Service Oriented Computing and Applications
https://doi.org/10.1007/s11761-019-00277-7

ORIGINAL RESEARCH PAPER

A context-aware recommendation-based system for service composition in smart environments

Soufiane Faieq[1,2] · Agnès Front[1] · Rajaa Saidi[2,3] · Hamid El Ghazi[4] · Moulay Driss Rahmani[2]

Received: 18 June 2019 / Revised: 19 October 2019 / Accepted: 21 October 2019

Abstract
The strong integration of technology in the physical world caused the emergence of smart environments. These environments are supposed to improve the quality of life of their users by providing them with customized services when needed and adapting to their changing needs. The dynamics of the users, the huge number of available services and the strong collaboration between the stakeholders, make traditional service-oriented approaches incapable of providing relevant services to the users in these environments. To deal with these issues, we propose a recommendation-based system for service composition targeting smart environments. The proposed system is able to capture the situation of the users through the analysis of their context information, which in turn allows the system to capture their requirements and select the appropriate service models to satisfy their needs. Then, based on the invocation log, the system implements two recommendation policies. First, it selects the best services in terms of QoS that satisfy the captured requirements. Second, the system recommends new tasks to be integrated in existing service models. The conducted experiments show the efficiency and effectiveness of the recommendation policies proposed. To illustrate the workings of the proposed system, we present a case study called SMARTROAD pertaining to the transport domain and road security.

Keywords Smart environment · Service composition · Service recommendation · Context awareness · Quality of service prediction · Service model

1 Introduction

The world of Information and Communication Technology and Computer Science holds great potential and promise in the transformation of today's big cities toward smart cities. The smart city has been defined as "…an innovative city that uses information and communication technologies (ICTs) and other means to improve quality of life, efficiency of urban operations and services, and competitiveness, while ensuring that it meets the needs of present and future generations with respect to economic, social and environmental aspects" [1]. Harnessing ICT advances is the new way to deal with the issues related to today's big cities (e.g., resource consumption), caused especially by the increasing number of residents within those cities [2]. This strong integration of technology in the physical world caused the emergence of "smart environments." We define these environments as *virtual extensions of the physical world that are capable of providing customized services to their users when needed and adapting them to their changing needs*. Hence, smart environments are characterized by (i) being dynamic, heterogeneous and mobile, (ii) being dependent on the advances of technology, (iii) having a user-centered vision and (iv) providing a myriad of services.

In smart environments, providing services that are relevant to the users is a necessity. There are different types of services that are and will be offered to the citizens in different application domains (e.g., transport, energy, home, health, etc.), by different service providers in the context of smart cities, and their enabling technologies (i.e., data storage services, data

✉ Soufiane Faieq
soufiane.faieq@univ-grenoble-alpes.fr

1 Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), LIG, 38000 Grenoble, France

2 LRIT Associated Unit to CNRST (URAC 29), Faculty of Sciences, Mohammed V University in Rabat, Rabat, Morocco

3 SI2M Laboratory, National Institute of Statistics and Applied Economics, Rabat, Morocco

4 National Institute of Posts and Telecommunications, Rabat, Morocco

Published online: 01 November 2019

processing services, sensing services, etc.) [3]. It comes naturally to think about service-oriented computing (SOC) when evoking the notion of services. Service-oriented computing has proven itself as the go-to paradigm for designing, delivering and consuming services, mainly through the realization of service-oriented architectures (SOA) [4]. Using SOA as a guideline allows the creation of distributed software systems that are loosely coupled, flexible and platform agnostic, mostly based on the principles of reuse and composition of several services that are modular, autonomous, loosely coupled and self-describing and that can be offered internally or through third parties. A popular example technology implementing SOA is Web services. Web services can be defined as Web applications that are able to interact and exchange messages via standard application-to-application protocols over well-defined interfaces. The interaction of more than two Web services is known as "Web Service Composition (WSC)" [5] and aims to develop value-added services while reducing development, integration and maintenance costs and time to market via increasing reuse, sharing and flexibility.

Meanwhile, data science and its related disciplines (e.g., machine learning, data mining, etc.) are expected to play a major role in the "smartification" movement [6]. It is used to develop systems that are capable of delivering actionable information to other systems or to decision makers, through the analysis of the huge amount of data transiting on the network nowadays. Recommender systems are a popular implementation of such systems, as they produce relevant items for the active users based on the analysis of previously recorded actions performed by past users. Recommender systems have shown their effectiveness and have been used for several application domains such as, tourism, e-commerce, news and entertainment. The use of recommender systems in the development of service-oriented software solutions can be beneficial in several aspects, especially those where we need to get actionable information from incomplete and sparse data (e.g., recommending services based on incomplete quality of service (QoS) data) [7].

The characteristics of smart environments mentioned above in the first paragraph make service composition in these environments challenging. On the one hand, the huge number of services available makes it harder to select relevant services for the user [8]. The difficulty here lies, not only in selecting the best service to perform a particular task (i.e., in terms of QoS), but also in analyzing the situation of the user and reacting to it while minimizing the interactions with the user. On the other hand, smart environments (e.g., smart city) are based on the close and strong collaboration between different actors, blurring the lines between the different domains. This makes it hard for domain experts and developers to create, maintain and update collaboration scenarios that are later translated into service models.

To address the challenges discussed above, we explore situational awareness using the notion of context to model and capture the situation of the user. Knowing the situation of the user is helpful for delivering the relevant services to the particular user in a transparent manner. We also explore the use of recommendation techniques to address the challenges at two levels: First, in selecting the best services in terms of non-functional properties for each user (i.e., QoS properties), second, in assisting the developers and domain experts in the maintenance and updates of the service models, through recommending new tasks to be integrated.

In this paper, we propose a context-aware system for service composition based on recommendation. The system captures and processes context information to deduce the situation of its users. Based on the deduced situation, the appropriate service model that is capable of reacting to the needs of each user is selected for execution. To select the best services to satisfy each task in the service model, we use a collaborative filtering framework based on autoencoders to predict the QoS (i.e., QoS) values and then choose the service with the best predicted value. The system also interacts with the domain experts and developers to recommend relevant tasks for integration in the predefined service models. The SMARTROAD case study is presented and used throughout the paper to illustrate the working process of the system.

The rest of this paper is structured as follows. Section 2 introduces the relevant notions and concepts used throughout this paper. Section 3 introduces the motivating scenario behind our system, which is also used to illustrate its working process. The system, its components and processes are described in Sect. 4. Section 5 presents the results of the experiments conducted to evaluate the performance of the recommendation approaches on the data generated by the system. In Sect. 6, we discuss the novelty of our approach when compared to existing systems in the literature. We finally conclude this paper and give some future research directions in Sect. 7.

2 Background

In this section, we give a small introduction into the main concepts that are in play in this paper. We first introduce service composition (SC) as the principle allowing to produce composite services from atomic (unit) services. Second, we present recommender systems (RS) as systems allowing to suggest interesting items to users based on historical records of implicit or explicit user feedback. Lastly, we talk about context awareness and its potential for capturing the situation of the users of the system and reacting to it by providing the users with what they need.

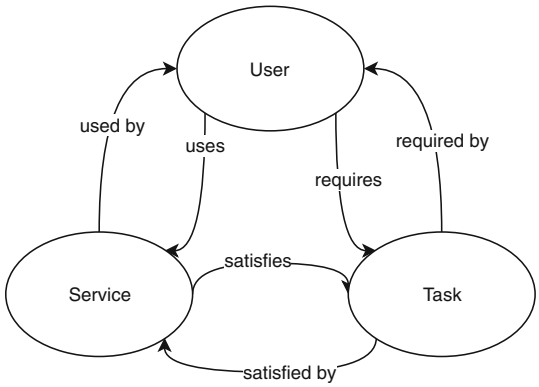


Fig. 1 Interactions between the main concepts in SOC

2.1 Service composition

Service composition (SC) is one of the core design principles of service-oriented computing (SOC) and ensures that atomic services (i.e., simple or unit services) can collaborate and be aggregated into one or more composite services (i.e., complex services) [5]. The service composition process involves several procedures, and these procedures in their turn involve other procedures. According to [9], this process is depicted as a life cycle consisting of four major steps: (1) *Definition*, which allows the translation of the request of the user toward an abstract process (i.e., service model); (2) *Selection*, which achieves the mappings between each task in the service model to a concrete service that is capable of performing the said task; (3) *Deployment*, which converts the abstract process to an executable process using the selected concrete services; (4) *Execution*, which is where the executable process is ran and can be monitored. Hence, there are several concepts and roles involved in typical SC approaches. Figure 1 depicts the interactions between the main actors in typical service-oriented solutions, where three concepts are in constant interaction, namely services, users and tasks.

