal concept analysis*. PhD thesis, University of Tours (France) and University of Sfax (Tunisia), 2017.

[3] A. Abid, N. Messai, M. Rouached, M. Abid, and T. Devogele. Semantic similarity based web services composition framework. In A. Seffah, B. Penzenstadler, C. Alves, and X. Peng, editors, *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, pages 1319–1325. ACM, 2017.

[4] A. Abid, N. Messai, M. Rouached, T. Devogele, and M. Abid. IDECSE: A semantic integrated development environment for composite services engineering. In S. Nurcan, E. Pimenidis, O. Pastor, and Y. Vassiliou, editors, *Joint Proceedings of the CAiSE 2014 Forum and CAiSE 2014 Doctoral Consortium co-located with the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Thessaloniki, Greece*, volume 1164 of *CEUR Workshop Proceedings*, pages 105–112. CEUR-WS.org, 2014.

[5] A. Abid, N. Messai, M. Rouached, T. Devogele, and M. Abid. A semantic-aware framework for composite services engineering based on semantic similarity and concept lattices. In S. Nurcan and E. Pimenidis, editors, *Information Systems Engineering in Complex Environments - CAiSE Forum 2014, Thessaloniki, Greece, Selected Extended Papers*, volume 204 of *Lecture Notes in Business Information Processing*, pages 148–164. Springer, 2014.

[6] A. Abid, N. Messai, M. Rouached, T. Devogele, and M. Abid. A semantic similarity measure for conceptual web services classification. In S. Reddy, editor, *24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative*

Enterprises, WETICE 2015, Larnaca, Cyprus, June 15-17, 2015, pages 128–133. IEEE Computer Society, 2015.

[7] A. Abid, M. Rouached, and N. Messai. Semantic web service composition using semantic similarity measures and formal concept analysis. *Multim. Tools Appl.*, 79(9-10):6569–6597, 2020.

[8] A. Abid, M. Rouached, N. Messai, M. Abid, and T. Devogele. A semantic matching engine for web service composition. *Int. J. Bus. Inf. Syst.*, 30(1):92–108, 2019.

[9] S. Aghaee and C. Pautasso. End-user development of mashups with naturalmash. *Journal of Visual Languages & Computing*, 25(4):414–432, 2014.

[10] E. Al-Masri and Q. H. Mahmoud. Mobieureka: an approach for enhancing the discovery of mobile web services. *Pers. Ubiquitous Comput.*, 14(7):609–620, 2010.

[11] S. D. Alfarhood. *Exploiting Semantic Distance in Linked Open Data for Recommendation*. PhD thesis, University of Arkansas, Fayetteville, 2017.

[12] C. Ardito, P. Buono, G. Desolda, and M. Matera. From smart objects to smart experiences: An end-user development approach. *International Journal of Human-Computer Studies*, 114:51–68, 2018.

[13] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha, and C. Tibermacine. Selection of composable web services driven by user requirements. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 395–402. IEEE, 2011.

[14] Z. Azmeh, F. Hamoui, M. Huchard, N. Messai, C. Tibermacine, C. Urtado, and S. Vauttier. Backing composite web services using formal concept analysis. In *Formal Concept Analysis - 9th International Conference, ICFCA 2011, Nicosia, Cyprus, May 2-6, 2011. Proceedings*, pages 26–41, 2011.

[15] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[16] M. Barhamgi and D. Benslimane. Composing data-providing web services. In *VLDB PhD Workshop*, 2009.

[17] G. Baryannis, O. Danylevych, D. Karastoyanova, K. Kritikos, P. Leitner, F. Rosenberg, and B. Wetzstein. Service composition. In *Service Research Challenges and Solutions for the Future Internet - S-Cube - Towards Engineering, Managing and Adapting Service-Based Systems*, pages 55–84, 2010.

[18] A. Belkhirat, A. Belkhir, and A. Bouras. A new similarity measure for the profiles management. In D. Al-Dabass, A. Orsoni, R. J. Cant, and A. Abraham, editors, *Proceedings of the 13th UKSim-AMSS International Conference on Computer Modelling and Simulation, Cambridge University, Emmanuel College, Cambridge, UK, 30 March - 1 April 2011*, pages 255–259. IEEE Computer Society, 2011.

[19] T. Berners-Lee. Linked data -. *W3C Design Issues*, 2006.

[20] T. Berners-Lee, J. A. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[21] P. Bhargava, J. Lampton, and A. Agrawala. Bootstrapped discovery and ranking of relevant services and information in context-aware systems. *EAI Endorsed Transactions on Context-aware Systems and Applications*, 2, 08 2015.

[22] S. Bhiri, W. Gaaloul, C. Godart, O. Perrin, M. Zaremba, and W. Derguech. Ensuring customised transactional reliability of composite services. *J. Database Manag.*, 22(2):64–92, 2011.

[23] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009.

[24] A. Bouguettaya, Q. Z. Sheng, B. Benatallah, A. G. Neiat, S. Mistry, A. Ghose, S. Nepal, and L. Yao. An internet of things service roadmap. *Commun. ACM*, 64(9):86–95, 2021.

