

Service Oriented Computing in Mobile Environments: Abstractions and Mechanisms for Interoperability and Composition

Nikolaos Georgantas

▶ **To cite this version:**

Nikolaos Georgantas. Service Oriented Computing in Mobile Environments: Abstractions and Mechanisms for Interoperability and Composition. Computer Science [cs]. Sorbonne Université, 2018. tel-01740629

HAL Id: tel-01740629
https://inria.hal.science/tel-01740629

Submitted on 22 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.





SORBONNE UNIVERSITÉ

HABILITATION À DIRIGER DES RECHERCHES

Service Oriented Computing in Mobile Environments: Abstractions and Mechanisms for Interoperability and Composition

---

Nikolaos Georgantas

Submitted in total fulfillment of the requirements
for the degree of Habilitation à Diriger des Recherches

Defense on February 2, 2018

Commitee:

Rapporteurs:	Kurt Geihs	Professor, Universität Kassel
	Siobhán Clarke	Professor, Trinity College Dublin
	Daniela Grigori	Professor, Université Paris Dauphine
Examiners:	Khalil Drira	Research Director, LAAS-CNRS, Université de Toulouse
	Yolande Berbers	Professor, Katholieke Universiteit Leuven
	Laurence Duchien	Professor, Université de Lille 1 - sciences et technologies
	Pierre Sens	Professor, Sorbonne Université
	Valérie Issarny	Research Director, Inria Paris

Abstract

While mobile services incorporate and apply the fundamental principles of Service Oriented Architecture (SOA), they present a number of specifics that push certain challenges related to service oriented systems to their extreme and additionally introduce new unique research challenges. Such specifics relate to: (i) dynamism – open mobile environments are much more volatile than typical service environments, with services emerging and disappearing in arbitrary ways without prior notification; (ii) heterogeneity – a direct consequence of ad hoc mobile environments is that no safe assumption can be made about the technological and business features of the services encountered; (iii) awareness – in most mobile service applications, the business capabilities of services are not the only ones that matter, the multi-faceted context of services is equally important; and (iv) the equation among QoS expectations on services, scalability, and required resources is hard to solve, due to the resource constraints that are typical to mobile environments. Dealing with the identified specifics gets even more complex if we consider both traditional computing services and services attached to the physical world by means of sensors and actuators, i.e., Things.

In this habilitation thesis, I provide an overview and discussion of my main research results in these last years. In the context discussed above, I have focused on two principal aspects of service oriented computing in mobile environments: Interoperability and Composition. In my research work reported herein, Interoperability refers to the middleware layer, while Composition refers to the application layer. More precisely, service Composition relies on a common underlying middleware that supports Interoperability independently of the development and deployment internals of services (hardware platforms, operating systems, programming languages); however, when heterogeneous middleware is employed, Interoperability needs to be worked out also at this layer. Particularly in the open mobile environment reaching out to the whole Internet, these two aspects are closely interrelated. This calls for extensions to the classic SOA architectural style: services can be very diverse networked entities (e.g., Things, sensor-actuator networks, service feeds, data streams). Hence, stating this briefly, my research contributions have aimed at extending the SOA style with Interoperability and Composition enablers for mobile computing.

Acknowledgements

I would first like to thank my habilitation thesis committee members, both rapporteurs and examiners, Kurt Geihs, Siobhán Clarke, Daniela Grigori, Khalil Drira, Yolande Berbers, Laurence Duchien, Pierre Sens and Valérie Issarny, who did me the honor of being in my committee. I thank them for their evaluation of this work. I really appreciated the reports and the stimulating questions and following discussion at the defense.

Certainly, my special thanks goes to Valérie, who has been my mentor, director and now colleague and collaborator for all these years at Inria. Her guidance and support have played a major role in my research work and scientific evolution. Next to her, I would like to thank all the colleagues from the successive Inria ARLES and MiMove teams, fellow researchers, PhD students, engineers and interns that took active part in this research. The results that I am presenting herein are also their results. As the enabling factor of all this, I would like to acknowledge Inria, a wonderful place for research and personal fulfillment, and especially the research center of Rocquencourt that then became the one of Paris, where all this research work was carried out.

Last but above all, I owe my endless gratitude to Mélanie. You have stood by me with unfailing love, patience and support during this whole writing – and not only – even when I was getting impossible…

For Mélanie

Contents

1 Introduction **1**
1.1 Contributions 3

2 Mobile Service Discovery and Composition **5**
2.1 Service discovery based on service semantics 5
2.2 Service composition based on service behavior 12
2.3 Service composition with QoS optimization 17

3 Interoperability of Middleware Protocols **25**
3.1 Interoperability across interaction paradigms 25
3.2 Timed analysis of mobile interactions 30
3.3 QoS analysis of mobile interactions across interaction paradigms 34

4 Conclusion and Outlook **43**

Bibliography **44**

Chapter 1

Introduction

Given the prevalence of global networking and computing infrastructures (such as the Internet and the Cloud), mobile networking environments, powerful hand-held user devices, and physical-world sensing and actuation devices, the possibilities of mobile distributed systems have reached unprecedented levels. Such systems are dynamically composed from networked resources in the environment, which may span from the immediate neighborhood of the users – as advocated by pervasive computing – up to the entire globe – as envisioned by the current and Future Internet and one of its major constituents, the Internet of Things. Hence, truly ubiquitous computing is now getting tangible.

A computing paradigm that has proved particularly appropriate for ubiquitous and mobile computing systems is Service Oriented Computing (SOC) or Service Oriented Architecture (SOA) [116,117]. Therein, networked devices and their hosted applications are abstracted as services delivered and consumed on demand. The benefit of this approach lies in the loose coupling (in terms of implementation and hosting platform internals) of the components making up a system, hence the increased ability to make systems evolve when, e.g., requirements change or the computing & networking environment changes. The SOA approach, as, e.g., enabled by SOAP or REST Web services, is a convenient architectural style enabling integration of mobile services; the latter can be retrieved and composed dynamically thanks to service discovery and service interaction protocols.

While mobile services incorporate and apply the fundamental principles of SOA, they present a number of specifics that push certain challenges related to service oriented systems to their extreme and additionally introduce new unique research challenges. Such specifics relate to: (i) dynamism – open mobile environments are much more volatile than typical service environments, with services emerging and disappearing in arbitrary ways without prior notification; (ii) heterogeneity – a direct consequence of ad hoc mobile environments is that no safe assumption can be made about the technological and business features of the services encountered; (iii) awareness – in most mobile service applications, the business capabilities of services are not the only ones that matter, the multi-faceted context of services is equally important; and (iv) the equation among QoS expectations on services, scalability, and required resources is hard to solve, due to the resource constraints that are typical to mobile environments. Dealing with the identified specifics gets even more complex if we consider both traditional computing services and services attached to the physical world by means of sensors and actuators, i.e., Things.

In the following, we discuss research challenges related to SOC in the mobile environment [19, 36, 41, 42, 73, 85, 86]:

Service description and discovery. Service description is a fundamental element in SOA, as it determines the information that a service needs to expose to its environment for enabling its unambiguous identification and use. A variety of service description languages have been proposed to cover different service aspects and are currently in use. Some of them have reached the status of a standard and others are still research proposals. Such aspects include the service profile, that is, a high-level business description of a service, the interface, the behavior, that is, the observable supported execution patterns of the service in coordination with its environment, the QoS properties, and the service binding, that is, the information required for accessing the service at the underlying middleware protocol level. Besides employing an established syntax for describing these aspects, these languages may further make the semantics of the different aspects explicit by referring to a structured vocabulary of terms (ontology), which represents a specific area of knowledge. As already pointed out, mobile environments are typically

very dynamic and ad hoc, which calls for rich service descriptions, covering most of the above aspects, and their automated processing at runtime for enabling a realistic mobile SOA.

Additionally as raised above, mobile environments can be very heterogeneous, both locally but also when reaching out to the whole Internet of Services and Things, which aims to encompass all kinds of resources and present them as services. Service heterogeneity concerns both business semantics and communication middleware. The former issue has led to a wide use of ontologies and related technologies in SOA, as well as to extensive research in this area. As for the latter issue, it is due to the fact that mobile services may use different communication contexts. This calls for support for heterogeneous interaction styles, namely message-driven, event-driven, and data-driven styles. Different interaction styles apply to different needs; for instance, asynchronous, event-based publish/subscribe is more appropriate for highly dynamic environments with frequent disconnections of involved entities. This fact makes the various service bindings supported by most service description languages too stringent, since they comply with a single (client/server) message-based interaction style. Service description should be able to abstract and comprehensively specify the various service bindings of mobile services. This further implies extending the notion of service and introducing adequate service interaction modeling.

Heterogeneity further concerns available resources. Mobile services may be hosted on platforms that range from wireless resource-rich machines to wireless resource-constrained devices, and further to any physical object (Thing) enhanced with some networking capacity. Consequently, the need for automated runtime processing of service descriptions creates a trade-off between richness and efficiency given the resource constraints of mobile devices. This trade-off concerns the storage, publication, search, access, and reasoning about service descriptions, as enacted for the purposes of service discovery. In particular, it calls for advances in the expressiveness and processing efficiency of XML-based service description languages, as well as in the efficient encoding and reasoning about semantic annotations that are part of such languages.

On the other hand, service discovery & selection deals with matching stored advertisements to requests. Matching between service descriptions is based on syntactic, semantic and behavioral analyses. Despite existing work in the area, this is commonly not adapted to the specifics of mobile services, where resource constraints and user interactivity place inflexible efficiency and performance requirements. Related to this latter issue is also the need for efficient service repositories, where the organization of stored service descriptions and related indexing techniques can considerably accelerate the lookup of services. Additional challenges here are that such repositories should be maintained at runtime and they should also constantly reflect the extremely volatile population of available mobile services.

Service interaction and composition. As already pointed out, communication contexts of mobile services are characterized by high diversity of the middleware infrastructures with respect to employed interaction styles. Hence, as entities interacting in ad hoc settings cannot be assumed to share the same interaction style, mobile service access and composition are required to support heterogeneous styles and to enable interoperability among them. Despite a number of approaches dealing with interoperability between interaction styles or paradigms, provided solutions are in general ad hoc and concern specific cases. An overall solution to this issue is required for mobile services, based on appropriate modeling abstractions and transformation mappings between models. Moreover, a precise evaluation of such mappings is needed with respect to the preservation of semantics of the interconnected paradigms. Besides functional semantics, the impact on end-to-end QoS semantics is key here.

Composition in the mobile environment is further about realizing complex services and goals by combining available services. Networked system components should adapt at runtime their functional and non-functional behavior in order to be composed and interoperate with other components. Moreover, supporting composition and interoperation requires the definition of behavioral conformance relations to reason on the correctness of dynamically composed systems with respect to both functional and non-functional properties. With respect to the former, the composition must enforce selection of the appropriate component systems and coordination protocols that conform to the specification of the component systems. With respect to the latter, it is necessary to account for the QoS delivered by component systems and their integration. Specifically, the dynamic composition of mobile distributed systems must both minimize resource consumption on mobile nodes and satisfy the users' requirements with respect to perceived QoS. Proposed solutions must take into account the rich semantics and, at the same time, the resource constraints of mobile nodes, but also the user-interactivity of the mobile environment, thus employing efficient mechanisms for QoS-aware service composition.