Services usually perform specific tasks and can be classified into atomic services and composite services [10]. An atomic service is a single unit which can no longer be decomposed, while a composite service is the ordered or parallel execution of two or more services. Tasks can be considered as the functional part of the service. Consequently, it is natural that services follow tasks in their nature (i.e., atomic or composite). In SC, a task (atomic or composite) is also called a *service model* or an abstract service [11] because it represents the functional aspect of the service, ignoring its implementation details. Users can be developers who create the service models (composite tasks) by specifying the tasks that need to be performed and the order in which they are performed later on by the services, or final users who consume the available developed services to accomplish specific tasks.

2.2 Recommender systems

With the plethora of information available online today, users find themselves in a sea of irrelevant data depending on the task they are trying to achieve. Recommender systems (RS) came as a way to address that problem and are highly adopted in e-commerce applications where there is a myriad of products and were also used in different information retrieval problems [12]. In recommender systems, the products to be recommended are generally called Items and these items are recommended to Users. In these systems, we assume that each user can evaluate an item through a Rating system, either explicitly (e.g., the user gives a number of stars on a maximum of five stars) or implicitly (e.g., browsed items in a session, number of clicks, etc.).

Different approaches are used to implement recommender systems depending on the nature of the data describing the users, items and ratings (see Fig. 2). The most popular approach is *Collaborative Filtering* (CF) [13], where the general idea is to model the process of recommendation through collaboration in real life (i.e., Person A recommending to Person B a certain product or movie) to filter the items [14]. This is translated to systems as the assumption that people are more likely to be interested in items that are already liked by other people with a similar taste to their own. CF techniques can be generally classified into neighborhood-based techniques or model-based techniques. Neighborhood-based techniques use the neighborhood of the active user (i.e., users similar to the active user based on past ratings or likes or showing somewhat similar behavior) or the item (i.e., items similar to the items previously liked or highly rated by the active user) to recommend interesting items. These types of recommenders scale linearly to the number of users and items [15]. Model-based techniques use the available information to build a model that is able to generate the recommendation. Most often, the data are transformed into a latent space model where the weights are learned through a training process. Among this family of techniques, methods based on matrix factorization [16] have particularly attracted much attention after they showed good efficiency at the Netflix Prize competition.[1] However, other methods based on neural networks [17], genetic algorithms [18], latent features and graph theory [19] are also widely used.

Content-based recommenders adopt a different concept. They recommend items that are similar or related to the ones that the user liked (or highly rated) in the past. This similarity is usually computed based on the features of the items alone [20]. Thus, this kind of recommenders is based on the assumption that the items can easily be described into categorical data types. This requires a strong domain knowledge which is not always given and can be hard to maintain. For

[1] https://www.netflixprize.com/.

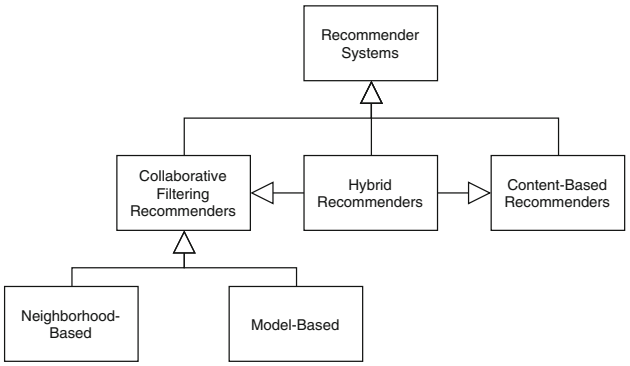


Fig. 2 Recommender systems: types and techniques

example, movies can be described by their genres, cast, director, year of production, etc. Another drawback of these types of systems is serendipity, which is translated by their inability to recommend new items that might interest the user but that are not similar or related to previously rated items.

Hybrid recommenders can be any combination of the previously mentioned recommenders using any combination of techniques [21]. The goal is to leverage the strength of each type of recommender to address or mitigate the drawbacks of another.

2.3 Context awareness

Context information has been defined by Abowd and Dey [22] as "…any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". This definition inspired the concept of context-aware computing which is centered around the ability of a system to collect, analyze and react to the changes in the context. This ability makes systems capable of making smarter decisions and offering more tailored and personalized content, services and experiences.

Due to the excessive amount of information available nowadays, context information can be found everywhere. This information can be of several types, representing different things. For example, location, as a popular context type, can be expressed by an address or by latitude and longitude coordinates. However, some types of context information can have more impact and significance than others, depending on the application. Hence, it is necessary to analyze the impact of each type of contextual information on the performance of the systems to be able to leverage contextual awareness effectively and efficiently [23]. Focusing on the context information that brings the most value also reduces the amount of data to be stored and processed which reflects positively on the performance of context-aware systems.

3 Case study: SMARTROAD

In this section, we introduce a case study to get a better understanding of the problem and the objectives of this paper.

There are 1.35 million human deaths related to traffic accidents worldwide. According to the World Health Organization, the majority of this number is registered in Africa [24]. In Morocco, for example, the road safety situation has been described as a "road war," with close to 82,000 accidents in 2016, an increase of 3.8% when compared to 2015, which is mainly due to the rise of the number in vehicles and automobile industry in the country [25]. This amount of accidents has effects on the economy, environment, society and mobility, and a similar situation is also noticed in the rest of the Arab Maghreb Union and developing countries in general. In this regard, Morocco had launched the national strategy for road safety 2016–2025, aiming at halving the number of deaths in road accidents.

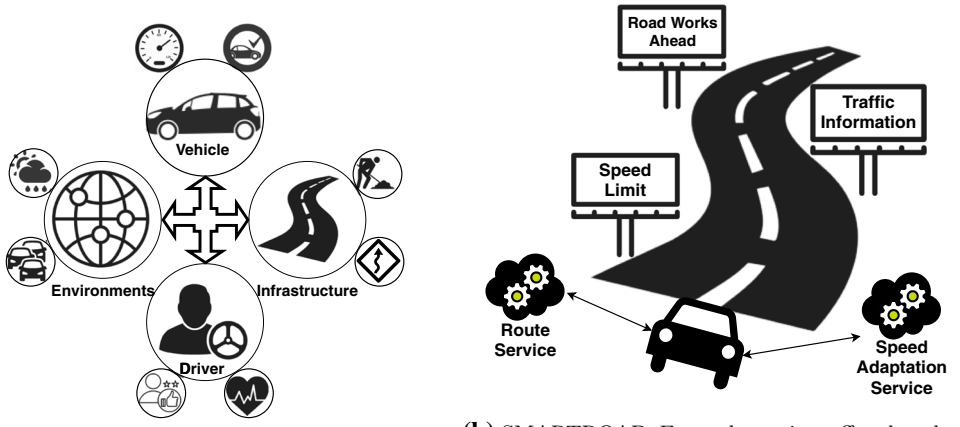
Road safety, however, is only one of the issues related to the surface transport landscape. In fact, transport is at the heart of any nation's continuous and sustainable development. From an organizational perspective, different stakeholders are involved and each one has a particular perspective in the planning, building , managing, using and analyzing of the transport landscape (e.g., Policymakers, vehicle suppliers, energy providers, service providers, end-customers, etc.). These stakeholders are actors that influence the transport ecosystem in mainly four levels: (1) infrastructure, (2) vehicle, (3) driver and (4) environment. Any effort to reduce the number of accidents and deaths on the road has to deal with the pre-crash phase and the post-crash phase with respect to the different actors in the transport ecosystem. In this paper, we focus on the pre-crash phase, where the goal is to manage and minimize the risk factors related to each actor using the services provided by the different stakeholders.