[25] M. Boulakbech. *Mashup de services par configuration : application au domaine touristique*. PhD thesis, Université de Tours, 2020.

[26] M. Boulakbech, N. Cheniki, N. Messai, Y. Sam, and T. Devogele. Linked data graphs for semantic data integration in the CART system. In C. Pautasso, F. Sánchez-Figueroa, K. Systä, and J. M. M. Rodríguez, editors, *Current Trends in Web Engineering - ICWE 2018 International Workshops, MATWEP, EnWot, KD-WEB, WEOD, TourismKG, Cáceres, Spain, June 5, 2018, Revised Selected Papers*, volume 11153 of *Lecture Notes in Computer Science*, pages 221–226. Springer, 2018.

[27] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. A smart mobiweb mashup trip planner tool. In A. Seffah, B. Penzenstadler, C. Alves, and X. Peng, editors, *Proceedings of the Symposium on Applied Computing, SAC 2017, Marrakech, Morocco, April 3-7, 2017*, pages 686–688. ACM, 2017.

[28] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. Visual configuration for restful mobile web mashups. In I. Altintas and S. Chen, editors, *2017 IEEE International*

Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017, pages 870–873. IEEE, 2017.

[29] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. Configuring restful web services for personalized trip planning. In *27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2018, Paris, France, June 27-29, 2018*, pages 213–216. IEEE Computer Society, 2018.

[30] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. Attentive knowledge-aware path network for explainable travel mashup. In R. Chbeir, Z. H. Huang, F. Silvestri, Y. Manolopoulos, and Y. Zhang, editors, *Web Information Systems Engineering - WISE 2022 - 23rd International Conference, Biarritz, France, November 1-3, 2022, Proceedings*, volume 13724 of *Lecture Notes in Computer Science*, pages 519–533. Springer, 2022.

[31] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. Configuration approach for personalized travel mashup. *Concurr. Comput. Pract. Exp.*, 35(11), 2023.

[32] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele. Deep learning model for personalized web service recommendations using attention mechanism. In *21st International Conference on Service-Oriented Computing - ICSOC 2023, Rome, Italy, November 28th – December 1st, 2023, Proceedings*, Lecture Notes in Computer Science, 2023.

[33] M. Boulakbech, N. Messai, Y. Sam, T. Devogele, and L. Étienne. Smartloire: A web mashup based tool for personalized touristic plans construction. In S. Reddy and W. Gaaloul, editors, *25th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2016, Paris, France, June 13-15, 2016*, pages 259–260. IEEE Computer Society, 2016.

[34] M. Boulakbech, N. Messai, Y. Sam, T. Devogele, and M. Hammoudeh. Iot mashups: From iot big data to iot big service. In M. Hammoudeh and R. M. Newman, editors, *Proceedings of the International Conference on Future Networks and Distributed Systems, ICFNDS 2017, Cambridge, United Kingdom, July 19-20, 2017*, page 20. ACM, 2017.

[35] A. Brahem, T. Henry, S. Bhiri, T. Devogele, N. Laga, N. Messai, Y. Sam, W. Gaaloul, and B. Benatallah. A trustworthy decentralized change propagation mechanism for declarative choreographies. In C. D. Ciccio, R. M. Dijkman, A. del-Río-Ortega, and S. Rinderle-Ma, editors, *Business Process Management - 20th International Conference, BPM 2022, Münster, Germany, September 11-16, 2022, Proceedings*, volume 13420 of *Lecture Notes in Computer Science*, pages 418–435. Springer, 2022.

[36] A. Brahem, N. Messai, Y. Sam, S. Bhiri, T. Devogele, and W. Gaaloul. Blockchain's fame reaches the execution of personalized touristic itineraries. In S. Reddy, editor, *28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2019, Naples, Italy, June 12-14, 2019*, pages 186–191. IEEE, 2019.

[37] A. Brahem, N. Messai, Y. Sam, S. Bhiri, T. Devogele, and W. Gaaloul. Running transactional business processes with blockchain's smart contracts. In *2020 IEEE International Conference on Web Services, ICWS 2020, Beijing, China, October 19-23, 2020*, pages 89–93. IEEE, 2020.

[38] A. E. Cano, A.-S. Dadzie, and F. Ciravegna. Travel mashups. In *Semantic Mashups*, pages 321–347. Springer, 2013.

[39] C. Cappiello, F. Daniel, M. Matera, and C. Pautasso. Information quality in mashups. *IEEE Internet Comput.*, 14(4):14–22, 2010.

[40] V. Casamayor-Pujol, P. K. Donta, A. Morichetta, I. Murturi, and S. Dustdar. Edge intelligence - research opportunities for distributed computing continuum systems. *IEEE Internet Comput.*, 27(4):53–74, 2023.

[41] F. Casati, F. Daniel, A. D. Angeli, M. Imran, S. Soi, C. R. Wilkinson, and M. Marchese. Developing mashup tools for end-users: On the importance of the application domain. *Int. J. Next Gener. Comput.*, 3(2), 2012.