1.1 Contributions

In this manuscript, I provide an overview and discussion of my main research results in these last years. In the context discussed above, I have focused on two principal aspects of service oriented computing in mobile environments: *Interoperability* and *Composition*. In my research work reported herein, Interoperability refers to the middleware layer, while Composition refers to the application layer. More precisely, service Composition relies on a common underlying middleware that supports Interoperability independently of the development and deployment internals of services (hardware platforms, operating systems, programming languages); however, when heterogeneous middleware is employed, Interoperability needs to be worked out also at this layer. Particularly in the open mobile environment reaching out to the whole Internet, these two aspects are closely interrelated. This calls for extensions to the classic SOA architectural style: services can be very diverse networked entities (e.g., Things, sensor-actuator networks, service feeds, data streams). Hence, stating this briefly, my research contributions have aimed at extending the SOA style with Interoperability and Composition enablers for mobile computing as outlined below.

Mobile service discovery and composition. Our work on service composition includes inevitably service discovery. We have introduced EASY, a solution to efficient semantic service discovery in mobile environments. EASY proposes a set of semantic conformance relations for service matching that take into account fine specificities of services and their descriptions. EASY matching addresses both functional and non-functional (QoS, context) service properties. Furthermore, EASY introduces two mechanisms for accelerating the computationally costly semantic reasoning within resource-constrained environments. First, it applies a numerical encoding scheme to ontologies and service descriptions, thus turning semantic reasoning into a series of arithmetic operations. Second, it indexes and organizes service descriptions inside a repository based on subsumption relations, thus reducing considerably the number of required semantic matching operations for resolving a service request.

By generalizing our service discovery solution, we have introduced COCOA, which discovers and selects a set of services for realizing a complex user goal in the mobile environment. The distinctive feature of COCOA is that it flexibly composes services with complex behaviors into a complex task expressing the user goal. Both services and tasks are high-level workflows of capabilities. COCOA ensures a correct composition that conforms to the user task specification and enforces valid consumption of the services. Furthermore, COCOA takes into account the QoS properties of services and the user's global QoS requirements regarding the whole task; it selects the service composition that optimally satisfies these requirements.

While QoS-aware service selection and composition in the two previous contributions results in a small number of combinations to compare, this is not generally the case. We have focused on the problem of QoS optimization in service composition, which is computationally NP-hard. Our proposed algorithm QASSA performs service selection in two steps, locally for each activity of a user task and globally for the whole task. For the local selection step, QASSA is inspired from multi-objective optimisation, which QASSA tightens or relaxes in certain aspects in order to select services that have, if possible, good levels for all QoS properties. Finally, in the global selection step, QASSA selects a set of near-optimal and interchangeable compositions; this enables late binding and adaptation in the uncertain mobile environment. QASSA achieves good optimality and excellent performance results, which makes it suitable for the resource-constrained mobile environment.

Interoperability of middleware protocols. Our work in this area has focused on the problem of interconnecting systems that employ different interaction styles or paradigms. We include in particular the client/server, publish/subscribe, data streaming and tuple space paradigms, which cover the majority of communication middleware platforms enabling system interaction. In our approach, we first study the semantics of these paradigms across the dimensions of space coupling, time coupling, concurrency, and synchronization coupling. In a second step, we introduce the Generic Middleware (GM) connector abstraction that represents in a unified way the common abstract semantics of the different paradigms; a set of programming interface and corresponding networked protocol primitives is established for GM. Based on this, we introduce a systematic approach for building interoperability software artifacts that enable interconnection of systems that employ different interaction paradigms; this allows the automated synthesis of such artifacts. Our interoperability solution is realized as a lightweight service bus, VSB, which targets the mobile IoT environment integrating both services and Things into complex applications.

In the same context, we have further studied the timed behavior of mobile interactions that follow the identified interaction paradigms. Our modeling includes time coupling semantics of the interaction, but also lifetime constraints of application data and user-related disconnection behavior of the receiving mobile application. Our abstractions enable a unified view on the different paradigms. Based on this, we build a formal model of the mobile interaction with timed automata. Our analysis results in a set of general conditions for successful interactions, relying on observable and potentially tunable system and environment parameters. In a second step, we perform statistical simulations of our timed model. These can be used along with the previous formal conditions by application designers in order to analyze quantitatively and possibly tune the tradeoff between success rates and latencies of mobile application interactions.

Going further with our modeling, we have enriched the previous (abstract) timed models with the features of the underlying mobile protocol infrastructure, including intermittent connectivity and varying reliability features, as well as with message queueing effects along the end-to-end interaction. We now employ queueing networks for our modeling. In particular, we have proposed an analytical solution to a queueing system with intermittent service and validated our model with data traces coming from real datasets. By using this queueing system model and others, we introduce performance modeling patterns covering the different interaction types supported by the identified interaction paradigms. We have developed a queueing network simulator that implements these patterns and have performed analysis of success rates and latencies similarly to our previous piece of work. The present models and simulations enable application designers to carry out a much finer analysis and tuning of mobile application interactions. In particular, performance modeling patterns can be combined to model and analyze end-to-end mobile interactions that rely on different interaction paradigms interconnected via VSB. This allows evaluating the end-to-end QoS of such interactions. This complements our interoperability solution with an assessment of its *effectiveness*, i.e., the potential effect for the application and/or the user's experience.

Remainder of the manuscript. The remainder of this manuscript is organized as follows. Chapter 2 presents the above outlined contributions with regard to service discovery and composition. It comprises three sections, which focus on semantic service discovery, service composition based on service behavior, and QoS optimization in service composition. Chapter 3 presents the contributions concerning middleware interoperability. Its three sections deal with interoperability across interaction paradigms, timed analysis of mobile interactions, and QoS analysis of mobile interactions across interaction paradigms. Finally, Chapter 4 concludes this manuscript and presents an outlook on ongoing and future work.

Chapter 2

Mobile Service Discovery and Composition

Service discovery and composition are key functionalities in SOA and the mobile service environment. I discuss in this chapter three solutions in this area, which are interrelated concerning the developed methods and also follow the same conceptual scheme.

In *Service discovery based on service semantics* (Section 2.1), a user of the mobile environment expresses her goal as a sought service capability. A capability provided by an available service is then discovered so that it optimally matches the requested capability in terms of functional and non-functional (QoS, context) properties.

In *Service composition based on service behavior* (Section 2.2), the mobile user expresses a more complex goal in the form of a task, a high-level workflow of requested capabilities. Available services may also be more complex: they provide several capabilities also coordinated by high-level workflows. By generalizing the approach of Section 2.1, a composition of services is produced so that: (i) it jointly provides the requested workflow of the user task, and (ii) it optimally matches the requested task in terms of capabilities and global (i.e., end-to-end for the composition) QoS properties.

Finally, in *Service composition with QoS optimization* (Section 2.3), the user task workflow is composed from simple services that do not expose workflows. Here, we employ the service matching solutions of the two previous sections and focus on the global QoS optimization problem for the service composition, which is computationally hard.

2.1 Service discovery based on service semantics

(Publications: [32, 33])
(Ph.D. thesis of Sonia Ben Mokhtar, UPMC, 2007 [29]. Co-supervision with Valérie Issarny.)

Service discovery is an essential functionality for establishing *ad hoc associations* between service providers and service requesters. In particular, *runtime* ad hoc associations are necessary in open and dynamic runtime environments, such as the mobile environment. There, users express interactive service requests. Moreover, software services and software clients (taking the role of service requesters on behalf of users) are designed, developed and deployed independently. During service discovery, semantics underlying service descriptions and service requests should be matched. A second step of syntactic mapping (i.e., in terms of data types and possibly service technologies) between the software interfaces of services and clients should follow. A flexible software client may also obtain from the service's interface the exact way of invoking the service, which may further be complex and expressed as a workflow of service operations. This step allows eventually having executable bindings. Nevertheless, the first step is the decisive one.

While semantic matching can be based on comparing the natural language vocabulary terms employed in the software interfaces of services and clients (commonly called syntactic matching in this case), this produces ambiguous results when there is no previous agreement on the use of terms; this is typically the case in open environments. A solution was proposed to this problem with the use of techniques coming from the knowledge representation domain. More specifically, the semantic Web [40] paradigm proposed to enrich published information with machine-interpretable semantics by referring to ontologies;

an ontology is a formally structured vocabulary of terms representing a specific area of knowledge. This is enabled by ontology languages, such as the Web Ontology Language (OWL) [12], which support formal descriptions and machine reasoning. Building on the semantic Web, the semantic Web services paradigm was introduced [106]. Following, numerous efforts investigated the automated service discovery of semantic Web services, but also their invocation, composition and execution monitoring [56, 102, 127, 132, 133]. Semantic service matching and discovery remains a very active field with a number of recent approaches [114, 124, 129, 130].

In this context, our contribution concerning service discovery has been principally twofold.

- First, we have proposed a set of conformance relations for semantic service matching between service descriptions and service requests, leading to optimal service selection. We have included in the elaboration of these relations extensive considerations concerning the specificities of services and of their semantic descriptions. Our complete service discovery solution, *EASY*, includes *EASY-L*, a language for semantic specification of functional and non-functional service properties, as well as *EASY-M*, a corresponding set of conformance relations that take into account both of these types of properties.

- Second, we have targeted the specificities of the mobile environment, in particular its high dynamism, resource-constraints and interactivity with the user. These call for resource-efficient and fast service discovery mechanisms. This, however, challenges the use of semantic service discovery: the underlying semantic reasoning is particularly costly in terms of computational resources. To deal with this challenge, we have included in EASY two mechanisms that achieve satisfying performance for semantic service discovery in the mobile environment.

Semantic service description. EASY-Language (EASY-L) supports the semantic, context- and QoS-aware specification of services as well as clients' requests. EASY-L is specified using OWL. Ontologies have conveniently been employed in the semantic specification of services, via a plethora of proposed languages such as OWL-S [57], WSMO [125], SAWSDL [92], FLOWS [80]. EASY-L captures the set of crosscutting concepts shared by these various languages with additional support for QoS and context properties of services. EASY-L can thus be seen as a 'meta-language' that can be easily extended and/or mapped to any of the above languages.

At the heart of EASY-L, we distinguish the notion of *capability*, which corresponds to the description of any functionality that may be advertised by a service or sought by a client. This description is given in terms of *inputs*, *outputs*, service *category* and (non-functional, QoS & context) *properties*. Inputs, outputs and category are defined with reference to one or more ontology concepts. Non-functional properties of services can be of two types: qualitative or quantitative. Qualitative properties are those described with reference to ontology concepts (e.g., security levels or system platform features), whereas quantitative properties are those that can be measured and assigned with numeric values (e.g., service latency, service cost, environment temperature). For the description of any one of them, we employ the operators *is-a*, *is-exactly-a* and *is-not-a*. For the description of quantitative properties, we use the operators *equal*, *not-equal*, *more-than*, *less-than*, *max-value-of*, *min-value-of*. Finally, the *and*, *or* and *not* operators are used in combination with the previous operators to build composite expressions. We defined our context and QoS models inside EASY-L by relying on the context ontology proposed by Preuveneers et al. in [121], and the QoS model proposed by Liu and Issarny in [101].