In this paper, we are interested in the services provided to the end-customers by the service providers and city authorities. Table 1 presents some examples of the services that can be offered to the smart city inhabitants, their service provider and their supporting entity [26,27].

To effectively deal with the potentially problematic situations (e.g., traffic, bad weather, etc.) that may arise on the roads and their potential causes (In this paper, we call them *Risk Factors*), several of the services presented in Table 1 have to be composed. We have focused on road safety by defining situations that can be considered dangerous and the risk factors that can contribute to the occurrence of these dangerous situations. We consider that each risk factor can cause a dangerous situation. These risk factors are related to the different entities involved in the road transport ecosystem (i.e., vehicle, driver, infrastructure and environment) and can generally be monitored through sensors that can be accessed via exposed services (e.g., speed, driver health and weather).

Table 1 Examples of provided services

Acronym	Service Name	Stakeholder
AEVW	Approaching Emergency Vehicle Warning	Emergency Services
CBW	Car Breakdown Warning	Automobile Industry
RMS	Road Monitoring Service	Road Operators
VSMS	Vehicle State Monitoring Service	Automobile Industry
ISA	Intelligent Speed Adaptation	Automobile Industry
IVS	In-Vehicle Signage	Automobile Industry
TJAW	Traffic Jam Ahead Warning	Road Operators
ERP	Electronic Road Panel Service	Road Operators
RWW	Road Works Warning	Road Operators
WWS	Weather Warning Service	Weather Services
SMSD	Short Messaging Service for Driver	Automobile Industry
RSS	Route Selection Service	Automobile Industry
ERS	Emergency Rescue Service	Emergency Services
NAAS	Nearby Area Alarming Service	Road Operators
TSS	Towing Selection Service	Navigation Services
OCS	Oil Calculation Service	Automobile Industry
GSS	Garage Selection Service	Navigation Services
GSSS	Gas Station Selection Service	Navigation Services
HSS	Hospital Selection Service	Health-care Services



(a) SMARTROAD: Actors and Risk Factors

(b) SMARTROAD: Example services offered to the driver

Fig. 3 SMARTROAD: actors, risk factors and services

Figure 3a shows these entities and their related risk factors, as defined in this case study.

We linked each situation to a composite service model (composition model) to inform the entities that can be influenced by each given situation. The aim is to make these entities aware of the dangers and to enable the authorities to take the necessary steps to return to a normal state of traffic. Table 2 presents each risk factor, the name of the situation and the service model to mitigate its dangers.

Like almost all smart city applications, the case study presented in this subsection involves the participation of many actors (stakeholders and decision makers), and their exposed services need to work together to achieve the true connectedness of the smart city. In this paper, we divide the actors into final users (drivers) and professionals (transport specialists with software knowledge or developers). In service-based applications, final user's are usually ignorant to the logic of the application. However, the common practice is to allow them to choose the suitable services to perform specific tasks when it is possible, especially when there is no time constraint on the choice being taken (e.g., Garage Selection Service).

4 Recommendation-based system for service composition

In this section, we first present the design of our recommendation-based system for service composition and its com-

Table 2 List of possible situations and their triggered compositions

Situation Name	Triggered Composition
RiskyDriverHealth	ERS, AEVW, ERP, ISA, RSS, IVS, NAAS, IVS
RiskyVehicleState	NAAS, IVS, VSMS, ISA, GSS, RSS, IVS
RiskyRoad	NAAS, ERP, ISA, RMS, RSS, IVS, RWW, SMSD
BadWeather	NAAS, ERP, WWS, ISA, RSS, IVS

ponents through the system architecture and a component diagram. Then, we move on to explain in more details the workings of each component.

4.1 System overview

As mentioned in Sect. 1, the proliferation of services in smart environments has made service selection a challenging task. On the one hand, the selected services should be tailored to the need of the user and deliver the best QoS possible at execution time. On the other hand, the high collaboration between the different stakeholders in these environments makes it difficult for the service developers and domain experts to conceive service models that are capable of satisfying the specific requirements of each user. To address these challenges, we propose in this paper a recommender-based system for context-aware service composition. The goals of this system are threefold:

1. Analyze the context of the user to personalize the provided services to his particular situation and needs
2. Simplify the collaboration procedures between the different stakeholders to define the service models through recommending the tasks to be integrated
3. Offer the best services in terms of quality to each user through analyzing previous invocation data

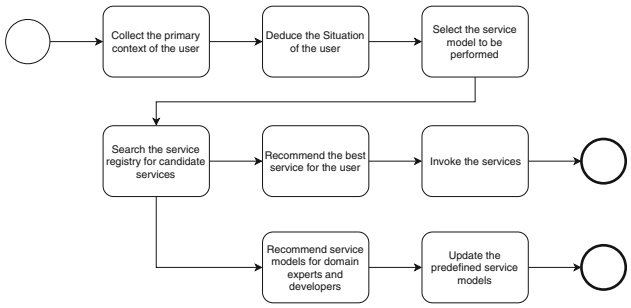


Fig. 4 Execution process of the recommender-based service composition system

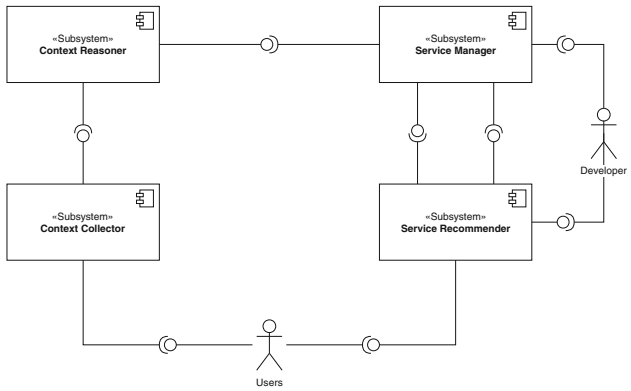


Fig. 5 Recommender-based service composition system: component diagram

Figure 4 shows a process that illustrates the order in which each task of the system is executed. Figure 5 illustrates the actors and the software components of our proposed system. Figure 6 presents the system architecture when applied to the SMARTROAD case study. We chose the layers based on a Separation of Concerns principle [28]. The functions of each layer in the system architecture is detailed in the following subsections.

4.2 Context collection

This is the first step, and the bottom layer that is needed in every context-aware or knowledge-based information system or software. It provides the necessary methods and processes to extract the necessary data from the users of the system. As we stated before in Sect. 3, context is every bit of information that can help in identifying the situation of a user. The context factors to take into account are closely related to the application. In our case study, for example, the goal is to make the road secure. Hence, the context factors that are of interest are the ones that would allow the identification of there being a risk, the nature of the risk, the actors responsible for it and the actors that are affected by it. We denote by $C = (c_1, c_2, \ldots, c_n)$ the set of context factors that are relevant to the application and n the number of these factors.