[42] F. Casati, F. Daniel, A. De Angeli, M. Imran, S. Soi, C. R. Wilkinson, and M. Marchese. Developing mashup tools for end-users: on the importance of the application domain. *International Journal of Next-Generation Computing*, 3(2), 2012.

[43] S. A. Chellouche, D. Négru, J. Arnaud, and J. M. Batalla. Context-aware multimedia services provisioning in future internet using ontology and rules. In *2014 International Conference and Workshop on the Network of the Future, NOF 2014, Paris, France, December 3-5, 2014*, pages 1–5. IEEE, 2014.

[44] F. Chen, C. Lu, H. Wu, and M. Li. A semantic similarity measure integrating multiple conceptual relationships for web service discovery. *Expert Systems with Applications*, 67:19–31, 2017.

[45] N. Cheniki. *Découverte des Web Services Mobiles*. PhD thesis, Université des Sciences et de la Technologie Houari Boumediène, Algérie, 2017.

[46] N. Cheniki, A. Belkhir, Y. Sam, and N. Messai. LODS: A linked open data based similarity measure. In S. Reddy and W. Gaaloul, editors, *25th IEEE International*

Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2016, Paris, France, June 13-15, 2016, pages 229–234. IEEE Computer Society, 2016.

[47] N. Cheniki, M. Boulakbech, H. Labbaci, Y. Sam, N. Messai, and T. Devogele. A linked open data based approach for trip recommendation. In S. Reddy, editor, *28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2019, Naples, Italy, June 12-14, 2019*, pages 192–195. IEEE, 2019.

[48] N. Cheniki, Y. Sam, N. Messai, and A. Belkhir. Context-aware and linked open data based service discovery. In T. Mikkonen, R. Klamma, and J. Hernández, editors, *Web Engineering - 18th International Conference, ICWE 2018, Cáceres, Spain, June 5-8, 2018, Proceedings*, volume 10845 of *Lecture Notes in Computer Science*, pages 448–462. Springer, 2018.

[49] C. P. da Silva. *The CUBE: A User-Centric system-model architecture for Web service migration through multiple devices.* PhD thesis, University of Tours, 2019.

[50] C. P. da Silva and N. Messai. Beyond traditional web technologies for locally web-services migration. In H. Fill, F. J. D. Mayo, M. van Sinderen, and L. A. Maciaszek, editors, *Proceedings of the 18th International Conference on Software Technologies, ICSOFT 2023, Rome, Italy, July 10-12, 2023*, pages 620–628. SCITEPRESS, 2023.

[51] C. P. da Silva, N. Messai, Y. Sam, and T. Devogele. CUBE system: A REST and restful based platform for liquid software approaches. In T. A. Majchrzak, P. Traverso, K. Krempels, and V. Monfort, editors, *Web Information Systems and Technologies - 13th International Conference, WEBIST 2017, Porto, Portugal, April 25-27, 2017, Revised Selected Papers*, volume 322 of *Lecture Notes in Business Information Processing*, pages 115–131. Springer, 2017.

[52] C. P. da Silva, N. Messai, Y. Sam, and T. Devogele. Diamond - A cube model proposal based on a centric architecture approach to enhance liquid software model approaches. In T. A. Majchrzak, P. Traverso, K. Krempels, and V. Monfort, editors, *Proceedings of the 13th International Conference on Web Information Systems and Technologies, WEBIST 2017, Porto, Portugal, April 25-27, 2017*, pages 382–387. SciTePress, 2017.

[53] C. P. da Silva, N. Messai, Y. Sam, and T. Devogele. Liquid mail - A client mail based on CUBE model. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*, pages 1539–1540. IEEE Computer Society, 2018.

[54] C. P. da Silva, N. Messai, Y. Sam, and T. Devogele. User-side service synchronization in multiple devices environment. In M. Bieliková, T. Mikkonen, and C. Pautasso, editors, *Web Engineering - 20th International Conference, ICWE 2020, Helsinki, Finland, June 9-12, 2020, Proceedings*, volume 12128 of *Lecture Notes in Computer Science*, pages 451–466. Springer, 2020.

[55] E. G. Da Silva, L. F. Pires, and M. Van Sinderen. Towards runtime discovery, selection and composition of semantic services. *Computer communications*, 34(2):159–168, 2011.

[56] J. Danado and F. Paternò. Puzzle: A mobile application development environment using a jigsaw metaphor. *Journal of Visual Languages & Computing*, 25(4):297–315, 2014.

[57] F. Daniel, M. Matera, F. Daniel, and M. Matera. *Mashups*. Springer, 2014.

[58] S. Debois, T. T. Hildebrandt, and T. Slaats. Replication, refinement & reachability: complexity in dynamic condition-response graphs. *Acta Informatica*, 55(6):489–520, 2018.

[59] S. Dustdar and W. Schreiner. A survey on web services composition. *Int. J. Web Grid Serv.*, 1(1):1–30, August 2005.