Specificities of semantic service matching. Based on EASY-L, we have introduced EASY-Matching (EASY-M), a set of conformance relations for matching services in terms of their functional and non-functional properties. Matching between services aims at comparing the suitability of advertised services against a service request. A number of research efforts have been conducted in the area of matching semantic Web services based on their signatures. Signature matching deals with the identification of subsumption relationships for the ontology concepts describing the inputs and outputs of service capabilities [143]. A base algorithm for service signature matching has been proposed in [115, 132]. Other solutions based on this matching algorithm have been proposed in the literature [68, 102, 133]. Some recent approaches also adopt, constrain or extend this algorithm with additional relations besides subsumption [124, 129, 130]. Furthermore, specification matching of software components or, more particularly, semantic Web services, has been studied in the literature [127, 131, 144]. Specification matching deals with matching pre- and post-conditions that describe the functional semantics of components.

Matching semantic Web services based on their signatures can be challenging because of the specificities of services and their descriptions. We raise the following points:

(1) The subsumption relation inside ontologies is an inheritance relation: an ontology concept subclass inherits all the properties of its parent class (similarly to class hierarchies in object-oriented programming languages). Let us consider a typical ontology example where a class `Car` has three subclasses: `Sedan`, `Station Wagon` and `Four-Wheel Drive` [115]. Each one of these subclasses has all the properties of `Car` and additional proper properties.

(2) A class is a collection of properties that describe a set of individuals, called the *class extension* in OWL [12]. A class has an *intensional meaning* (the underlying concept), which is related but not equal to its class extension. However, often a class is used or understood as its class extension. In our example, a service provider that sells cars is meant to either (i) providing individuals that have the properties of a car (intensional meaning), i.e., some cars, or (ii) providing the whole set of individuals described by this class (class extension), i.e., all cars. While the second case seems unrealistic, we can still assume that the service provider provides cars of all three subkinds identified in the ontology, i.e., `Sedan`, `Station Wagon` and `Four-Wheel Drive`. Correspondingly, a service requester that wishes to buy a car may either (i) be happy with any available car (no specific preferences), or (ii) would like to have the choice among all possible cars (again, choice among the three identified subkinds, to be realistic). The problem is that when an ontology class is employed in the description of service, typically it is not explicitly identified which of these two differing class semantics is meant. We have included explicit identification of the meant semantics in our approach.

(3) As mentioned above, the suitability of a provided service against a service request is evaluated by identifying subsumption relationships between the ontology concepts describing them. This includes the case where the concepts are the *same* (same intensional meaning) or *equivalent* (same class extension) [12]. The general rule is that a provided element should satisfy a required element, meaning that the former can be used there where the latter is requested even if they are not exactly the same. In our approach, the way of checking for subsumption depends on which of the two distinct class semantics identified in (2) is applied. We use some variants of the above example to illustrate this. In the case a class is used with its intensional meaning, a provider of only, e.g., `Sedan`s *(more specific)* can satisfy a requester of a `Car` *(more generic)* who has no specific preferences, since any `Sedan` is a `Car`. This is the matching relation that typically applies to objects and classes in object-oriented programming languages. Conversely in the case a class is used as its class extension, a provider of `Car`s *(more generic)* can satisfy a requester who looks for `Sedan`s *(more specific)*, since the set of `Car`s includes the one of `Sedan`s. The above conditions imply *strong satisfaction relations* between a provided and a required element. In a similar way, we cover also the cases where different class semantics are used for the provided and the required element.

(4) When checking for subsumption as introduced in (3), the sought relationships may not hold. For instance, in another variant of our example and for intensional class semantics, a provider of `Car`s *(more generic)* may or may not satisfy a requester of a `Sedan` *(more specific)*, because there may not be any `Sedan`s among the available `Car`s. In this case, a *weak satisfaction relation* is identified between a provided and a required element. While this was already introduced in [115], we have enriched it in our approach with the two distinct class semantics identified in (2). On the other hand, for both strong and weak satisfaction relations, the *degree of satisfaction* depends on how close the related concepts are in the class hierarchy. To illustrate this, we extend the ontology of our example: the class `Car` has now a parent class `Vehicle`. Assuming class extension semantics, a provider of `Vehicle`s can satisfy a requester who looks for `Sedan`s, but probably less well than a provider of `Car`s; mathematically, the set of `Vehicle`s includes the set of `Car`s which includes the one of `Sedan`s, but in practice a provider of `Vehicle`s should sell some kinds of `Car`s but probably not all kinds of `Sedan`s [115]. In our approach, we quantify the degree of satisfaction, for both strong and weak satisfaction relations. Moreover, while a strong satisfaction relation is in principle better than a weak one, we also consider the degree of satisfaction when comparing between them but as well in any matching decision.

(5) As discussed in (3) and (4), matching is performed between provided and required elements of services and service requests. The outputs of a service are typically related to what the service can

provide (provided elements); these are matched against the outputs specified in a service request (required elements). On the other hand, the inputs of a service may be related either to what a service needs in order to function properly (required elements) or to what a service can handle (provided elements); these are matched against the inputs specified in a service request (provided or required elements, respectively). In our approach, we explicitly identify the two cases for the semantics of inputs; this differentiates the way of applying the matching specified in (3) and (4).

Conformance relations for semantic service matching. We have taken into account Points (1) to (5) in the elaboration of EASY-M. EASY-M introduces three sets of relations.

The first set includes relations that evaluate qualitatively the level of matching between concepts or capabilities. More specifically, we introduce three relations for characterizing the matching between the inputs, outputs and categories of a requested capability *Req* and an advertised capability *Adv*: $ExactCapabilityMatch(Adv, Req)$, $InclusiveCapabilityMatch(Adv, Req)$, and $WeakCapabilityMatch(Adv, Req)$. In these three relations, we use the relation $ConceptMatch(c1, c2)$ to compare two concepts of an ontology. $ConceptMatch()$ identifies one of four matching levels:

i. *exact*, when the two concepts are the same or equivalent;

ii. *inclusive*, when the two concepts are not the same or equivalent, however the first concept can be used there where the second is required, as identified in Point (3) above; both (i) and (ii) correspond to a strong satisfaction relation;

iii. *weak*, when the first concept can possibly or partially substitute for the second, as identified in (4); this corresponds to a weak satisfaction relation;

iv. *fail*, when none of the above relations (i) to (iii) holds between the two concepts.

The $ExactCapabilityMatch()$ relation allows to find advertised capabilities that exactly match a requested capability, i.e., only exact matches between concepts are considered. The $InclusiveCapabilityMatch()$ allows to find capabilities that can as well be inclusive of the requested capability, i.e., the outputs and category of the advertised capability are inclusive of the outputs and category of the requested capability. More attention is paid to the inputs. As discussed in (5), if an input relates to what a service can handle, we also require here that the input of the advertised capability be inclusive of the input of the requested capability. If, on the other hand, an input relates to what a service requires, the inverse must hold: the input of the requested capability must be inclusive of the input of the advertised capability. Finally, the $WeakCapabilityMatch()$ relation is the least restrictive matching relation among the three relations: provided concepts of capabilities can also be in weak match with requested concepts. Note that $ExactCapabilityMatch()$ implies $InclusiveCapabilityMatch()$ implies $WeakCapabilityMatch()$. $ExactCapabilityMatch()$ is considered as the best match between a requested and an advertised capability. If $ExactCapabilityMatch()$ does not hold, we prefer in general $InclusiveCapabilityMatch()$ to $WeakCapabilityMatch()$. The former means that the advertised capability is able to fulfill what the requested capability asks for. Whereas, the latter means that the advertised capability may be able to fulfill or partially fulfill what the requested capability asks for. However, as pointed out in (4), the degree of fulfillment must also be considered in the matching decision. Hence, we introduce a second set of relations which evaluate more accurately the matching between capabilities.

This second set of relations evaluates quantitively the degree of matching between concepts or capabilities when a match holds, i.e., no fail result has been produced by $ConceptMatch()$. We introduce the function $CapabilityDegreeOfMatch(Adv, Req)$ for two capabilities, which is based on the function $ConceptDegreeOfMatch(c1, c2)$ between two concepts $c1$ and $c2$ in an ontology. $ConceptDegreeOfMatch()$ equals the number of hierarchy levels that separate $c1$ and $c2$ in the ontology. Then, $CapabilityDegreeOfMatch()$ is the weighted sum of the $ConceptDegreeOfMatch()$ values for each pair of matched concepts of the two capabilities. Different weights may be given to the two levels of match *inclusive* and *weak*, respectively, such that $w_{\mathrm{i}} <= w_{\mathrm{w}}$. A lower value of $CapabilityDegreeOfMatch()$ indicates a stronger match between capabilities.

Finally, the third set of relations evaluates quantitively the degree of matching between non-functional properties associated to capabilities, when all constraints set by required properties are met by provided properties. As supported by EASY-L, quantitative properties and constraints are expressed as equalities or inequalities (e.g., latency *less-than* 5, temperature *more-than* 20, price *less-than* 50, price *is-equal*

40), whereas qualitative properties are expressed in relation to concepts (e.g., NetworkConnectivity *is-exactly-a* BluetoothConnectivity, VirtualMachine *is-a* JVM8). To determine first if constraints are met, quantitative properties are evaluated numerically, while qualitative properties are checked using the relation $ConceptMatch()$. Regarding the latter, we accept any of *exact*, *inclusive* or *weak* level of matching. Since properties are heterogeneous – i.e., qualitative, quantitative, having different ranges of values, stronger with either bigger or smaller values – data normalization is needed in order to collectively evaluate their degree of matching. The first normalization that we introduce is assigning numeric values to qualitative properties. These values are given by the function $ConceptDegreeOfMatch()$. This allows evaluating quantitively a provided qualitative property with respect to a required property. Indeed, the smaller the $ConceptDegreeOfMatch()$ between a provided qualitative property and a required one is, the better. The second normalization that we apply is the *standard deviation normalization* on the various properties as in [101], where all properties become stronger with smaller values. We introduce the function $PropertiesDegreeOfMatch()$ for collectively evaluating the non-functional properties of a capability with respect to the required properties. $PropertiesDegreeOfMatch()$ is the weighted sum of the normalized values of the different properties, where we consider that lower values for the properties means a better match. Hence, a lower value of $PropertiesDegreeOfMatch()$ indicates a better match for non-functional properties.

$PropertiesDegreeOfMatch()$ reduces the problem of finding a good match with respect to non-functional properties to a *single-objective optimization* problem. A different approach would reduce the problem to a *multi-objective optimization problem (MOP)*. Rather than finding a single solution, the goal would be to find good compromises or trade-offs among multiple (possibly contradicting) objectives, resulting in a set of solutions often referred to as the *Pareto optimal set*. In Section 2.3, we tackle more specifically the problem of service composition with QoS optimization, where we have included MOP considerations.