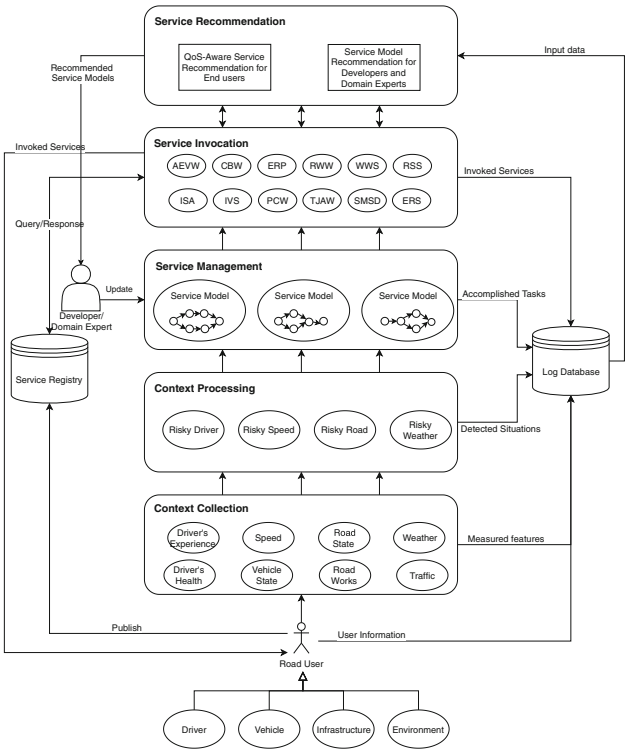


Fig. 6 Recommender-based system architecture for the SMARTROAD prototype

Context factors can further be divided into two categories. The first category is the primary context, which consists of the set of context factors that are directly involved in the logic of the system (e.g., risk factors in our case study). The context factors in this category are a must have for the system to operate. The second category is the secondary context, which assembles the context factors that may enhance the performance of the system in specific tasks (e.g., location of the user). The context factors in this category are not necessary for the system to function. However, they can significantly improve its performance when and if they are available and used effectively.

In a previous work, we proposed a MOF [29] (Meta-Object Facility) compliant metamodel (presented in Fig. 7) to simplify the specification of contextual properties related to the users of the system [30]. This enables the classification of the contextual properties and answers the questions associated with the nature of each context property. At the same time, it allows an abstraction from specifying an exact format to transmit the collected data (e.g., key-value, xml, ontologies, etc.).

In this paper, we assume that the needed context information is accessible through services provided by the actors in the system. In our case study, we consider each risk factor as a primary contextual factor that is monitored through the previously mentioned services (e.g., the speed of the vehicles, the health of the drivers, etc.).

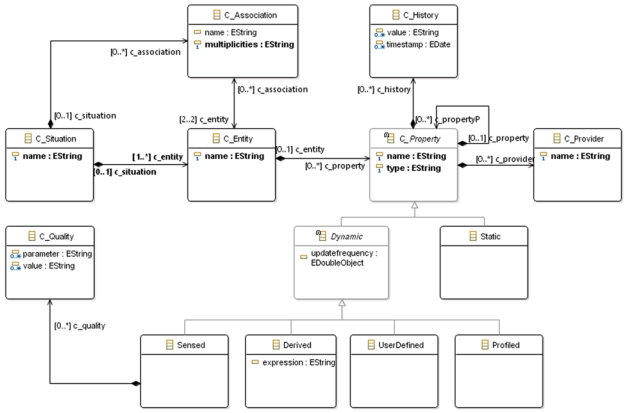


Fig. 7 MOF compliant context metamodel

4.3 Context processing

Context processing is the task of reasoning over the collected context information. The goal of this layer is to be able to identify the situation of one or more users of the system. To do that, we analyze the collected values of the context factors that are of interest in the application. The analysis consists on setting rules about the values of context factors that would allow an action to be performed upon the activation of a rule statement (e.g., *if* VEHICLE_SPEED $\geq$ ALLOWED_SPEED *then* INFORM_POLICE). Domain experts are usually tasked with defining these rules if the application domain is of a sensitive nature (e.g., Road security).

We periodically check the situation of the user through analyzing the different context factors specified at the modeling stage. After applying each specified rule on its corresponding context factor, the result is a one-hot vector of size n. We call this vector the *Status* of the user. An example of the resulting vector based on our case study is presented in Table 3. The example shows that the current driver is inexperienced, driving in a dangerous road and in a bad weather.

4.4 Service model management

Service management encompasses the different tasks that are necessary to the design, storage and execution of orchestrations (i.e., service models). The goal of this layer is to provide the developers and domain experts with the necessary tools to create the service models. Particularly, they need to specify the tasks to be performed, the order in which those tasks will be performed and any eventual data dependencies between the tasks.

This layer reacts to the signals sent by the context processing layer. Indeed, service models are designed to react specifically to each situation that is defined in the context processing layer. The service models are just abstractions of the needed concrete services. They only describe the general

Table 3 User status example

Environment		Infrastructure		Vehicle		Driver	
Traffic	Weather	Condition	Safety	Speed	State	Experience	Health
0	1	1	0	0	0	1	0

behavior of the services and the tasks that are to be performed. For simplicity, we defined a unique service model for each situation. In our case study, the list of situations and their corresponding service models are presented in Table 2.

4.5 Service invocation

The service execution layer is responsible for the discovery, matching and consumption of the concrete services that are available in the registry. The resulting services need to satisfy one or multiple tasks in the service model. The goal here is to handle: (i) checking the availability of the services, (ii) network access between the system and the available services and (iii) the data dependencies of the service (i.e., the inputs required by each service if any). As the service discovery [31] topic is out of the scope of this paper, we only use the name of the task to discover functionally relevant services in the service repository.

4.6 Service recommendation

As mentioned in Sect. 2.2, recommendation systems are an important aspect of knowledge-based information systems. In service computing, service recommendation is a topic of great interest. The recommendations are generated based on one or several characteristics (e.g., time, location, QoS [32], trust, current activity, etc.). In this paper, we focus on two characteristics to generate service recommendations for two different users of the system.

First, for the final users of the system, we generate the best services that are capable of satisfying a particular task. We base the recommendation on the QoS recorded by the system in previous invocations. Second, for the developers and domain experts, we recommend the tasks that may be integrated or added to the predefined service models. To do so, we analyze the services that were invoked by the users and whose service model (i.e., task) is not integrated in the predefined service models.

4.6.1 QoS-aware service recommendation for end users

Smart and ubiquitous environments offer a big number of services potentially providing the same functionality to the users. QoS properties are used as non-functional parameters to select the best services. However, recorded QoS properties depend on the user, the service and their corresponding context. This makes the task of selecting the best service for a particular user troublesome. Furthermore, the knowledge about QoS properties is not complete as that would imply that every user has invoked every available service. This in turn makes service selection even more challenging.

The goal here is to provide the best service in terms of QoS to achieve each task in the service model. Knowing that a user only invokes a small set of the available services. This translates to a prediction problem where the goal is to predict the missing values of a matrix $R \in \mathbb{R}^{nu \times ns}$ where nu is the number of users and ns is the number of services. The rows of the matrix R represent the users of the system U, where each user $u \in U = (1, \ldots, nu)$ can be represented by a partially observed vector $r_{(u)} = (R_{u,1}, \ldots, R_{u,ns}) \in \mathbb{R}^{ns}$. The columns of R represent the services S, where each service $s \in S = (1, \ldots, ns)$ can be represented by a partially observed vector $r_{(s)} = (R_{s,1}, \ldots, R_{s,nu}) \in \mathbb{R}^{nu}$. Each cell in $c \in R$ represents a QoS property value recorded when a user $u \in U$ invokes a service $s \in S$.

The challenges here are mainly related to the nature of the data generated by the service. In this paper, we focused on QoS data which are objective, meaning that contrary to classic recommendation problems where the data are subjective and hence reflect the user's satisfaction with an item (i.e., movie, music, news, etc.); data in QoS service recommendation data are objective and hence only represent objective factors about the services for it a certain user. Basically, that means that unlike recommendations over subjective data where identical user is more likely to give a close ratings for the same item, it is highly possible that the identical users have varying ratings for the same service in QoS data.