[60] J. El Hadad, M. Manouvrier, and M. Rukoz. Tqos: Transactional and qos-aware selection algorithm for automatic web service composition. *IEEE Transactions on Services Computing*, 3(1):73–85, 2010.

[61] K. Elgazzar, A. E. Hassan, and P. Martin. Clustering WSDL documents to bootstrap the discovery of web services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 147–154. IEEE, 2010.

[62] K. Elgazzar, H. S. Hassanein, and P. Martin. Daas: Cloud-based mobile web service discovery. *Pervasive Mob. Comput.*, 13:67–84, 2014.

[63] G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov. Smart contract invocation protocol (SCIP): A protocol for the uniform integration of heterogeneous blockchain smart contracts. In S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, editors, *Advanced Information Systems Engineering - 32nd International Conference, CAiSE 2020, Grenoble, France, June 8-12, 2020, Proceedings*, volume 12127 of *Lecture Notes in Computer Science*, pages 134–149. Springer, 2020.

[64] G. Falazi, U. Breitenbücher, M. Falkenthal, L. Harzenetter, F. Leymann, and V. Yussupov. Blockchain-based collaborative development of application deployment models. In H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I*, volume 11229 of *Lecture Notes in Computer Science*, pages 40–60. Springer, 2018.

[65] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert. Dealing with change in process choreographies: Design and implementation of propagation algorithms. *Information Systems*, 49:1–24, 2015.

[66] G. Fenza, D. Furno, and V. Loia. Hybrid approach for context-aware service discovery in healthcare domain. *J. Comput. Syst. Sci.*, 78(4):1232–1247, 2012.

[67] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2(2):115–150, 2002.

[68] F. Firouzi, S. Jiang, K. Chakrabarty, B. J. Farahani, M. Daneshmand, J. Song, and K. Mankodiya. Fusion of iot, ai, edge-fog-cloud, and blockchain: Challenges, solutions, and a case study in healthcare and medicine. *IEEE Internet Things J.*, 10(5):3686–3705, 2023.

[69] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau. An empirical study on web service evolution. In *2011 IEEE International Conference on Web Services*, pages 49–56. IEEE, 2011.

[70] A. Gallidabino, C. Pautasso, V. Ilvonen, T. Mikkonen, K. Systä, J.-P. Voutilainen, and A. Taivalsaari. On the architecture of liquid software: Technology alternatives and design space. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 122–127, 2016.

[71] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer Berlin Heidelberg, 1999.

[72] Z. Gao, Y. Fan, C. Wu, W. Tan, and J. Zhang. Service recommendation from the evolution of composition patterns. In *2017 IEEE International Conference on Services Computing (SCC)*, pages 108–115, 2017.

[73] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber. Optimized execution of business processes on blockchain. In *Business Process Management: 15th International Conference, BPM 2017, Barcelona, Spain, September 10–15, 2017, Proceedings 15*, pages 130–146. Springer, 2017.

[74] D. Georgakopoulos and M. P. Papazoglou. *Service-Oriented Computing*. The MIT Press, 11 2008.

[75] M. Ghallab, A. Howe, D. McDermott, A. Ram, M. Veloso, D. Weld, and W. David. Pddl - the planning domain definition language (version 1.2). Technical report, Yale Center for Computational Vision and Control, 1998.

[76] G. Ghiani, F. Paternò, L. D. Spano, and G. Pintori. An environment for end-user development of web mashups. *International Journal of Human-Computer Studies*, 87:38–64, 2016.

[77] L. Hamel, M. Graiet, M. Kmimech, M. T. Bhiri, and W. Gaaloul. Verifying composite service transactional behavior with EVENT-B. In I. Crnkovic, V. Gruhn, and M. Book, editors, *Software Architecture - 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings*, volume 6903 of *Lecture Notes in Computer Science*, pages 67–74. Springer, 2011.

[78] P. Hamilton and D. J. Wigdor. Conductor: Enabling and understanding cross-device interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'14, page 2773–2782, New York, NY, USA, 2014. Association for Computing Machinery.

[79] S. Harispe, S. Ranwez, S. Janaqi, and J. Montmain. Semantic measures based on RDF projections: Application to content-based recommendation systems. In R. Meersman, H. Panetto, T. S. Dillon, J. Eder, Z. Bellahsene, N. Ritter, P. D. Leenheer, and D. Dou, editors, *On the Move to Meaningful Internet Systems: OTM 2013 Conferences - Confederated International Conferences: CoopIS, DOA-Trusted Cloud, and ODBASE 2013, Graz, Austria, September 9-13, 2013. Proceedings*, volume 8185 of *Lecture Notes in Computer Science*, pages 606–615. Springer, 2013.

[80] J. Hartman, U. Manber, L. Peterson, and T. Proebsting. Liquid software: A new paradigm for networked systems. Technical report, USA, 1996.

[81] O. Hatzi, M. Nikolaidou, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas. Semantically aware web service composition through AI planning. *IJAIT*, 2015.

[82] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas. The PORSCE II framework: Using AI planning for automated semantic Web service composition. *The Knowledge Engineering Review*, 28(02):137–156, 2013.