Efficient semantic service discovery. EASY-M, and in particular the underlying $ConceptMatch()$ relation, relies on semantic reasoning on ontologies. OWL has its formal foundation in the family of Description Logics (DL) [15], which enables formal reasoning on OWL ontologies by using a DL-reasoner. Such reasoning allows inferring implicit relations between concepts from the explicit definitions of these concepts in an ontology. Subsumption is an essential relation inside an ontology. After complete subsumption reasoning on an ontology, the resulting concept hierarchy is referred to as the *classified ontology*. Semantic reasoning used to assess subsumption relations between concepts is computationally costly. To precisely evaluate this cost and its impact on semantic service matching and discovery, we have carried out a detailed experimental analysis. Specifically, we evaluate signature matching of service requests against service advertisements; this matching relies on identifying subsumption relations between the concepts involved in the service advertisements and the service requests. Note that we tested basic subsumption and not the more advanced EASY-M conformance relations, which nevertheless rely on the former. We employed 3 different DL reasoners to reason on concepts included in an OWL ontology of 99 classes. We employed a repository of Web services described in OWL-S [57]. OWL-S supports the specification of service functional capabilities similarly to EASY-L. We conducted two different kinds of experiments.

Our first experiment analyzes the cost of each step of the matching process, i.e.: (1) the time to parse the service advertisement and the service request; (2) the time for the reasoner to load and classify the ontologies involved in the service advertisement and request descriptions; and (3) the time to match the concepts involved in the advertisement and the request, i.e., to assess subsumption relations between these ontology concepts. In this experiment, the service request comprises 10 concepts. Results for all three reasoners of the total time of the matching process are in the order of 4-5 sec. Furthermore, for all three reasoners, the most expensive phase is the one of loading and classifying the involved ontology, taking 74-76% of the total time. The matching phase takes 19-21% of the total time.

Our second experiment measures the time taken by each reasoner to match the concepts involved in the service request and the service advertisement for an increasing number of concepts. We notice that this processing time increases significantly: it grows proportionally to the number of concepts and goes from 0.8-1 sec for 10 concepts in the service request to 1.2-1.6 sec for 14 concepts. In the case of matching a service request against all service advertisements of a service repository to select the best fitting service, this processing time will have to be multiplied by the number of available service advertisements.

From the above experiments, we conclude that semantic matching of service capabilities is a heavy process. It is obvious that the cost of semantic reasoning and matching is unsuitable for online use in the

interactive mobile environment. We have introduced a number of improvements to increase the efficiency of semantic matching. A first direct improvement is to perform the step of loading and classifying an ontology only once, offline. Later on, the classified ontology can be accessed several times online for reasoning on concepts. Then, we introduce two more improvements in the matching process:

1. We encode the classified ontology hierarchy, where each concept is assigned a unique numerical code. Furthermore, we assume that service advertisements and service requests are annotated with the codes corresponding to the concepts that they involve. Then, this reduces the semantic matching of concepts performed by $ConceptMatch()$ to a comparison of numerical codes, which can accelerate subsumption reasoning significantly.

2. We organize the potentially very big population of service advertisements within a service repository. In particular, we index and classify them according to their semantic similarity. This can reduce significantly the number of semantic matchings performed between the service advertisements and a service request, which allows efficiently resolving service requests but also efficiently advertising services.

Encoding semantic concept hierarchies. Encoding object class hierarchies with multiple inheritance for quickly performing subtype testing has been a very active field of research [49, 95, 147]. However, such encoding solutions rely on a closed world assumption, while encoding ontologies requires an open and scalable solution for potentially very large ontologies with support for conflict-free incremental encoding, so as to easily reuse previously encoded concepts. Moreover, the encoding should be as compact as possible and enable efficient matching. To deal with the above requirements, we applied a *prime number* based encoding technique for subsumption checking of classes in ontologies, which supports incremental encoding that avoids conflicts and a set of techniques for compaction. The adoption of this technique derived from our collaboration with the authors of [120].

Each concept is assigned a unique prime number as a *personal gene*. Unique prime numbers ensure conflict-free incremental encoding. Then, the encoding of a concept is produced as the multiplication of its personal gene and the personal genes of all its ancestors. A class A is subsumed by a class B if the gene of class B divides the encoding of class A. For incrementally encoding a new class, the next available prime number is used as a personal gene. The encoding of the new class can be computed without traversing the hierarchy to collect the genes of the ancestors, but by using the *least common multiple* function of the encoding of its parents. Targeting further more compact representations, this encoding technique is enriched with heuristics for minimizing the total encoding length of the hierarchy for all classes together or for minimizing the longest encoding of a single class in the hierarchy. These heuristics propose certain strategies for deciding the order of assigning prime numbers. Further heuristics are then proposed that can fast show when subsumption does not hold.

Organizing services in the repository. Several efforts towards efficient semantic service discovery have been proposed in the literature [59, 128, 130]. These efforts rely on indexing [59] or clustering [130] techniques to organize services in a repository; alternatively, they annotate services with pre-computed semantic reasoning information [128]. These approaches opt for overloading the service advertisement phase with costly computations in order to later achieve efficiency upon resolving service requests. In our solution, we aimed at achieving both lightweight service advertisement and lightweight request resolution, as service discovery in the mobile environment needs to be performed as well on resource constrained-devices.

At a pre-processing phase, our approach constructs directed acyclic graphs (DAGs) of capabilities of the advertised services. These graphs are indexed according to the ontologies being used in the capabilities that they contain. To construct a graph G, we employ the relations $ExactCapabilityMatch()$ and $InclusiveCapabilityMatch()$. Specifically, if $ExactCapabilityMatch(C1, C2)$ holds between two service capabilities $C1$ and $C2$, then these capabilities are represented by a single node in the graph. On the other hand, if $InclusiveCapabilityMatch(C1, C2)$ holds and $ExactCapabilityMatch(C1, C2)$ does not, $C1$ and $C2$ are represented by two distinct nodes with a directed edge from $C1$ to $C2$. Using this grouping technique, the most inclusive capabilities are represented by root nodes in the graph G, noted $Roots(G)$, i.e., nodes of G that do not have predecessors in G. These capabilities are said to be more inclusive than other capabilities contained in their subgraph because they provide outputs, inputs and category that *include* the respective ones of their successors in the graph. Similarly, we define $Leaves(G)$ as the set of

nodes in the graph G that do not have successors in G. The capabilities classified in the set $Leaves(G)$ of the graph G are the least inclusive capabilities of the graph, which means that they provide outputs, inputs and category that *are included by* the respected ones of their predecessors in the graph.

When a new service is registered with the repository, the set of capabilities that it provides are classified among the existing graphs. The algorithm of classifying a new capability into an existing graph is based on the following two properties that we have proved:

> (Prop 1) If $InclusiveCapabilityMatch()$ fails between a node $Root_{\text{i}}$ in $Roots(G)$ and Adv, it will also fail with all the successors of $Root_{\text{i}}$ in G, i.e., Adv will not have a predecessor in G.
>
> (Prop 2) If $InclusiveCapabilityMatch()$ fails between Adv and a node $Leaf_{\text{i}}$ in $Leaves(G)$, it will also fail with all the predecessors of $Leaf_{\text{i}}$, i.e., Adv will not have a successor in G.

(Prop 1) and (Prop 2) are used to check whether an advertised capability Adv will have a predecessor and/or a successor in the graph G without applying $InclusiveCapabilityMatch()$ with all the nodes of the graph. We briefly present in the following the algorithm for classifying the capabilities of a new service into a set of existing graphs. For a capability C advertised by the new service, a subset of graphs is preselected according to the ontologies used by C. For each graph G in the subset, the algorithm first checks whether C can be inserted in the subgraph of one of the root nodes of G. This is done by verifying if there exists a node $Root_{\text{i}}$ in $Roots(G)$ such that $InclusiveCapabilityMatch(Root_{\text{i}}, C)$ holds. If it holds, then C will have a predecessor in G. The next step is to find the first node, N, among the successors of the node $Root_{\text{i}}$, such that $InclusiveCapabilityMatch(Succ(N), C)$ fails, and to draw an edge from N to C. Moreover, C could have a successor in G. Thus, the algorithm tries to find among the set $Leaves(G)$ if there is a node $Leaf_{\text{i}}$ such that $InclusiveCapabilityMatch(C, Leaf_{\text{i}})$ holds. If it holds, then C will have a successor in G. The next step is to find the first node, M, among the predecessors of $Leaf_{\text{i}}$ such that $InclusiveCapabilityMatch(C, Pred(M))$ fails, and to draw an edge from C to M. On the other hand, if $InclusiveCapabilityMatch(Root_{\text{i}}, C)$ does not hold for any $Root_{\text{i}}$ in $Roots(G)$, C will not have a predecessor in G. Nevertheless, C could have a successor in G; this is checked as above.

We also briefly present next the algorithm for resolving a service request in a set of existing graphs. For a capability C sought by the service request, a subset of graphs is preselected according to the ontologies used by C. For each graph G in the subset, the algorithm performs matching between C and the nodes of $Roots(G)$. Specifically, if $WeakCapabilityMatch()$ does not hold between C and any node $Root_{\text{i}}$ in $Roots(G)$, the graph G is filtered out, and the next preselected graph is checked. This filtering is based on the fact that $WeakCapabilityMatch()$ holds between all nodes of G, and also it is transitive. Note that we use the $WeakCapabilityMatch()$ relation instead of the other two relations, i.e., $ExactCapabilityMatch()$ and $InclusiveCapabilityMatch()$, to filter out graphs, because it includes the other two relations. On the other hand, if $WeakCapabilityMatch()$ holds between C and a $Root_{\text{i}}$, the algorithm tries to find the node in the subgraph of $Root_{\text{i}}$ that minimizes $CapabilityDegreeOfMatch()$ with C and meets all the required non-functional properties of C. By performing the above for all matching $Root_{\text{i}}$ of G and for all pre-selected graphs, the node in the repository that best matches C is selected. Finally, selection among the semantically equivalent capabilities – if more than one – represented by the identified node is based on the $PropertiesDegreeOfMatch()$ relation. Alternatively, we may apply from the beginning a selection that takes into account both functional and non-functional properties in a combined way. In this case, we seek to minimize the weighted sum of $CapabilityDegreeOfMatch()$ and $PropertiesDegreeOfMatch()$ for each examined capability of each node.

Our solution enables resource-efficient resolution of a service request inside the repository, considerably reducing the number of matchings performed for this purpose with advertisements. Our solution further enables resource-efficient insertion of a new service advertisement in the repository, considerably reducing as well the number of matchings performed for this purpose with advertisements already registered.

Prototype implementation and performance evaluation. We have implemented a prototype for evaluating the performance of our semantic service matching and service discovery solution. In all the experiments that we performed, we increased the number of services from 10 to 100. The service descriptions given in EASY-L are using 22 different ontologies, and each service description contains a single provided capability. Based on their descriptions, services are grouped in 12 groups of various sizes.

Our first experiment evaluates the time to create graphs of services in an empty repository. We take two measurements: (1) the time to parse the service descriptions; (2) the time to organize the service

capabilities into graphs. We observe that the time to create the graphs (in the order of few ms) is negligible compared to the time to parse service descriptions (in the order of few hundreds of ms), i.e., the XML parsing time, which is mandatory due to the use of Web services and semantic Web technologies.