To achieve this task, we adopted the framework proposed by [33] called AutoRec, which is a collaboration filtering framework based on autoencoders. In their approach, they aim to design an autoencoder which can take as input a set of partially observed $r_{(s)}$ or $r_{(u)}$, project it into a low-dimensional latent space and then reconstruct it in the output space to predict missing ratings. They formally defined the problem as, given a set X of vectors in $\mathbb{R}^d$, and some $k \in \mathbb{N}_+$, an autoencoder solves

$$\min_{\theta} \sum_{r \in S} \|r - h(r; \theta)\|_2^2,$$

where $h(r; \theta)$ is the reconstruction of input $r \in \mathbb{R}^d$,

$$h(r; \theta) = f(W.g(Vr + \mu) + b)$$

$f(.)$ and $g(.)$ are activation functions. Here, $\theta = \{W; V; \mu; b\}$, where $V \in \mathbb{R}^{k \times d}$ and $W \in \mathbb{R}^{d \times k}$ are the weight matrices for encoding (i.e., projecting) and decoding (i.e., reconstructing) the input, respectively, and $\mu \in \mathbb{R}^k$ and $b \in \mathbb{R}^d$ are biases. This objective corresponds to an auto-associative neural network [34], with a single hidden layer (HL) with k dimension. The parameters θ are learned using backpropagation. To avoid overfitting, we regularize the learned parameters by introducing the regularization term $R = \|W\|_F^2 + \|V\|_F^2$ and use a regularization rate parameter λ to control the strength of the regularization.

To the best of our knowledge, the application of this approach in the service computing domain was never attempted. Hence, this paper can be considered a stepping stone toward the integration and application of more deep learning frameworks in service computing.

4.6.2 Service model recommendation for developers and domain experts

Nowadays, the changes in business processes are quite frequent due to changing market, policies, misconceptions, new trends, etc. Domain experts and developers cannot keep up with these changes. This exposes the need for recommendations about the tasks to be integrated in the predefined service models and business processes.

The goal here is to inform domain experts and developers about the need for potential changes in the predefined service models. Even further, we aim to provide them with the potential elements to be integrated into that change. To do so, we analyze the execution traces (i.e., $Events$) that were recorded to discover new patterns in the data. We proceed first by filtering the traces to extract the tasks which do not belong to any predefined service model (i.e., $UniqueAtomicTasks$, where UAT which means $UniqueAtomicTask$ is an element of $UniqueAtomicTasks$). In the mean time, we group the atomic events together by task (i.e., $GroupedAtomicEvents$, where GAE which means $GroupedAtomic Event$ is an element of $GroupedAtomicEvents$). Second, we extract the set of users who needed the tasks of the invoked services (i.e., $Users$, where U which means $User$ is an element of $Users$). Third, we analyze the events targeting those users to extract the most frequently invoked service models (i.e., $MostFrequentServiceModelSets$, where FSM which means $FrequentServiceModel$ is an element of $MostFrequentServiceModelSets$). In this final step, we use the FPGrowth algorithm [35], which is a classic Frequent Item Mining and Association Rules Mining approach known for its efficiency and effectiveness when compared to other known algorithms like Apriori and Eclat [36]. The general process is presented in algorithm 1.

The proposed algorithm takes as input two parameters. First, a log file containing a set of *Events*. Each *Event* in the log file is a tuple $(User, Task, ServiceModel)$ where each tuple (i.e., event) represents a performed task, its targeted user and the service model it belongs to if any. Second, a value P with $0 < P \leq 100$ represents a percentage threshold after which the recommendation becomes relevant to the domain experts and developers. Note, this threshold affects only atomic tasks and is used as a minimum support to check whether an atomic task is frequently associated with a service model. As output, the algorithm returns a set of recommendations R. Each $r \in R$ is a tuple (SM, AT) where SM is a predefined service model and AT is an atomic task than be used to extend SM.

Filtering the data allows to only extract the itemsets that are associated with the atomic tasks. However, the domain experts and the developers still have to configure the frequent itemset mining algorithm by setting the minimum support. The setup of the algorithm's parameters is needed to make the decision of adding the atomic task to the predefined service models.

5 Experiments and results

This section describes the experiments that we conducted on the recommendation layer. We start by describing the data preparation process. Then, we present the results of our experiments, before starting a discussion to interpret the results.

5.1 Data preparation

To conduct our experiments successfully, we collected the data generated from each layer of our approach. At the bottom layer (i.e., context collection), the data consist of the users list and the values of their corresponding context factors. The directly above layer (i.e., context processing) generates the situations based on the values of context factors. The service management layer generates the data related to the service models. Service invocation generates the data about the services that are invoked by the users. We extracted the data using a developed simulation of the case study presented in Sect. 3 and extracted the QoS values from the WSDREAM dataset [2] collected by [7]. The dataset describes the real-world QoS evaluation results from 339 users on 5825 Web services. The dataset also contains the location information (e.g., country, autonomous system, latitude, longitude) of the users and the services. For more details, users can refer to the work of [7]. In our prototype, we map the country to

[2] https://github.com/wsdream/wsdream-dataset/tree/master/dataset1.

Algorithm 1: Algorithm for Service Model Recommendation for Developers and Domain Experts

Input : The set of recorded events $Events$, such that each event is described as: $Event = (User, Task, ServiceModel)$ and P being the threshold after which the service is considered relevant for recommendation

Output: Set of pairs $R = (SM, AT)$, Where SM is the predefined Service Model and AT is the recommended Atomic Task for integration

$R \leftarrow \emptyset$
$AtomicEvents \leftarrow getAtomicEvents(Events)$
    // Events with empty ServiceModel field
$GroupedAtomicEvents \leftarrow AtomicEvents.GroupbyTask()$
$UniqueAtomicTasks \leftarrow AtomicEvents.GetUniqueTasks()$
foreach $UAT \in UniqueAtomicTasks$ **do**
    foreach *GAE in GroupedAtomicEvents.getGroup(UAT)* **do**
        $Users \leftarrow GAE.getUsers()$
        $NumberOfUsers \leftarrow Users.getCount()$
        $ServiceModels \leftarrow \emptyset$
        foreach $U \in Users$ **do**
            $ServiceModels.Add(U.getServiceModels())$
        end
        $FrequentServiceModels \leftarrow FPGrowth(ServiceModels, NumberOfUsers \times \frac{P}{100})$
        if $FrequentServiceModels \neq \emptyset$ **then**
            $MostFrequentServiceModelSets \leftarrow FrequentServiceModels.pop(3)$
                // get first three results
            foreach $FSM \in MostFrequentServiceModelSets$ **do**
                $R.Add(FSM, UAT)$
            end
        end
    end
end
return R

the environment and the AS (i.e., autonomous system) to the infrastructure in both the services and the users files.

We aggregated and preprocessed the data to generate three separate files. The files contain the following elements:

- The *users* data file containing the information about the users (i.e., the values of their context factors and their situation)
- The *services* data file containing the information about the services (e.g., the tasks they perform, location, etc.)
- The *invocations* data file containing the information about the invoked services by each user (i.e., QoS parameters). To simplify the process of reconstructing the service models and filtering the atomic tasks, we also stored the preceding task when performing the current task and the service model it belongs to. Table 4 shows an extract of the invocations file, where the user column represents the identifiers of each user, the service column represents the identifiers of the invoked services, the SM column represents the service model from which the service was executed (indicating the user's situation) and response time and throughput represent objective QoS data about the invocation of each service by each user. The *invocation* data file is available on *csv* format in the linked repository at footnote.[3]

5.2 Results

5.2.1 QoS-aware service recommendation for end users

Experiment Setup We ran an implementation of the user-based AutoRec (U-AutoRec) approach on the data collected in the *invocations* file using TensorFlow.[4] The experiment was conducted using a laptop with a seventh generation Intel i5 processor, 8GB of RAM and 6MB of cache memory. The model parameters chosen throughout the experiments are presented in Table 5.

As mentioned in Sect. 4.6.1, the knowledge about QoS properties is not complete in real use cases, as that would imply that every user has invoked every available service. To reflect this aspect of sparsity on the WSDREAM dataset, we used less of one-third of the data to build the model. Particularly, we built the model using four different matrix density levels: $MatrixDensity = 5\%, 10\%, 20\%, 30\%$. $MatrixDensity = 5\%$ means that we randomly select 5% of the data to predict the remaining 95%. Each experiment was ran five times.