[83] T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[84] T. Henry, A. Brahem, N. Laga, J. Hatin, W. Gaaloul, and B. Benatallah. Trustworthy cross-organizational collaborations with hybrid on/off-chain declarative choreographies. In *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings*, page 81–96. Springer-Verlag, 2021.

[85] T. T. Hildebrandt, T. Slaats, H. A. López, S. Debois, and M. Carbone. Declarative choreographies and liveness. In *Formal Techniques for Distributed Objects, Components, and Systems: 39th IFIP WG 6.1 International Conference, FORTE 2019*, page 129–147. Springer-Verlag, 2019.

[86] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[87] X. Hu, E. C. H. Ngai, G. Castellano, B. Hu, J. J. P. C. Rodrigues, and J. Song. Special issue on "toward intelligent internet of medical things and its covid-19 applications and beyond". *IEEE Internet of Things Journal*, 8(21):15649–15651, 2021.

[88] R. Hull, V. S. Batra, Y.-M. Chen, A. Deutsch, F. F. T. Heath III, and V. Vianu. Towards a shared ledger business collaboration language based on data-aware processes. In Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, editors, *Service-Oriented Computing*, pages 18–36, Cham, 2016. Springer International Publishing.

[89] C. Indiono and S. Rinderle-Ma. Dynamic change propagation for process choreography instances. In *OTM Conferences*, pages 334–352. Springer, 2017.

[90] K. Khalil, K. Elgazzar, M. Seliem, and M. Bayoumi. Resource discovery techniques in the internet of things: A review. *Internet of Things*, 12:100293, 2020.

[91] F. Komeiha, N. Cheniki, Y. Sam, A. Jaber, N. Messai, and T. Devogele. LDS: java library for linked open data based similarity measures. In J. He, H. Purohit, G. Huang, X. Gao, and K. Deng, editors, *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, WI/IAT 2020, Melbourne, Australia, December 14-17, 2020*, pages 476–481. IEEE, 2020.

[92] F. Komeiha, N. Cheniki, Y. Sam, A. Jaber, N. Messai, and T. Devogele. Towards a privacy conserved and linked open data based device recommendation in iot. In H. Hacid, F. Outay, H. Paik, A. Alloum, M. Petrocchi, M. R. Bouadjenek, A. Beheshti, X. Liu, and A. Maaradji, editors, *Service-Oriented Computing - ICSOC 2020 Workshops - AIOps, CFTIC, STRAPS, AI-PA, AI-IOTS, and Satellite Events, Dubai, United Arab Emirates, December 14-17, 2020, Proceedings*, volume 12632 of *Lecture Notes in Computer Science*, pages 32–39. Springer, 2020.

[93] J. Kwon, H. Kim, D. Lee, and S. Lee. Redundant-free web services composition based on a two-phase algorithm. In *2008 IEEE International Conference on Web Services (ICWS 2008), September 23-26, 2008, Beijing, China*, pages 361–368. IEEE Computer Society, 2008.

[94] H. Labbaci. *A Social Approach for Web Service Discovery and Community Detection*. PhD thesis, University of Science and Technology Houari Boumediène, Algiers, 2020.

[95] H. Labbaci, N. Cheniki, Y. Sam, N. Messai, B. Medjahed, and Y. Aklouf. A linked open data approach for web service evolution. In H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, and R. Meersman, editors, *On the Move to Meaningful Internet Systems: OTM 2019 Conferences - Confederated International Conferences: CoopIS, ODBASE, C&TC 2019, Rhodes, Greece, October 21-25, 2019, Proceedings*, volume 11877 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2019.

[96] H. Labbaci, B. Medjahed, and Y. Aklouf. Learning interactions from web service logs. In *International Conference on Database and Expert Systems Applications*, pages 275–289. Springer, 2017.

[97] H. Labbaci, B. Medjahed, Y. Aklouf, and Z. Malik. Follow the leader: A social network approach for service communities. In Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, editors, *Service-Oriented Computing - 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings*, volume 9936 of *Lecture Notes in Computer Science*, pages 705–712. Springer, 2016.

[98] H. Labbaci, B. Medjahed, F. Binzagr, and Y. Aklouf. A deep learning approach for web service interactions. In A. P. Sheth, A. Ngonga, Y. Wang, E. Chang, D. Slezak, B. Franczyk, R. Alt, X. Tao, and R. Unland, editors, *Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, August 23-26, 2017*, pages 848–854. ACM, 2017.

[99] H. Labbaci, B. Medjahed, F. Binzagr, and Y. Aklouf. A deep learning approach for web service interactions. In A. P. Sheth, A. Ngonga, Y. Wang, E. Chang, D. Slezak, B. Franczyk, R. Alt, X. Tao, and R. Unland, editors, *Proceedings of the International Conference on Web Intelligence, Leipzig, Germany, August 23-26, 2017*, pages 848–854. ACM, 2017.

[100] F. Lécué and A. Léger. A formal model for semantic Web service composition. In *The Semantic Web-ISWC 2006*, pages 385–398. Springer, 2006.