Our second experiment evaluates the time to insert a new service advertisement into a repository. We take two measurements: (1) the time to parse the EASY-L description of the new service; (2) the time to classify the service capabilities within the repository graphs. Results show that the time to classify service capabilities into a set of existing graphs (few ms) is negligible compared to the XML parsing time of the service description (around 150 ms). We also observe that this time is nearly constant. This is due to the fact that the number of semantic matchings performed in the repository in order to insert a capability does not depend on the total number of services in the repository. The number of semantic matchings depends: on the number of root and leaf nodes in the repository graphs that are preselected as relevant for the capability; as well as on the number of capabilities contained in the graph in which the capability will be inserted. Only few semantic matchings are needed thanks to our efficient organization of the repository.

Our third experiment evaluates the time to match a service request with services hosted by a repository. Furthermore, we compare the time to match a request in an organized repository with the time to match a request in an unorganized repository. We observe that without repository organization, the average overhead for matching is around 50% of the time to match when the repository is organized. Moreover, we notice that the time to match a request in the organized repository is nearly constant: a request does not have to be matched with all the services of the repository. We also remark that the response time to match a required capability, excluding XML parsing time, is in the order of a few milliseconds.

The last experiment that we performed is a comparison of the response time given by classic syntactic matching and our efficient semantic matching performed by EASY. We observe that while both response times are at the same level for 10 services (around 150 ms), the response time given by syntactic matching is increasing with the number of services available in the repository, while our semantic matching has an almost stable response time. This is again thanks to the efficient organization of the semantic repository.

We point out additionally that in all our experiments above, semantic matching is also significantly accelerated by the numerical encoding of ontologies and service descriptions. Performance results concerning the solution we applied for the encoding of ontologies are discussed in detail in [120].

2.2 Service composition based on service behavior

(Publications: [30, 31, 110])

(Ph.D. thesis of Sonia Ben Mokhtar, UPMC, 2007 [29]. Co-supervision with Valérie Issarny.)

Service composition enables realizing complex functionalities that are not provided by a single service. Service composition relies on service discovery and can be seen as a generalization of the latter: it results in establishing ad hoc associations between a service requester and a *set* of service providers that can jointly satisfy the service request. In the mobile environment in particular, users' interactive service requests expressing their goals are potentially complex tasks to be realized by integrating on-the-fly suitable services. A composition of services can be represented by a workflow or *conversation* coordinating a set of capabilities from the involved services by employing control constructs such as Sequence, Parallel Execution, Choice, etc. A service participating in a composition may itself be complex, providing several capabilities that are coordinated by a workflow proper to the service. In this case, the service composition workflow must comply with the service workflow.

Numerous research efforts have aimed at providing solutions to service composition. The proposed service composition algorithms rely on the descriptions of requested user tasks and provided services. In these efforts, a target user task is often concisely described as a requested capability (or alternatively a user's goal) and a service as a set of one or more provided capabilities. For instance in service chaining, including forward chaining and backward chaining, individual service capabilities are combined with each other based on the conformance of their signatures. The objective of this combination is to obtain a composite capability that conforms to the signature specification of the target user task capability (or that resolves the user's goal) [53,105,124]. This composition approach allows a certain flexibility in the service composition with regard to the capabilities that are eventually included, which can be advantageous in the dynamic mobile environment [53]. However, this approach involves a degree of uncertainty regarding the way service capabilities are combined. The resulting workflow may: (i) not be exactly what the user

had in mind, and/or (ii) certain workflows of involved complex services may not be respected. A more precise composition is possible when more expressive descriptions are available for the user task and the services, specifying, respectively, the target task conversation and the service conversations to conform to. We note also an interesting approach that lies between the two above ones: it relies on a more abstract description of a task based on user's intentions, which is refined at runtime together with its concrete realization [69].

In this context, we have introduced *COCOA*, a solution to service composition for the mobile environment. Our service composition relies on the EASY service discovery of the previous section. In COCOA, a user's potentially complex goal is expressed as a user task that takes the form of a workflow of requested capabilities. These are high-level abstract capabilities, meaning that their exact realization is determined later, when these are bound to concrete capabilities provided by services. Equally, we assume that services available in the mobile environment can be complex and are also expressed as workflows of capabilities. These are also high-level capabilities, which are nevertheless concrete: they may correspond to atomic service operations or may decompose into workflows of operations. The workflows inside service capabilities are transparent to COCOA; they need to be obtained following the service composition for realizing executable bindings between the user's software client that will orchestrate the service composition and the services. We introduce the COCOA-L language to describe services and tasks. COCOA-L relies on EASY-L for the semantic, QoS-aware specification of services and extends it for equally specifying tasks. Furthermore, COCOA-L relies on OWL-S [57] for the specification of services' conversations and – by extension – tasks' conversations. Our service composition approach employs the EASY-M matching relations to select services but also – by generalizing – optimal service compositions that best match the capabilities and meet the QoS properties requested by the user task. Hence, COCOA inherits from EASY its powerful semantic and QoS features. Additionally and most importantly, our service composition selects services and composes their capabilities and possibly workflows so that the resulting service compositions provide the workflow of capabilities and the QoS properties requested by the user task. Different composition schemes may apply: the same user task may be realized by binding to a single service, by composing individual service capabilities, by composing fragments of service conversations, or finally by interleaving fragments of service conversations. Hence, composition is adaptive according to the specifics of the mobile environment in terms of available services and their conversations. Moreover, COCOA enforces a valid consumption of the composed services, ensuring that their conversations are fulfilled. The above are supported by two mechanisms: *COCOA-SE* for service selection and *COCOA-CI* for conversation integration.

Description of services and tasks as workflows We have specified COCOA-L by using the Web Ontology Language (OWL) [12] and, as already mentioned, by mixing EASY-L and OWL-S [57] features.

OWL-S is a Web service ontology specified in OWL for describing semantic Web services. In OWL-S, the behavior of a complex service is described as a process. This process is decomposed into a set of sub-processes coordinated by a set of control constructs. The sub-processes can be either composite or atomic. Composite processes are decomposable into other composite or atomic processes, while atomic ones correspond to WSDL operations. We rely on this feature of OWL-S for describing the conversations of tasks and services as workflows of abstract and concrete capabilities, respectively.

COCOA-L identifies further two types of dependencies among the capabilities that participate in a service or task conversation. If two capabilities are executed one following the other in a service or task conversation, we define an *ordering dependency* between these two capabilities. For enforcing a valid consumption of the composed services, the dynamic realization of a user task has to fulfill service ordering dependencies. Furthermore, we define an *affiliation dependency* between two requested capabilities in a task conversation when they must provided by the same service. We take into account both of these types of dependencies when composing services.

To support our service composition approach, COCOA-L additionally introduces formal descriptions for service and task conversations based on finite state automata. Other approaches of formalizing Web services conversations and their composition have been proposed in the literature, based on Petri nets [135], process algebras [93], finite state machines [70] or graph-based representations [79]. We have introduced a set of mapping rules for translating an OWL-S process to a finite state automaton. In our mapping, automata symbols correspond to capabilities. Each control construct involved in a conversation is mapped to an automaton using our set of rules. Then, these automata are linked together in order to build a global automaton. Further details about our modeling of OWL-S processes as automata can be

found in [111].

For specifying QoS properties, COCOA-L relies on EASY-L for services' provided QoS and extends it properly for tasks' required QoS. More specifically, services specify the QoS properties of their provided capabilities; tasks may specify both *local QoS constraints* for their required capabilities and *global QoS constraints* for the entire task workflow. Local QoS constraints have to be satisfied by the capabilities of the selected services, whereas global QoS constraints have to be satisfied by the resulting service composition. To support QoS evaluation of a service composition, COCOA-L additionally introduces QoS formulae corresponding to each QoS metric. A number of research efforts propose reduction rules to compute the QoS of a workflow; reduction is performed on each workflow construct taking into account the nesting of constructs [48, 109, 146]. We used the model proposed by Cardoso et al. [48] to extract the formula of each QoS dimension corresponding to the task automaton's structure. While evaluating the QoS of a service composition, we provide two possible estimations for each QoS dimension: (1) a history-based, probabilistic estimation; and (2) a pessimistic estimation. The former corresponds to an average estimation, while the latter corresponds to a worst case estimation. The choice between these two estimations depends on the global QoS constraint (deterministic or probabilistic) expressed in the user task. If the user demands a deterministic QoS, our approach compares the requested QoS with the pessimistic estimation of the composite service. If the user requires an average QoS, the latter is compared against the probabilistic estimation.

Service selection based on service behavior. Based on COCOA-L descriptions of services and tasks, COCOA-SE selects services towards the realization of user tasks in the mobile environment. We assume that available services publish their descriptions in a structured (by applying our EASY techniques) or unstructured repository.

In a first step, we identify a set of services that provide semantically matching capabilities and meet local QoS constraints with respect to the task's requested capabilities. We do not select best matching services at this stage. For this step, we employ the EASY-M matching relations. Regarding functional properties of capabilities, we use $WeakCapabilityMatch()$, which enables selecting capabilities with an exact, inclusive or weak match. For instance in the case of an EASY repository, such a capability request will return the services included in a whole graph, provided that a root of this graph matches the request. On the other hand, to determine if local QoS constraints are met, quantitative properties are evaluated numerically, while qualitative properties are checked using the relation $ConceptMatch()$. In the latter case, we may accept, besides exact and inclusive, also a weak level of matching if we want to be more flexible.

In a second step, we filter the selected set of services based on the ordering dependencies inherent in their conversation specification compared against those of the user task. For instance, a service that provides a capability that matches with a requested capability of the user task could not be useful to the composition if, inside the service conversation, this capability has ordering dependencies with other capabilities that are not requested at all in the user task. Similarly, if two capabilities A and B are ordered as $< A, B >$ in the task conversation, while their matching capabilities A', B' are ordered as $< B', A' >$ in a service conversation, then this service should be filtered out. To allow for more flexibility, we enable alternative specifications of a task conversation where both $< A, B >$ and $< B, A >$ are possible if there is not a substantial ordering dependency between A and B. More specifically, we generate from the task conversation a *flexible task automaton* that contains all the rescheduling possibilities of the former. Furthermore, we add to the flexible task automaton empty transitions that connect each state with any other state on the same acyclic path; additionally, any state after the initial state can be final. The resulting *filtering automaton* can then be compared with the automaton of a service selected in the previous step to check whether there exists an intersection between the languages generated by the two automata. This enables selecting services that meet the ordering dependencies of the user task while allowing potential interleaving of their conversations. Furthermore, if an affiliation dependency is specified between two capabilities of the user task, only services that provide both these capabilities in their conversation are kept from the previously selected services.

Service conversation integration. Once service selection has been performed in the previous steps, COCOA-CI integrates the conversations of the selected services to realize the conversation of the user task. Integration is based on the finite state automata associated to the conversations of the services and the task.

A number of research efforts have aimed at conversation-based service composition. In [79], service behavioral matching is reduced to a subgraph isomorphism problem. In [38, 75, 78, 84, 140], service behaviors are modeled as finite-state automata, while in [126] they are modeled as nondeterministic finite automata. On the other hand, Petri nets are employed in [103]. In all composition approaches, the objective is to determine a composition that satisfies all the constraints that are formally expressed by the models of the target composite service (or task) and the candidate services. This has also been our objective in COCOA.