Evaluation Metrics To evaluate our U-AutoRec-based model when compared to other QoS prediction methods, we focused on the accuracy of the predicted values. We define accuracy as a function describing the distance between the observed data and the predicted data. In our case, the observed data correspond to the QoS data recorded in the WSDREAM dataset, meaning response time and throughput data. The predicted data correspond to the data predicted by our model as an output. In this context, we use MAE and RMSE as two basic and popular error metrics that are commonly used in evaluating recommender systems [15]. While MAE measures the mean magnitude of errors over a set of prediction, RMSE is the square root of the average of squared differences between prediction and actual observation. This makes RMSE more interesting when large errors have a lot more negative effects on the system than small errors. MAE and RMSE are defined as:

[3] https://github.com/faieqs/SMRec.

[4] https://www.tensorflow.org/.

Table 4 Example of the *invocation* data file generated by the SMARTROAD system

User	SM	Service	Throughput	Response time
User_7		Weather_serv_03	24.1	5.12
User_2	Risky vehicle	SpeedAdaptation_serv_2	14.5	0.48
User_42	Risky vehicle	Garage_serv_4	5.4	1.45
User_18		Towing_serv_06	16.34	2.15
User_68		Traffic_serv_01	45.14	1.84

Table 5 Model parameters

N° HL	k	Learning rate	λ	Batch size	Optimizer	A. Function
1	500	0.0001	0.1	32	Adam	Sigmoid

$$\text{MAE} = \frac{\sum_{u,s} \mid r_{u,s} - \widehat{r}_{u,s} \mid}{N}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{u,s}(r_{u,s} - \widehat{r}_{u,s})^2}{N}}$$

where $r_{u,s}$ denotes the expected QoS value of service s observed by user u, $\widehat{r}_{u,s}$ is the predicted QoS value and N is the number of values.

Performance Comparison We compare the performance of U-AutoRec with the performance of several other methods. In the literature, each method was evaluated on a different set of $MatrixDensity$ values. This makes the comparison highly challenging as source code is rarely publicly available. Hence, in this paper, we compared the U-AutoRec method with the methods reported in the WS-DREAM package [37]. The package reports the performance of several methods. Some of these methods are based on neighborhood [38–40], some on models [41–43] and others are location-aware [44–48]. Table 6 shows the mean MAE and RMSE values of different prediction methods in response time (RT) measured in seconds and throughput (TP) measured in Kbps, using 5, 10, 20 and 30 percent as $MatrixDensity$ (MD) values.

5.2.2 Service model recommendation for developers and domain experts

To validate our approach, we injected records of executed atomic services in the invocations file with different probabilities. The objective is to prove that the proposed algorithm is capable of extracting and recommending relevant atomic tasks for them to integrate into existing service models, thus achieving continuous improvements. Hence, the accuracy metric for this part is the capacity of our algorithm to discover these injected patterns. We considered two situations where we injected the following patterns.

- That 80% of the users who encountered a RiskyVehicleState situation have searched for a Gas Station to fill the gas tank of their vehicle (i.e., GSSS). Note, the GSSS service is not part of any predefined service model (see Fig. 2).
- That 40% of the users who encountered a RiskyVehicleState situation have searched for a Towing service to transport their vehicles (i.e., TSS).

We ran an implementation of algorithm 1 in Python, which is available in the linked repository at footnote .[5] As inputs, we entered the path to the *invocation* file and $P = 50$ is the threshold percentage of the users who have performed the task before it is linked to the service model. While no results were found for the TSS service, the GSSS was recommended to be integrated in the RiskyVehicleState as expected. We expect the algorithm to reveal interesting recommendations when applied on real data.

5.3 Analysis and discussion

The results presented in Table 6 show that model-based methods generally outperform those based on location awareness or neighborhood. Furthermore, U-AutoRec performs exceptionally well even when compared to other model-based methods. This is especially true, when analyzing the performance of the approach on throughput (TP) data, where U-AutoRec achieves 33.2% gains in MAE and 35.3% better in RMSE, when compared with the best approaches over all matrix density values individually. On the throughput (TP) data, U-AutoRec also scores an improvement of 32.1% in MAE and a 29.4% gain on RMSE.

While U-AutoRec already outperforms the other methods in most cases, there is still room for improvement. Indeed, with the huge number of combination that can be used to finetune the model (i.e., number of iterations, optimizers, number of hidden nodes, activation functions, etc.), it is most likely that our experiment did not reach the limits of the AutoRec framework. Extensive experiments and finetuning

[5] https://github.com/faieqs/SMRec.

Table 6 Performance comparison between the QoS prediction models

Methods	MD = 5%				MD = 10%				MD = 20%				MD = 30%			
	RT		TP		RT		TP		RT		TP		RT		TP	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
UIPCC [38]	0.6253	1.3879	26.7568	60.7985	0.5815	1.3302	22.3700	54.4563	0.4498	1.1968	18.9276	48.2950	0.4110	1.1422	17.0797	45.0599
ADF [39]	0.6094	1.3613	24.9961	60.7939	0.5443	1.2924	21.5013	54.2893	0.4636	1.1898	16.6536	45.2008	0.4276	1.1398	14.8244	41.7186
NRCF [40]	0.5532	1.4547	23.3275	59.9498	0.4905	1.3678	18.8571	52.9977	0.4261	1.2581	14.3444	44.5142	0.4059	1.1975	12.8267	40.7493
RegionKNN [44]	0.5883	1.5426	25.6324	67.8678	0.5477	1.5129	24.8380	67.5510	0.5158	1.5214	24.0361	66.1760	0.5091	1.5193	23.7984	65.5971
LACF [45]	0.6374	1.4436	23.1685	58.9666	0.5659	1.3420	19.6257	53.1050	0.4827	1.2298	16.6669	47.6247	0.4453	1.1720	15.2358	44.7729
LBR [46]	0.5499	1.4741	18.3187	56.0215	0.4802	1.2858	15.4272	47.0197	0.4300	1.1610	13.6512	41.3552	0.4103	1.1214	12.9824	39.2269
HMF [48]	0.5595	1.5248	19.1320	58.7088	0.4815	1.3105	15.7187	48.3461	0.4296	1.1661	13.6319	41.6678	0.4072	1.1178	12.7767	39.0613
LoRec [47]	0.6479	1.3957	27.7773	62.5533	0.5557	1.3087	24.7118	58.4275	0.4659	1.2040	21.7440	54.0139	0.4437	1.1504	20.5338	52.0747
PMF [41,49]	0.5690	1.5371	19.0946	57.9078	0.4866	1.3163	15.9741	48.0393	0.4308	1.1690	13.9085	41.7277	0.4082	1.1206	13.1782	39.4635
NMF [42,50]	0.5456	1.4727	18.8824	57.5177	0.4775	1.2824	15.5717	47.7903	0.4291	1.1616	13.5259	41.6482	0.4063	1.1191	12.7851	39.5068
BiasedMF [43]	0.5934	1.3821	21.8369	56.8575	0.5135	1.2625	17.8522	48.3275	0.4569	1.1786	14.9224	42.1408	0.4280	1.1416	13.6980	39.6656
LN-LFM [43]	0.5602	1.3065	20.4018	52.4209	0.5007	1.2284	17.7273	46.9713	0.4512	1.1541	16.3428	44.1434	0.4351	1.1282	16.0625	43.4570
Our model	0.361	1.010	13.542	37.119	0.3343	0.9035	10.360	30.668	0.2953	0.8019	8.644	26.569	0.2685	0.7688	7.952	24.562

techniques are needed in this direction to identify the best parameters for the model.