[101] F. Lécué, E. Silva, and L. F. Pires. A framework for dynamic web services composition. In *Emerging Web Services Technology*. Springer, 2008.

[102] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.

[103] A. L. Lemos, F. Daniel, and B. Benatallah. Web service composition: A survey of techniques and tools. *ACM Comput. Surv.*, 48(3):33, 2016.

[104] X. Li, X. Zhang, P. Wang, and Z. Cao. Web services recommendation based on metapath-guided graph attention network. *The Journal of Supercomputing*, 78(10):12621–12647, 2022.

[105] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu. Measuring similarity of web services based on WSDL. In *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010.

[106] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber. Controlled flexibility in blockchain-based collaborative business processes. *Information Systems*, 104:101622, 2022.

[107] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev. Caterpillar: A business process execution engine on the ethereum blockchain. *Software: Practice and Experience*, 49(7):1162–1193, 2019.

[108] A. Louati, J. El Haddad, and S. Pinson. Towards agent-based and trust-oriented service discovery approach in social networks. In *TRUST@ AAMAS*, pages 78–89, 2014.

[109] Y. Ma, S.-H. Jang, and J. Lee. Ontology-based resource management for cloud computing. In *Intelligent Information and Database Systems*, pages 343–352, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[110] Z. Maamar, N. Faci, L. Krug Wives, H. Yahyaoui, and H. Hacid. Towards a method for engineering social web services. In J. Ralyté, I. Mirbel, and R. Deneckère, editors, *Engineering Methods in the Service-Oriented Context*, pages 153–167, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[111] Z. Maamar, N. Faci, L. Wives, Y. Badr, P. Santos, and J. Palazzo M. Oliveira. Using social networks for web services discovery. *IEEE Internet Computing*, 15(4):48–54, 2011.

[112] E. Mansour, A. V. Sambra, S. Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, and T. Berners-Lee. A demonstration of the solid platform for social web applications. In J. Bourdeau, J. Hendler, R. Nkambou, I. Horrocks, and B. Y. Zhao, editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11-15, 2016, Companion Volume*, pages 223–226. ACM, 2016.

[113] A. Mattioli and F. Paternò. A visual environment for end-user creation of iot customization rules with recommendation support. In *Proceedings of the International Conference on Advanced Visual Interfaces*, pages 1–5, 2020.

[114] S. McIlraith, T. Son, and H. Zeng. Semantic web services. *Intelligent Systems, IEEE*, 16:46 – 53, 04 2001.

[115] W. Mebrek and A. Bouzeghoub. A multi-agent based framework for RDF stream processing. In L. Barolli, F. Hussain, and T. Enokido, editors, *Advanced Information Networking and Applications - Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), Sydney, NSW, Australia, 13-15 April 2022, Volume 1*, volume 449 of *Lecture Notes in Networks and Systems*, pages 516–528. Springer, 2022.

[116] B. Medjahed, A. Bouguettaya, and B. Benatallah. Introduction to special issue on semantic web services. *ACM Trans. Internet Techn.*, 8(1):1, 2007.

[117] J. Mendling, I. Weber, W. M. P. van der Aalst, J. vom Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and L. Zhu. Blockchains for business process management - challenges and opportunities. *ACM Trans. Manag. Inf. Syst.*, 9(1):4:1–4:16, 2018.

[118] N. Messai, M. Devignes, A. Napoli, and M. Smaïl-Tabbone. Many-valued concept lattices for conceptual clustering and information retrieval. In M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, editors, *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 127–131. IOS Press, 2008.

[119] N. Messai, M. Devignes, A. Napoli, and M. Smaïl-Tabbone. Using domain knowledge to guide lattice-based complex data exploration. In H. Coelho, R. Studer, and M. J.

Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 847–852. IOS Press, 2010.

[120] R. Meymandpour and J. G. Davis. Linked data informativeness. In Y. Ishikawa, J. Li, W. Wang, R. Zhang, and W. Zhang, editors, *Web Technologies and Applications - 15th Asia-Pacific Web Conference, APWeb 2013, Sydney, Australia, April 4-6, 2013. Proceedings*, volume 7808 of *Lecture Notes in Computer Science*, pages 629–637. Springer, 2013.

[121] R. Meymandpour and J. G. Davis. Enhancing recommender systems using linked open data-based semantic analysis of items. In J. G. Davis and A. Bozzon, editors, *3rd Australasian Web Conference, AWC 2015, Sydney, Australia, January 2015*, volume 166 of *CRPIT*, pages 11–17. Australian Computer Society, 2015.

[122] T. Mikkonen, K. Systä, and C. Pautasso. Towards liquid web applications. In P. Cimiano, F. Frasincar, G.-J. Houben, and D. Schwabe, editors, *Engineering the Web in the Big Data Era*, pages 134–143, Cham, 2015. Springer International Publishing.

[123] G. A. Miller and W. G. Charles. Contextual correlates of semantic similarity. *Language and Cognitive Processes*, 6(1):1–28, 1991.