COCOA-CI first integrates all the automata of the selected services in one global automaton. The global automaton contains a new start state and empty transitions that connect this state with the start states of all selected automata. The automaton also contains other empty transitions that connect the final states of each selected automaton with the new start state. The next step of COCOA-CI is to parse each state of the task automaton starting with its start state and following its transitions. Simultaneously, a parsing of the global automaton is carried out in order to find for each state of the task automaton a state of the global automaton that can *simulate* it. More specifically, a task automaton state is simulated by a global automaton state when for each next-transition of the former there is at least one semantically matching next-transition of the latter. In particular, COCOA-CI compares the labels of the next transitions of the states. These labels correspond to capabilities. Semantically matching transition labels or capabilities between the services and the user task have already been identified at the service selection step by COCOA-SE. COCOA-CI allows finding service compositions with possible interleaving of conversations of the involved services. This is done by managing *service sessions*. A service session characterizes the parsing stage of a service conversation by COCOA-CI. A session is opened when a service conversation starts and ends when this conversation finishes. Several sessions with several services can be open at the same time. This allows interleaving the interactions of the task with distinct services. More specifically, a session opened with one service can remain open and temporary inactive during parsing of the interaction of the task with another service. An important condition that has to be observed when managing sessions is that each opened session must be eventually closed, i.e., it must arrive to a final state of the service automaton. During the composition process, various paths in the global automaton, which represent intermediate compositions, are investigated. Some of these paths will be rejected during the composition, while some other will be kept (e.g., if a path involves a service in which a session has been opened but never closed, this path will be rejected).

While parsing the task and global automata, in addition to checking for each state the semantic matching between next-transition labels or capabilities, conformance to the global QoS constraints of the user task is checked. This is done by using the QoS formulae that have been extracted from the task automaton's structure as supported by COCOA-L. Taking the QoS formula for each QoS dimension, we initially assume that each service capability will provide the best value of the considered QoS dimension (for example, latency = 0, availability = 1); this removes the effect of the specific capability. Then, each time we examine a capability provided by a service, we replace the corresponding best value in the formula of the considered dimension with the real QoS value of the capability. This allows evaluating in each step of the integration the values of all QoS dimensions up to that step and provided that the capability examined in that step is selected. These values are then compared to the global QoS constraints set by the user task; if the constraints are not met, the path in the global automaton that includes the capability in question is rejected. COCOA-CI provides as output a set of sub-automata from the global automaton that conform to the task automaton's structure. Each of these automata is a composition of services that conforms to the conversation of the target user task, further enforcing valid service consumption.

Once the set of possible compositions is produced, the last selection stage is to choose the best among the resulting compositions, on the basis of provided QoS. For this, we rely on the EASY-M conformance relations and extend them for service compositions. More specifically, we seek to minimize either (i) first $CapabilityDegreeOfMatch()$ and then $PropertiesDegreeOfMatch()$ or (ii) their weighted sum at the same time. $CapabilityDegreeOfMatch()$ is calculated for each service composition as the sum of the corresponding $CapabilityDegreeOfMatch()$ values of the capabilities involved in the composition. $PropertiesDegreeOfMatch()$ is calculated for each service composition from the global QoS values calculated for each QoS property with the QoS formulae extracted from the task automaton's structure as discussed above. In the first case, we first look for the composition that best matches functionally the user task, and then, if there are equivalent compositions, we select among them the one that offers the best QoS (by construction all selected compositions meet the global QoS constraints of the task). In the second case, we look for the composition that best satisfies both functional and QoS properties of the task in a combined way, where the applied weights can express a preference between functional and

QoS properties. Finally, we note that we apply the same data normalization as in the previous section to deal with the heterogeneity of properties, i.e., qualitative, quantitative, having different ranges of values, stronger with either bigger or smaller values. This allows collectively evaluating their degree of matching.

The last step is to produce an executable user task from the winning service composition. For this, the conversation description of the user task is complemented with information coming from the composed services. Specifically, each abstract capability of the user task is replaced by the corresponding concrete capability of a service. This capability may correspond to either an atomic service operation or a workflow of operations. Furthermore, a grounding description for the user task, which contains the binding information of the composed services is generated. The complemented task description and the generated grounding are sent to the user's software client (e.g., an execution engine) that performs the user task by orchestrating the appropriate services.

Prototype implementation and performance evaluation COCOA-SE relies on the EASY-M conformance relations for the semantic matching of capabilities, while additionally it filters services based on the dependencies inherent in service and task conversations. The main computational cost here comes from the former, which we evaluated in the previous section. We now present our evaluation of COCOA-CI, which is at the heart of the composition process, as well as the impact of supporting QoS awareness.

The performance of COCOA-CI is proportional to the complexity of the task's and services' conversations. Specifically, the response time of the algorithm is proportional to the number of possible (intermediate) composition paths investigated during the execution of the algorithm. There are two main factors contributing to the increase of the intermediate composition paths: (1) the number of capabilities provided by the services that semantically match the capabilities requested by the task; and (2) the number of capabilities requested by the task. We have carried out two experiments, each one evaluating the impact of one of these two factors on the performance of COCOA-CI.

In the first experiment, we increase from 10 to 100 the number of capabilities provided by the services; all are semantically equivalent. Two cases for the user task are considered: one where the task is composed of a single capability, and one where it is composed of 5 capabilities in sequence; in both cases, the capabilities are semantically equivalent among them and with the capabilities provided by the services. We compare the performance of COCOA-CI with the XML parsing of the services' and task's conversation descriptions, which is inherent in the use of Web services and semantic Web technologies. Our results show that the cost of our algorithm (up to 300 ms, for 100 service capabilities and 5 task capabilities) is low compared to the XML parsing time (up to 600 ms in the same case). Moreover, it increases almost linearly with the number of service capabilities.

In the second experiment, we fix the number of capabilities provided by the services to the worst case coming from the previous experiment, i.e., 100 capabilities fitting the task, while the number of requested capabilities, structured in sequence in the task's conversation, is increasing from 1 to 20. We compare again the performance of COCOA-CI with the XML parsing of the services' and task's conversation descriptions. We note that this is an extreme scenario for our algorithm, as each capability requested in the task's conversation is matched against 100 capabilities, and the resulting number of possible compositions is 100^{nb}, where nb is the number of capabilities requested in the task's conversation. We observe that for a number of possible compositions less than 100^{10}, our algorithm takes less time than the XML parsing time (around 600 ms).

In realistic cases, both the user task and the services will contain various capabilities organized using various workflow constructs, thus leading to the decrease of possible resulting compositions. Consequently, the response time will be acceptable for the interactive mobile environment. Indeed, we have applied our algorithm in a real case example, in which the task's conversation contains 20 requested capabilities and the selected services provide 30 capabilities, including various control constructs (e.g., Sequence, Choice, Loop). In spite of the large number of capabilities requested in the task's conversation, the algorithm spent only 32 ms to find the 2 resulting compositions among 36 intermediate compositions, against 152 ms of XML parsing time.

Finally, we performed a variation of our second experiment, where we introduce consideration of QoS in our integration algorithm and study its impact. This produces a small increase in the XML parsing time (of around 20 ms), which is due to the addition of XML tags for describing QoS. At the same time, this results in a considerable decrease of the execution time of our algorithm (350 ms for a task of 10 capabilities compared to 600 ms above). This is attributed to the rejection of a number of paths that do not fulfill the QoS requirements of the user task during the integration of service conversations.

2.3 Service composition with QoS optimization

(Publications: [26–28])
(Ph.D. thesis of Nebil Ben Mabrouk, UPMC, 2012 [25]. Co-supervision with Valérie Issarny.)

In the previous section, we presented our approach to service composition, where, based on a user's goal expressed as a task, we discover and compose service capabilities and workflows so that the resulting compositions provide the workflow of capabilities and meet the global QoS constraints requested by the user task. As a final step, we select the best service composition, which optimizes separately or in a combined way the degree of match to the capabilities of the user task and the global QoS properties. The optimization problem there is simple, as the number of resulting service compositions is in general low due to the previous selection of services based on the composition of their workflows: we just compare exhaustively all resulting compositions.

We have focused further on the QoS optimization problem in service composition. Similarly to the previous section, service composition is performed for realizing a user task, which is a workflow of *activities* (equivalent to requested capabilities of the previous section). The optimization problem is related to the selection of services, among those that match one or more activities of the user task, so that the global QoS constraints requested by the user task are satisfied and the global QoS properties of the composition are optimized. We assume here that each service provides high-level capabilities that a user task may invoke individually or for performing activities with *affiliation dependencies*[1] between them. In the case of affiliation dependencies between user task activities, suitable services provide matching capabilities in a simple sequence workflow with the same *ordering dependencies* as the activities in the task; otherwise, there is no service workflow. Hence, the number of possible service compositions can potentially be high and the resulting combinatorial problem has a high computational cost. This is even more challenging in the mobile environment due to its dynamism, limited computational resources, and timeliness constraints when interacting with the user. In particular, it has been shown in the literature that the optimization problem in question is equivalent to a Multiple choice Multiple dimension Knapsack Problem (MMKP), which is NP-hard [10]. QoS-aware service selection algorithms fall under two broad classes with respect to their selection techniques. On the one hand, local selection (i.e., greedy selection) proceeds by selecting the best service in terms of QoS for each activity in the user task separately. This technique has a low computational cost but it cannot guarantee meeting global QoS requirements. On the other hand, global selection covers the scope of the whole composition and ensures meeting global QoS requirements. However, it is of high computational complexity.

In this context, we have introduced QASSA, a fast and resource-efficient service selection algorithm that provides the ground for QoS-aware service composition in mobile environments. QASSA combines local and global selection techniques by proceeding in two steps: (1) the local selection step aims at selecting services with the highest QoS for each activity in the user task, and (2) the global selection step aims at selecting near-optimal compositions of services resulting from the local selection.

A number of efforts have combined local and global selection techniques for determining optimal service compositions with respect to their global QoS. Some of those apply first local and then global selection, similarly to QASSA [6, 54, 58, 100, 123], while others invert the order of the two steps [5, 88, 99]. Furthermore, the approaches introduced in [6] and [54] employ multi-objective optimization in their local selection or in both the local and the global selection, respectively; in our approach we also rely on multi-objective optimization, but we extend it in certain ways, as we describe in the following.

QASSA further selects several alternative near-optimal compositions. Indeed, selecting only one service composition brings about several shortcomings such as the lack of choices for the user, the overload of hot services (i.e., services with high QoS) [87], and the lack of adaptation support [2] (i.e., deferred final selection and dynamic binding at run-time). More specifically, our main purpose in QASSA is to select alternative services (based on their advertised QoS) for each activity of the user task that can produce several near-optimal compositions, rather than identifying (as close as possible) the optimal service composition. This allows identifying a small set of services that can possibly be monitored for their runtime QoS, thus allowing to cope with the potential differences between advertised and runtime QoS of services. Based on the latter, the final selection of services can be performed at runtime, leading to late service binding, just before the corresponding task activity needs to be executed [55]. Additionally, selecting alternative services and service compositions allows substituting a service by another one in case of a service failure and subsequent need for runtime reconfiguration.

[1] Affiliation dependencies and ordering dependencies have here the same meanings as in Section 2.2.