We have also ran experiments using I-AutoRec, which follows the same model as U-AutoRec with the difference that it relies on the items representations rather than the users representations to build the model. I-AutoRec revealed some interesting results. I-AutoRec performs generally better on throughput data, reaching an MAE = 14.63 and an RMSE = 38.26, when MD = 10%, and an MAE = 12.55 and an RMSE = 34.77, when MD = 20%. However, U-AutoRec still outperforms I-AutoRec in both response time (RT) and throughput (TP) data. This leads us to believe that response time data are more sensitive to the features of the users, while throughput data are also sensitive to the features of the services. This is logical as response time would change depending on the network state of the users (e.g., signal strength), while throughput depends on the network state of the service's supporting infrastructure and that of the user.

6 Related works

Service-based systems and their application have evolved a lot over the last two decades. Recently, researchers have focused on using the advances in AI planning to automate the service composition process [51]. One of the most popular techniques are those based on HTN (Hierarchical Task Network). Among these, figures the work of Wu et al. [52], where the authors propose a HTN-based (Hierarchical Task Network) planner called SHOP2 to automatically compose service in DAML-S. Other techniques like [53] rely on the advances in Semantic Web, using ontologies to automatically compose services. However, both approaches are still incapable of handling naturally complex real- world applications. Which is why most service solutions in industry still use manual or semiautomatic techniques in which the expertise of domain experts and developers is strongly needed [9].

However, manual and semiautomatic techniques are not without their challenges. While domain experts usually have the necessary knowledge to compose relevant service models, they cannot create personalized services that are tailored to the preferences of the users. This lack in personalization aspects was discussed by [9]. In fact, in their survey, only 8% of the included works have investigated personalization. This problem gains more emphasis when the environments are of a distributed and ubiquitous nature (e.g., smart environments). To tackle this problem, context-aware approaches have been proposed by many researchers to integrate context information at different steps of the composition process [26,54]. Most of the proposed approaches rely on ontologies to represent context information and their innate use of first-order logic to reason on the collected information. In our approach, we integrated contextual information to extract the situation

of the user through a metamodeling approach for more generality.

Even though these methods are capable of identifying the particular needs, requirements and preferences of the users, domain experts and developers still find themselves unable to capture the different deviations and variations at user level and enact them in the defined service models. To this end, it was necessary to introduce new techniques to support them in this task. Existing methods suggest analyzing existing service models to help the users (i.e., domain expert and developers) in the creation of new service models [55–57]. In our approach, we analyze the history of used services to recommend services to be integrated into existing service models or to be added as new variability models.

Service selection is another important problem when dealing with distributed and ubiquitous environments, given the number of available services and the increasing mobility and dynamics of the users. Recently, recommender systems have been widely investigated, to great success, in dealing with this problem of selecting the best services for a particular user or a set of users [7]. Collaborative filtering techniques in particular have been very successful in dealing with this problem. Most of the methods proposed in the literature use registered QoS properties to recommend the best services. These methods can usually be classified into three categories: (1) neighborhood-based methods, relying on the similarity between the users in terms of past recorded QoS parameters to predict the values of missing values of QoS properties [38–40]; (2) location-aware methods, using the similarity between the user profiles in general and location in particular to predict the values of missing values of QoS properties [44–47]; and (3) model-based methods, using machine learning techniques to build models that are capable of predicting the values of missing values of QoS properties [41–43]. Model-based methods generally perform better than the other categories. In this paper, we build a model using the AutoRec framework [33], which outperforms other model-based methods.

7 Conclusion and future work

Nowadays, ubiquitous environments have made it difficult to propose relevant services to the users. First, due to the increasing mobility of the users and the different devices connected to the Internet, capturing the situation of the users is challenging. Second, the proliferation of available services in these environments makes the task of selecting the best services for each user problematic. Third, the variety of user requirements and needs makes the task of designing service models that are capable of satisfying each user needs highly complex.

In this paper, we proposed a recommendation-based system for service composition targeting smart environments. The main contributions of this paper can be summarized in four points: (1) a conceptual architecture for a context-aware recommendation-based system for service composition; (2) a metamodel to represent the situation of the users through context information; (3) a model based on the AutoRec framework for the prediction of QoS values and (4) an algorithm for recommending tasks to be integrated in existing service models or creating new variability models. The conducted experiments show the efficiency and effectiveness of the recommendation policies proposed. QoS prediction-wise, we achieved an improvement of 30% on MAE over existing methods. Task recommendation-wise, the proposed algorithm produces the expected results when applied to synthetic data.

Deep learning frameworks have shown their efficiency and effectiveness on many application domains. The application of deep learning algorithms in recommender systems is supposed to enhance their performance even further [58]. Research has shown the improvements in performance when applied to standard datasets such as MovieLens. Hence, it would be beneficial to explore the use of such frameworks to build new models for QoS prediction for service recommendation, as they can potentially provide better performances.

To date, there exists no dataset or log files recording the behavior of composite services. The service computing community could highly benefit from such data. The data could be used to explore the effect of the composition on the selection algorithm, meaning that considering two services s_1 and s_2 offering the same task t_1, where s_1 is better than s_2 and two other services s_3 and s_4 offering the same task t_2, where s_3 is better than s_4; would the composition of $s_1 \rightarrow s_3$ be better than the composition of $s_2 \rightarrow s_3$? and if not, then the research should maybe focus on predicting QoS over service patterns rather than individual services.

Another research direction that is worth investigating in recommender systems in general, and in service recommendation in particular, is the impact of these systems on the performance of the recommended services. Indeed, recommending the same service to a myriad of users could potentially be dangerous, as the service could be overloaded with multiple requests at the same time, which we assume could result in a serious drop in its QoS. Having the knowledge about the computing environment of the services could help investigate how many requests can a service handle before it suffers QoS drops. New recommender systems would need to take this information into account in the selection process to avoid negative effects on the service.

Acknowledgements This project was financially supported by CAMPUS FRANCE (PHC TOUBKAL 2017 (French-Morocco bilateral program) Grant Number: 36804YH).