[124] B. Montz, P. Bridges, P. A. Bigot, R. Piltz, J. J. Hartman, O. Spatscheck, L. L. Peterson, A. Bavier, and T. A. Proebsting. Joust: A platform for liquid software. *Computer*, 32(04):50–56, 1999.

[125] F. Moscato, R. Aversa, B. Di Martino, T.-F. Fortiş, and V. Munteanu. An analysis of mosaic ontology for cloud resources annotation. In *Proceedings of the Federated Conference onComputer Science and Information Systems*, 2011.

[126] M. L. Mouhoub, D. Grigori, and M. Manouvrier. A framework for searching semantic data and services with SPARQL. In *International Conference on Service-Oriented Computing, ICSOC 2014, Paris, France*, pages 123–138, 2014.

[127] R. R. Mukkamala. *A formal model for declarative workflows: dynamic condition response graphs*. PhD thesis, IT University of Copenhagen, 2012.

[128] R. R. Mukkamala, T. Hildebrandt, and T. Slaats. Towards trustworthy adaptive case management with dynamic condition response graphs. In *2013 17th IEEE International Enterprise Distributed Object Computing Conference, EDOC*, pages 127–136, 2013.

[129] H. Naim, M. Aznag, M. Quafafou, and N. Durand. Probabilistic approach for diversifying web services discovery and composition. In S. Reiff-Marganiec, editor, *IEEE International Conference on Web Services, ICWS 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, pages 73–80. IEEE Computer Society, 2016.

[130] S. Najar, M. Kirsch-Pinheiro, C. Souveyet, and L. A. Steffenel. Service discovery mechanism for an intentional pervasive information system. In C. A. Goble, P. P. Chen, and J. Zhang, editors, *2012 IEEE 19th International Conference on Web Services, Honolulu, HI, USA, June 24-29, 2012*, pages 520–527. IEEE Computer Society, 2012.

[131] I. A. Narayana and T.Roopa. Extending android application programming framework for seamless cloud integration. In *Proceedings of the 2012 IEEE First International Conference on Mobile Services*, pages 96–104. IEEE Computer Society, 2012.

[132] D. Nouicer, N. Messai, Y. Sam, and I. C. Msadaa. Semantic rule-based device recommendation for service migration in multiple device contexts. In *30th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2021, Bayonne, France, October 27-29, 2021*, pages 171–176. IEEE, 2021.

[133] S. Ovadia. Automate the internet with "if this then that"(ifttt). *Behavioral & social sciences librarian*, 33(4):208–211, 2014.

[134] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.

[135] A. Passant. Measuring semantic distance on linking data and using it for resources recommendations. In *AAAI spring symposium: linked data meets artificial intelligence*, volume 77, page 123, 2010.

[136] C. Pautasso. *RESTful Web Services: Principles, Patterns, Emerging Technologies*, pages 31–51. Springer New York, New York, NY, 2014.

[137] G. Piao, S. showkat Ara, and J. G. Breslin. Computing the semantic similarity of resources in DBpedia for recommendation purposes. In *Semantic Technology*, pages 185–200. Springer International Publishing, 2016.

[138] G. Pirrò. Reword: Semantic relatedness in the web of data. In J. Hoffmann and B. Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, pages 129–135. AAAI Press, 2012.

[139] P. Plebani and B. Pernici. URBE: Web service retrieval based on similarity evaluation. *Knowledge and Data Engineering, IEEE Transactions on*, 2009.

[140] C. Prybila, S. Schulte, C. Hochreiner, and I. Weber. Runtime verification for business processes utilizing the Bitcoin blockchain. *Future generation computer systems*, 107:816–831, 2020.

[141] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, Inc., May 2007.

[142] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes. An integrated semantic web service discovery and composition framework. *IEEE Transactions on Services Computing*, 9(4):537–550, 2016.

[143] H. Rubenstein and J. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8:627–633, 10 1965.

[144] M. Ruta, F. Scioscia, E. Di Sciascio, and G. Piscitelli. Semantic matchmaking for location-aware ubiquitous resource discovery. *International Journal On Advances in Intelligent Systems*, 4(3/4):113–127, 2012.

[145] D. Sánchez, M. Batet, D. Isern, and A. Valls. Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*, 39(9):7718–7728, 2012.

[146] V. Saquicela, L. M. Vilches-Blázquez, and O. Corcho. Semantic annotation of restful services using external resources. volume 6385, pages 266–276, 07 2010.

[147] M. Schrepp, A. Hinderks, and J. Thomaschewski. Construction of a benchmark for the user experience questionnaire (ueq). *IJIMAI*, 4(4):40–44, 2017.

[148] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and isolation for transactional processes. *ACM Transactions on Database Systems (TODS)*, 27(1):63–116, 2002.

[149] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu. Web services composition: A decade's overview. *Inf. Sci.*, 280:218–238, 2014.

[150] M. Shi, J. Liu, et al. Functional and contextual attention-based LSTM for service recommendation in mashup creation. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1077–1090, 2018.