Local Selection Phase. Most QoS-aware service selection approaches use utility functions to evaluate the overall QoS of a service [54]. Such functions generally: (i) apply data normalization to QoS values of single properties, (ii) set weights to the corresponding properties based on user preferences, then (iii) sum up the properties multiplied by the weights. Thus, they reduce QoS-aware service selection to a single-objective optimization problem. Applying such a utility function includes the risk of unnoticeably compensating bad values of one or more QoS properties by good values of other properties. Hence, this may lead to selecting services with some bad QoS properties. Further guidance to the service selection can be provided by setting (e.g., user-defined) constraints for the service QoS properties. Then, before applying the utility function, services are filtered based on these constraints. This guarantees that the finally selected services with the highest overall QoS satisfy the user constraints for all the QoS properties. In this case, having some QoS properties that are 'less good' than other QoS properties is not anymore a problem. We adopted this approach in the EASY service selection of Section 2.1.

In the local selection phase of QASSA, there are no local QoS constraints to guide the selection (users express only global QoS constraints). Hence, our aim is to identify per activity services that have – as far as possible – good levels for all QoS properties. These services can most likely produce near-optimal service compositions in the global selection phase. Accordingly, for our approach concerning the local selection phase, we are inspired from *multi-objective optimisation*. Multi-objective optimisation aims at identifying the *Pareto-optimal set* of solutions, where each solution optimizes one or more objectives. These solutions are called *non-dominated*, meaning that, for each one of them, there exists no other solution that improves at least one objective without degrading at least one other objective. Each Pareto-optimal solution expresses a specific trade-off among the multiple objectives. Determining the Pareto-optimal set is typically exponential in the size of the problem instance [136]. Therefore, resolving multi-objective optimisation is often reduced to identifying a good approximation of the Pareto-optimal set, or *Pareto set approximation* for short.

Evolutionary multi-objective optimization (EMO) applies evolutionary algorithms to finding a Pareto set approximation. In EMO, multi-objective optimisation is defined as a set problem, where the search space consists of all potential Pareto set approximations rather than single solutions, i.e., the search space is a set of sets [148]. To fully specify this set problem, an appropriate order on the search space is required; this is defined as a *set preference relation*. Then, the search relies on this set preference relation: for any two Pareto set approximations, it says whether one set is better than the other or not. The set preference relation can be defined in terms of a *quality indicator*. The quality indicator is a function that assigns to each solution set a scalar value reflecting its quality according to the optimisation objectives, i.e., a fitness function defined over sets.

Our local selection approach relies on the above concepts. Nevertheless as stated above, we are looking for services per activity that have, if possible, good levels for all QoS properties. Hence, we tighten here the search for a Pareto set approximation and look for Pareto-optimal solutions that have good levels for the highest possible number of QoS properties. Additionally, we are looking for multiple equivalent – as far as possible – services per activity, i.e., services with similar levels of QoS properties, which can substitute for each other in case of late service binding or runtime reconfiguration of the selected service composition. Hence, we relax here the search for a Pareto set approximation and allow solutions that may be dominated by other solutions with similar levels of QoS properties inside the resulting set; the former solutions provide the sought redundancy.

More specifically, we define *QoS Class* to be an element in the search space. A QoS class represents a set of services having roughly the same QoS and reflecting the same trade-off between QoS properties. To determine QoS Classes among the service candidates associated with an activity, we employ clustering techniques, notably the *k-means* algorithm. We group service candidates per activity into clusters according to their QoS values. The advantage of clustering is that it enables determining the set of services providing good levels of QoS dynamically, depending on the density of available services and the level of QoS they offer. Conversely, use of a static criterion could lead to either discarding services with good QoS that can potentially fulfill the user task, or investigating services with bad QoS in the next steps of the selection. Moreover, establishing several clusters of services representing different QoS levels allows for flexible selection in the next steps. In our case, we consider only services belonging to the best QoS cluster. In this way, we considerably reduce the number of service compositions to be investigated in the global selection phase, hence achieving good timeliness for that phase. This is without much loss in the optimality of the resulting composition, since we consider the services with the best QoS. If higher optimality is sought, the next best QoS cluster(s) can be also included in the global selection, which results however in lower timeliness. Finally, we have chosen k-means because it is a simple algorithm with low

computational cost; thus, using it as a preliminary selection phase does not increase the complexity of the problem. On the contrary, k-means reduces the overall computational cost of QASSA, since it prunes the number of service compositions to be investigated in the global selection phase, especially since our method of using k-means is highly selective.

Typically k-means clustering is based on the n-dimensional Euclidean distance. In our problem, each service is represented by an n-dimensional vector containing its normalized values for each QoS property[2]. In order to have a more fine-grained clustering, representative of the different levels of each QoS property and without the risk of compensation among QoS properties, we perform instead *one-dimensional clustering applied n times* (once per QoS property). That is, we cluster service candidates (per activity) for each QoS property separately. To provide a concrete example, we may cluster services into 3 clusters per QoS property, corresponding respectively to 'bad', 'medium' and 'good' quality levels for the specific QoS property; another number for the clusters may be equally chosen. We employ the *k-means++* [11] algorithm as initialization method for our k-means algorithm. k-means++ determines initial values for the cluster centers before one can proceed with the standard k-means heuristic iterations. It has been shown that the application of the k-means++ initialization improves the speed and optimality of k-means.

We define *QoS Level* to characterize the quality level of the clusters across all QoS properties (e.g., bad, medium or good, in the example above). Hence, a service has a specific QoS Level for each one of its QoS properties. We define further a QoS Class to be the set of services that have the same QoS Level for each one of their respective QoS properties. To compare QoS classes, we introduce a quality indicator I_{q}. I_{q} is the sum of the QoS Levels of all QoS properties (of any service belonging to this QoS Class) weighted by the weights associated with the QoS properties. These weights should in principle be related to the global QoS constraints, meaning that a QoS property that has higher priority for the user should be associated to a stricter global constraint. That is, the quality indicator I_{q} of a QoS class QC is higher (i.e., QC includes services with better QoS) when: (i) QC has higher numbers of QoS properties in the higher QoS Levels, (ii) the QoS properties that are in the higher QoS Levels are of higher priority for the user. I_{q} evaluates the distribution of QoS properties in QoS Levels. Even if I_{q} produces a scalar value for all QoS properties, it does not rely on the actual QoS values but only on their QoS Levels. Hence, there is no hidden compensation among QoS properties values. There may still be compensation among QoS Levels, however this compensation is reflected in the value of I_{q}.

By relying on I_{q}, the local selection phase produces per activity a ranking of the available QoS Classes. If we want to be selective, we keep only the QoS Class with the highest I_{q} for the global selection phase. For example, if this QoS Class has all the QoS properties at the highest QoS Level, this is sufficient. A less selective local selection would also maintain the QoS Class with the next highest I_{q}, etc. This can also be guided by feedback coming from the global selection phase.

Global selection phase. The global selection phase receives as input the sets of services selected per activity of the user task by the local selection phase and combines them into service compositions. The computational complexity of the global optimization problem is still NP-hard, even though our local selection phase is highly selective and reduces considerably the number of services to be examined. Nevertheless, as already pointed out, the main purpose of our global selection is not to identify (as close as possible) the optimal service composition fulfilling the global QoS requirements, but rather several alternative near-optimal compositions that can additionally substitute for each other in case of late binding of services or runtime reconfiguration of the composition. More specifically, we select several services for each activity in the user task such that, no matter what is the service finally executed for each activity, the resulting composition satisfies the global QoS requirements. We introduce a heuristic algorithm to address the global selection phase. Our heuristic consists of the four following steps:

1) Determine the initial service composition. To determine the initial service composition, for each activity in the user task we sort service candidates resulting from the local selection phase and choose the best service. Services are ranked in descending order, first based on their QoS Class (its I_{q}), and then, inside each QoS Class with respect to their overall QoS. The overall QoS of a given service is determined using a QoS utility function that aggregates the weighted and normalized values of all QoS properties. Services inside a QoS Class share the same QoS Level for each one of their respective QoS properties. Hence, it is safe to rank services inside a QoS Class based on their aggregated QoS utilities without risking an inefficient ranking due to hidden compensation among QoS properties.

[2]We note that, inversely to COCOA, here QoS properties are normalized to be stronger with bigger values.

2) Determine the next service composition. Our algorithm checks whether a service composition C_v meets the global QoS requirements. The global QoS values of C_v for each QoS property are calculated based on the structure of the user task, in the same way as for the COCOA service composition detailed in Section 2.2. If the global QoS requirements are met, C_v is considered as a solution (i.e., a near-optimal composition). After that, our algorithm proceeds to checking another service composition C_w obtained by changing a single service in C_v by its next ranked service inside the same activity. We introduce a utility function for evaluating the 'next ranked' services of all activities: it depends on the QoS Class (I_q) and overall QoS of the successor service over the same quantities of the initial service. As a result, our algorithm switches to the service that is the closest one, QoS-wise, to its predecessor, hence it introduces the least local QoS change. We assume that switching to this service will also introduce the least global QoS change (without performing the complete calculation that would take into account the structure of the task). Hence, the resulting composition C_w should yield nearly the same global QoS as C_v, hence it, too, is most likely a near-optimal composition. If conversely C_w does not meet global QoS requirements, it is rejected, the last selected service is removed from the list of its associated activity, and the algorithm steps back to the previous accepted composition C_v. In the case of a rejected initial composition, we form another initial service composition by randomly replacing one or more services by their next ranked services. We remove the replaced services from their associated activities.

3) Checking and selecting service compositions. As already stated, QASSA guarantees that, if a service composition fails during its execution, we can switch to any accepted service composition (i.e., while always respecting the global QoS requirements). We assume here that there are no affiliation dependencies between the already executed part of the composition and the remaining part (we deal with affiliation dependencies later in this section). More specifically, when checking whether the current service composition C_w meets the global QoS requirements, we further make sure that C_w can replace any previously accepted composition C_v without violating the QoS requirements. To achieve this, we maintain the worst value per QoS property and per activity among all the services that have already been accepted or are currently examined (for C_w) for this activity. Accordingly, we calculate global QoS properties from these (currently worst) local QoS values and evaluate C_w based on these (currently worst) global QoS properties. If these meet the global QoS requirements, this means that any combination of services belonging to C_w or the previously accepted compositions meet the global QoS requirements. Only then is C_w selected as a solution. We note here that this also the reason for removing a service that produces a rejected composition in Step (2): even if this service could possibly be successfully combined in another combination of services, there is at least one possible combination (the rejected composition) that does not meet the global QoS requirements.

4) Sorting the selected compositions. Our algorithm iterates on checking service compositions until examining all the services associated with each activity in the user task as selected by the local selection phase. It yields as a result several alternative service compositions ranked with respect to their overall global QoS utility that aggregates the weighted and normalized global values of all QoS properties.