References

1. Kondepudi SN, Ramanarayanan V, Jain A, Singh GN, Nitin Agarwal NK, Kumar R, Singh R, Bergmark P, Hashitani T, Gemma P et al (2014) Smart sustainable cities analysis of definitions. The ITU-T focus group for smart sustainable cities
2. DESA United Nations (2015) World urbanization prospects: the 2014 revision. United Nations Department of Economics and Social Affairs, Population Division, New York, NY, USA
3. Schaffers H, Komninos N, Pallot M, Trousse B, Nilsson M, Oliveira A (2011) Smart cities and the future internet: towards cooperation frameworks for open innovation. In: The future internet. Springer, Berlin, pp 431–446
4. Bieberstein N, Bose S, Walker L, Lynch A (2005) Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. IBM Syst J 44(4):691–708
5. Baresi L, Di Nitto E, Ghezzi C, Guinea S (2007) A framework for the deployment of adaptable web service compositions. Serv Oriented Comput Appl 1(1):75–91
6. Hashem IAT, Chang V, Anuar NB, Ibrar Yaqoob KA, Gani A, Ahmed E, Chiroma H (2016) The role of big data in smart city. Int J Inf Manag 36(5):748–758
7. Zheng Z, Zhang Y, Lyu MR (2014) Investigating qos of real-world web services. IEEE Trans Serv Comput 7(1):32–39
8. Jin X, Chun S, Jung J, Lee K-H (2017) A fast and scalable approach for iot service selection based on a physical service model. Inf Syst Front 19(6):1357–1372
9. Sheng QZ, Qiao X, Vasilakos AV, Szabo C, Bourne S, Xiaofei X (2014) Web services composition: a decade's overview. Inf Sci 280:218–238
10. Casati F, Shan M-C (2001) Dynamic and adaptive composition of e-services. Inf Syst 26(3):143–163
11. Fang J, Hu S, Han Y (2004) A service interoperability assessment model for service composition. In: IEEE international conference on services computing, 2004 (SCC 2004). Proceedings. 2004, pp 153–158
12. Konstan JA, Riedl J (2012) Recommender systems: from algorithms to user experience. User Model User Adapt Interact 22(1):101–123
13. Schafer JB, Frankowski D, Herlocker J, Sen S (2007) Collaborative filtering recommender systems. In: The adaptive web. Springer, pp 291–324
14. Resnick P, Iacovou N, Suchak M, Bergstrom P, Riedl J (1994) Grouplens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on computer supported cooperative work. CSCW '94. ACM, New York, NY, USA, pp 175–186
15. Candillier L, Meyer F, Boullé M (2007) Comparing state-of-the-art collaborative filtering systems. In: Petra P (ed) Machine learning and data mining in pattern recognition. Springer, Berlin, pp 548–562
16. Koren Y, Bell R, Volinsky C (2009) Matrix factorization techniques for recommender systems. Computer 42(8):30–37
17. Lee M, Choi P, Woo Y (2002) A hybrid recommender system combining collaborative filtering with neural network. In: International conference on adaptive hypermedia and adaptive web-based systems. Springer, pp 531–534
18. Bobadilla J, Ortega F, Hernando A, Alcalá J (2011) Improving collaborative filtering recommender system results and performance using genetic algorithms. Knowl Based Syst 24(8):1310–1316
19. Aggarwal CC, Wolf JL, Wu K-L, Yu PS (1999) Horting hatches an egg: a new graph-theoretic approach to collaborative filtering. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '99. ACM, New York, NY, USA, pp 201–212
20. Son J, Kim SB (2017) Content-based filtering for recommendation systems using multiattribute networks. Expert Syst Appl 89:404–412
21. Burke R (2002) Hybrid recommender systems: survey and experiments. User Model User Adapt Interact 12(4):331–370
22. Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P (1999) Towards a better understanding of context and context-awareness. In: International symposium on handheld and ubiquitous computing. Springer, pp 304–307
23. Pinheiro MK, Souveyet C (2018) Supporting context on software applications: a survey on context engineering. Modélisation et utilisation du contexte 2(1):1–29. https://doi.org/10.21494/ISTE.OP.2018.0275
24. World Health Organization et al (2018) Global status report on road safety 2018. World Health Organization, Geneva
25. National Committee for the Prevention of Traffic Accidents (Morocco). Provisional assessment of road accidents for 2017, 2016
26. Chang J, Yao W, Li X (2017) The design of a context-aware service system in intelligent transportation system. Int J Distrib Sens Netw 13(10):1550147717738165
27. Ait-Cheik-Bihi W, Nait-Sidi-Moh A, Bakhouya M, Gaber J, Wack M (2012) Transportml platform for collaborative location-based services. Serv Oriented Comput Appl 6(4):363–378
28. Dijkstra EW (1982) On the role of scientific thought. In: Selected writings on computing: a personal perspective. Springer, Berlin, pp 60–66. ISBN 0-387-90652-5
29. OMG (2014) Meta object facility meta object facility (MOF) core specification. Adopted Specification, OMG document formal/14-04-03. https://www.omg.org/spec/MOF/2.4.2
30. Faieq S, Saidi R, Elghazi H, Rahmani MD (2017) A conceptual architecture for a cloud-based context-aware service composition. In: Advances in ubiquitous networking 2. Springer, Singapore, pp 235–246
31. Gu Q, Lago P (2009) Exploring service-oriented system engineering challenges: a systematic literature review. Serv Orient Comput Appl 3(3):171–188
32. Chengyuan Y, Huang L (2016) A web service qos prediction approach based on time- and location-aware collaborative filtering. Serv Oriented Comput Appl 10(2):135–149
33. Sedhain S, Menon AK, Sanner S, Xie L (2015) Autorec: autoencoders meet collaborative filtering. In: Proceedings of the 24th international conference on world wide web, WWW '15 Companion. ACM, New York, NY, USA, pp 111–112
34. Stone VM (2008) The auto-associative neural network—a network architecture worth considering. In: 2008 world automation congress, pp 1–4
35. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD international conference on management of data, SIGMOD'00. ACM, New York, NY, USA, pp 1–12
36. Garg K, Kumar D (2013) Comparing the performance of frequent pattern mining algorithms. Int J Comput Appl 69(25):21–28
37. WSDREAM team (2011) Ws-dream: A package of open source-code and datasets to benchmark QoS prediction approaches of web services
38. Zheng Z, Ma H, Lyu MR, King I (2011) Qos-aware web service recommendation by collaborative filtering. IEEE Trans Serv Comput 4(2):140–152
39. Wu J, Chen L, Feng Y, Zheng Z, Zhou MC, Wu Z (2013) Predicting quality of service for selection by neighborhood-based collaborative filtering. IEEE Trans Syst Man Cybern Syst 43(2):428–439
40. Sun H, Zheng Z, Chen J, Lyu MR (2013) Personalized web service recommendation via normal recovery collaborative filtering. IEEE Trans Serv Comput 6(4):573–579

41. Zheng Z, Ma H, Lyu MR, King I (2013) Collaborative web service qos prediction via neighborhood integrated matrix factorization. IEEE Trans Serv Comput 6(3):289–299
42. Zhang Y, Zheng Z, Lyu MR (2011) Exploring latent features for memory-based qos prediction in cloud computing. In: 2011 IEEE 30th international symposium on reliable distributed systems, pp 1–10
43. Yu D, Liu Y, Xu Y, Yin Y (2014) Personalized QoS prediction for web services using latent factor models. In: 2014 IEEE international conference on services computing, pp 107–114
44. Chen X, Liu X, Huang Z, Sun H (2010) Regionknn: a scalable hybrid collaborative filtering algorithm for personalized web service recommendation. In: 2010 IEEE international conference on web services, pp 9–16
45. Tang M, Jiang Y, Liu J, Liu X (2012) Location-aware collaborative filtering for QoS-based service recommendation. In: 2012 IEEE 19th international conference on web services, pp 202–209
46. Lo W, Yin J, Deng S, Li Y, Wu Z (2012) Collaborative web service GOS prediction with location-based regularization. In: 2012 IEEE 19th international conference on web services, pp 464–471
47. Chen X, Zheng Z, Yu Q, Lyu MR (2014) Web service recommendation via exploiting location and qos information. IEEE Trans Parallel Distrib Syst 25(7):1913–1924
48. He P, Zhu J, Zheng Z, Xu J, Lyu MR (2014) Location-based hierarchical matrix factorization for web service recommendation. In: 2014 IEEE international conference on web services, pp 297–304
49. Salakhutdinov R, Mnih A (2007) Probabilistic matrix factorization. In: Proceedings of the 20th international conference on neural information processing systems, NIPS'07. Curran Associates Inc, USA, pp 1257–1264
50. Lee Daniel D, Seung, H. Sebastian (2000) Algorithms for non-negative matrix factorization. In: Proceedings of the 13th international conference on neural information processing systems, NIPS'00. MIT Press, Cambridge, MA, USA, pp 535–541
51. Rao J, Su X (2005) A survey of automated web service composition methods. In: Semantic web services and web process composition. Springer, Berlin, pp 43–54
52. Wu D, Parsia B, Sirin E, Hendler J, Nau D (2003) Automating DAML-S web services composition using shop 2. In: Proceedings of the second international conference on semantic web conference, LNCS-ISWC'03. Springer, Berlin, pp 195–210
53. Slaimi F, Hassine AB, Tagina M (2014) Ontology based vertical web service composition. Int J Know Based Intell Eng Syst 18(1):1–12
54. Furno A, Zimeo E (2014) Context-aware composition of semantic web services. Mobile Netw Appl 19(2):235–248
55. Chowdhury SR, Daniel F, Casati F (2011) Efficient, interactive recommendation of mashup composition knowledge. In: Service-oriented computing. Springer, Berlin, pp 374–388
56. Budiselić I (2014) Component recommendation for development of composite consumer applications. PhD thesis, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu
57. Greenshpan O, Milo T, Polyzotis N (2009) Autocompletion for mashups. Proc VLDB Endow 2(1):538–549
58. Zhang S, Yao L, Sun A, Tay Y (2017) Deep learning based recommender system: a survey and new perspectives. arXiv preprint arXiv:1707.07435