[151] B. Stein. *Functional models in configuration systems*. PhD thesis, University of Paderborn, Germany, 1995.

[152] V. Suraci, S. Mignanti, and A. Aiuto. Context-aware semantic service discovery. In *2007 16th IST Mobile and Wireless Communications Summit*, pages 1 – 5, 08 2007.

[153] A. ter Hofstede, W. M. van der Aalst, A. H. ter Hofstede, and M. Weske. Business process management: A survey. In *Business Process Management: International Conference, BPM 2003 Eindhoven, The Netherlands, June 26–27, 2003 Proceedings 1*, pages 1–12. Springer, 2003.

[154] O. Tibermacine, C. Tibermacine, and F. Cherif. Wssim: a tool for the measurement of web service interface similarity. In *Proceedings of the french-speaking Conference on Software Architectures (CAL)*, 2013.

[155] A. Toninelli, A. Corradi, and R. Montanari. Semantic-based discovery to support mobile context-aware service access. *Comput. Commun.*, 31(5):935–949, 2008.

[156] H. T. Tran, A. Jahl, K. Geihs, R. Kuppili, X. T. Nguyen, and T. T. B. Huynh. Decom: A framework to support evolution of iot services. In *Proceedings of the Ninth International Symposium on Information and Communication Technology*, pages 389–396. ACM, 2018.

[157] A. Tversky. Features of similarity. *Psychological review*, 84(4):327, 1977.

[158] S. Underwood. Blockchain beyond Bitcoin. *Communications of the ACM*, 59(11):15–17, 2016.

[159] M. Unger, A. Tuzhilin, and A. Livne. Context-aware recommendations based on deep learning frameworks. *ACM Transactions on Management Information Systems (TMIS)*, 11(2):1–15, 2020.

[160] P. Valderas, V. Torres, I. Mansanet, and V. Pelechano. A mobile-based solution for supporting end-users in the composition of services. *Multimedia Tools and Applications*, 76(15):16315–16345, 2017.

[161] P. Valderas, V. Torres, and V. Pelechano. A social network for supporting end users in the composition of services: definition and proof of concept. *Computing*, pages 1–32, 2020.

[162] P. Valderas, V. Torres, and V. Pelochano. Supporting a hybrid composition of microservices. the eucaliptool platform. *Journal of Software Engineering Research and Development*, 8:1–1, 2020.

[163] W. M. P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H. M. W. Verbeek. Choreography conformance checking: An approach based on BPEL and petri nets.

In F. Leymann, W. Reisig, S. R. Thatte, and W. M. P. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures, 16.07. - 21.07.2006*, volume 06291 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[164] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske, editors, *Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003.

[165] M. P. Villalon, M. C. Suárez-Figueroa, R. García-Castro, and A. Gómez-Pérez. A context ontology for mobile environments. In *Proceedings of Workshop on Context, Information and Ontologies - CIAO 2010 Co-located with EKAW 2010*, volume 626, Alemania, Octubre 2010. CEUR-WS.

[166] G. Vladislava. Semantic description of web services and possibilities of bpel4ws. *International Journal Information Theories and Applications*, 13(2):183–187, Nov. 2006.

[167] S. Wang, W. A. Higashino, M. Hayes, and M. A. Capretz. Service evolution patterns. In *2014 IEEE International Conference on Web Services*, pages 201–208. IEEE, 2014.

[168] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 950–958, 2019.

[169] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. Untrusted business process monitoring and execution using blockchain. In M. La Rosa, P. Loos, and O. Pastor, editors, *Business Process Management*, pages 329–347, Cham, 2016. Springer International Publishing.

[170] D. Wolters, J. Kirchhoff, C. Gerth, and G. Engels. Cross-device integration of android apps. In Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, editors, *Service-Oriented Computing*, pages 171–185, Cham, 2016. Springer International Publishing.

[171] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu. Clustering web services to facilitate service discovery. *Knowledge and information systems*, 38(1):207–229, 2014.

[172] F. Xie, L. Chen, D. Lin, Z. Zheng, and X. Lin. Personalized service recommendation with mashup group preference in heterogeneous information network. *IEEE Access*, 7:16155–16167, 2019.

[173] O. Yessenbayev, M. Comuzzi, G. Meroni, and D. C. D. Nguyen. A Middleware for Hybrid Blockchain Applications: Towards Fast, Affordable, and Accountable Integration. In *21st International Conference on Service-Oriented Computing - ICSOC 2023, Rome, Italy, November 28th – December 1st, 2023, Proceedings*, Lecture Notes in Computer Science, 2023.

[174] A. V. Zarras, P. Vassiliadis, and I. Dinos. Keep calm and wait for the spike! insights on the evolution of amazon services. In *International Conference on Advanced Information Systems Engineering*, pages 444–458. Springer, 2016.

[175] G. Zou, Z. Qin, Q. He, P. Wang, B. Zhang, and Y. Gan. Deepwsc: Clustering web services via integrating service composability into deep semantic features. *IEEE Transactions on Services Computing*, 15(4):1940–1953, 2022.