At runtime, the service composition ranked first will be enacted. If the run-time QoS of a service in this composition degrades, we enact another service for the same activity belonging to another composition. The overall QoS of the resulting composition will be different, but it will respect the global QoS requirements (we do not include here the effect of the failed service on the overall QoS). The importance of this flexibility in choosing and enacting services is twofold. On the one hand, it avoids managing complicated QoS correlations among services [18] (i.e., being obliged to execute a precisely specific set of services to satisfy QoS constraints). On the other hand, it avoids rolling back the part of the user task already executed when a service composition fails (again we assume here that there are no affiliation dependencies between the already executed part of the task and the remaining part). We further note that if we consider our global selection approach apart (without the local selection phase), it may lead to discarding a lot of possible solutions. Indeed, it selects service compositions based on the least QoS values among the currently checked composition and the previously accepted ones. That is, our global selection approach is highly selective and hard to satisfy. However, our global selection approach strongly relies on the quality of the local selection phase; the two phases are complementary. More specifically, the services yielded by local selection exhibit high QoS and belong (for each activity in the user task) to the same or successive QoS Classes, therefore they share the same or similar QoS Levels for each one of their QoS properties. Hence, the global selection phase will most likely produce a number of service compositions that will satisfy the global QoS requirements; additionally, these will most likely be near-optimal compositions. Finally, our overall selection approach considerably reduces computational complexity, when compared with the complexity of the global optimization problem.

Computational complexity analysis. We analyze the computational complexity of the local and global selection phases of QASSA. The local selection phase is performed using the k-means++ algorithm, the complexity of which is of $O(log(k))$ [11], where k denotes the number of clusters. As we execute k-means++ for each QoS property and for all the activities in the user task, the overall complexity of local selection is then of $O(Z.N.log(k))$, where Z and N denote the number of activities in the user task and the number of QoS properties, respectively. Therefore, the local selection phase runs in a linearithmic time. Concerning the global selection phase, we first compose Z services (one service per activity) to build the initial service composition. Then, we iteratively check new service compositions by replacing a single service in each iteration. Hence, we perform one iteration for each one of the remaining $T - Z$ services, where T denotes the total number of services associated with all the activities in the user task as selected by the local selection phase. Therefore, the total number of iterations is $T - Z + 1$. Additionally, for each composition, we execute $Z.N$ arithmetic instructions to aggregate the N QoS values of the Z services forming the composition. Then, we execute N comparison instructions to determine whether the N QoS values of the composition satisfy the global QoS requirements. The global selection phase runs then in quadratic time of $O((Z.N + N)(T - Z + 1))$. Based on the above results, we state that QASSA executes in quadratic time. This is much lower than the computational complexity of service selection under global QoS requirements, known to be NP-hard [5, 10].

Service dependencies. During the service selection process, QASSA takes into account affiliation dependencies between activities as expressed in the user task. QASSA deals with affiliation dependencies in a pre-processing step before proceeding to the local and global service selection. Two activities correlated with an affiliation dependency will eventually have as candidate services the intersection of their initial candidate services. These are services that are able to perform both the activities.

In the case of runtime reconfiguration due to a service failure, we may change from a service performing more than one activities to a new one with the same role. If there is no related affiliation dependency between the already executed part of the composition and the remaining part, then this change can be executed without problem. If conversely there is a related affiliation dependency, we would be obliged to roll back a part of the user task already executed so as to employ the new service for all the activities linked by the affiliation dependency.

Distributed execution of service selection. The version of QASSA presented so far assumes the presence of a centralized resource-rich infrastructure. Nevertheless, in mobile environments, it is not always possible to assume the support of such an infrastructure. QoS-aware service composition in mobile environments can be rather underpinned by ad hoc infrastructures formed of mobile and resource-constrained devices. For this reason, we introduce a distributed version of QASSA, which is capable of operating on top of ad hoc infrastructures. Distributed QASSA enables accomplishing service selection as a synergistic interaction between the user device (referred to as *requester*) and other devices available in the environment and willing to contribute (referred to as *helpers*). More specifically, distributed QASSA performs the local selection phase using several helpers simultaneously. That is, we divide the local selection (for the whole user task) into several elementary selection operations, each dealing with a single activity. Then, elementary selection operations are flexibly assigned to helpers (with respect to the number of helpers and their computational capabilities). Ideally, for each activity in the user task, the local selection is executed using a separate helper. After that, the requester collects the local selection results from helpers and performs the global selection phase on the user device. The global selection phase is difficult to carry out in a distributed way, because it requires a global vision of QoS information and the structure of the composition [99]. Typically, global optimization requires a resource-rich device, given the computational complexity of the problem. However as discussed in the previous, our global selection approach has a low computational complexity; thus it can be carried out using only the resource-constrained device of the requester.

Experimental evaluation. We conducted a set of experiments to assess first the centralized version of QASSA. For the evaluation of QASSA, we are interested in two metrics:

- *Execution time*: It measures the timeliness of QASSA with respect to the size of the selection problem in terms of the number of activities and the number of candidate services per activity.

- *Optimality*: It measures how optimal is the overall global QoS utility of the highest ranked service composition provided by QASSA. This is determined by the ratio of the QoS utility resulting from QASSA over the optimal QoS utility given by a brute-force algorithm.

For the purpose of our experiments, we developed a *Composition Generator* that randomly generates service compositions. Our Composition Generator takes as parameters the number of activities (denoted Z) and the number of candidate services per activity (denoted M), and it proceeds in two steps: (i) builds a user task which comprises Z activities structured with randomly chosen control constructs, (ii) associates M concrete services to each activity in the composition. QoS values for these services are acquired from the QWS dataset available online[3]. This dataset consists of 5000 real Web services, each with a set of 9 QoS properties measured using commercial benchmark tools [3]. Once service compositions are generated, we further need to configure the execution of QASSA with respect to the following parameters:

- Aggregation approach: As already mentioned, QASSA supports the two QoS aggregation approaches that we also applied in COCOA for calculating the global QoS properties of service compositions. These are worst-case and mean-value aggregation. We opt for the worst-case approach as the default method for aggregating QoS values. We further perform experimentation with respect to the two aggregation approaches in order to study their impact on the timeliness and optimality of QASSA.
- Global QoS constraints: QASSA requires as input global QoS constraints for each QoS property imposed by the user on the whole composition. As we do not have real values of user requirements, we opt for a statistical method to determine global QoS constraints. For each QoS property, we calculate the mean value m_{i} of service candidates associated with each activity A_{i}, then we aggregate all mean values according to the structure of the user task. That is, we set the global QoS constraint for each QoS property to the result of this aggregation. We further vary the global QoS constraints with respect to some statistical limits in order to analyse their impact on the timeliness and optimality of QASSA.

We have measured the execution time of QASSA with respect to the numbers of activities and services per activity, as well as with respect to the number of QoS properties. For up to 50 activities, up to 200 services per activity and up to 5 QoS properties, the execution time increases up to 89ms. This increase is almost linear along with the number of activities. In terms of absolute numbers, the execution time (less than 0.09s) is certainly satisfactory for user interactive applications in the mobile environment [113]. We have further compared the above results with those published in [5], which presents an efficient approach that combines local and global selection techniques for service selection under global QoS constraints. The authors consider the same dataset (i.e., QWS dataset) and configuration as in our experiments; however they use a more powerful experimental setup. Their results are of the same order as ours (execution time up to approximately 0.09s).

Concerning the optimality of QASSA, we measure it while varying the number of activities between 5 and 10, the number of services per activity between 100 and 200, and the number of QoS properties between 2 and 5. We observe that the optimality of QASSA is generally more than 90%, and it can reach 100%. Comparing with [16] in terms of optimality, we see that for the same configuration both works produce roughly the same optimality.

We have further evaluated our selection approach with respect to more difficult conditions to see its optimality (we also checked that these have no effect on its timeliness). More specifically, we tried:

1. Decreasing the numbers of activities and candidate services per activity. We evaluated the case of 5 activities and 50 services per activity.
2. Comparing between the worst-case and mean-value aggregation approaches for calculating the global QoS properties of service compositions.
3. Making the global QoS constraints more stringent. As we do not have real values of user requirements and similarly to our experiments so far, we rely on the QoS properties of service candidates per activity. This time, we add to the mean value m_{i} per property the associated standard deviation σ_{i} (where properties are normalized to be stronger with bigger values). Again, the corresponding global QoS constraint is calculated by aggregating all $m_{\mathrm{i}} + \sigma_{\mathrm{i}}$ values according to the structure of the user task.

[3] `http://www.uoguelph.ca/~qmahmoud/qws`

We have seen that the above conditions have an impact on the optimality of QASSA, which, in certain cases accumulating these conditions, drops below 50%. Regarding Case (1), when the service population per activity is lower, the probability to find services with a satisfactory value for most QoS properties is also lower. We note here that in our experiments we applied a highly selective local selection, which means that only one QoS Class (the one with the highest I_{q}) qualifies for the global selection phase. This produces less service compositions for the global selection phase. A low number of activities in the task decreases further the number of service compositions (as one service from one activity is selected in each iteration until all services are examined). Consequently, the probability to find near-optimal compositions is also lower, which yields a lower optimality. Regarding Case (2), the worst-case aggregation approach leads to discarding more service compositions in the global selection phase than the mean-value aggregation approach. We see that the former results in general in lower optimality than the latter. Finally, Case (3) leads also to less accepted service compositions in the global selection phase. As the QWS dataset deals with a large number of services, the central limit theorem states that the associated QoS values follow the normal distribution law. Then, we note indicatively that if we applied the corresponding $m_{\mathrm{i}} + \sigma_{\mathrm{i}}$ as local QoS constraints for each activity (as a projection of the global QoS constraints), 84,1% of service candidates associated with each activity would be discarded (50% if m_{i} was the local QoS constraint). Accordingly, a large number of service compositions are rejected by the global selection, which results again in lower optimality. Overall, we see that the results of QASSA are of lower optimality when the number of service compositions qualified for the global selection phase and/or finally accepted by the global selection is low. A remedy to this is to apply a less selective local selection, allowing more successive QoS Classes to qualify for the global selection phase. This produces more service compositions for the global selection. We note also here that while we evaluate the optimality of QASSA, its main objective is to produce a number of interchangeable good service compositions that satisfy the global QoS constraints, while at the same time complying with the time and resource constraints of the interactive mobile environment.

Finally, we have evaluated the distributed version of QASSA using an experimental setup employing mobile devices, and hence having less computational resources than our previous experiments. In particular, we evaluate the execution time of the local and global selection phases of the distributed version of QASSA. We do not take into account the coordination and communication overhead among the devices participating in this distributed execution, which is not significant – if we assume normal network connectivity – compared to the overall execution time of the algorithm [91]. We note that the distributed execution has no effect on the optimality of QASSA. For the local selection, we measured the execution time for only one activity. This is because each helper device processes in parallel local selection for a single activity in the user task. We have seen that for 5 QoS constraints, up to 200 services and up to 50 activities, the local selection is executed in at most 0.022s, whereas the global selection is executed in at most 1.2s. Hence, also the distributed version of QASSA shows satisfactory timeliness with respect to on-the-fly service composition in mobile environments.

Chapter 3

Interoperability of Middleware Protocols

The issue of software system interoperability is very old, it exists since one wished to interconnect software components or systems developed independently. Distributed systems add more complexity to the problem: system heterogeneity may have to be tackled at multiple layers. We focus in this chapter on communication middleware protocol interoperability. Communication middleware determines to a large extent a system's adopted technology (assuming IP everywhere, even if interoperability in the lower layers is also a hard problem); it also determines the interaction style of the overlying applications. We present three contributions that aim at dealing with the middleware interoperability problem in a general way and for the majority of existing (and possibly future) protocols.

In *Interoperability across interaction paradigms* (Section 3.1), we analyze the semantics of the various interaction styles supported by existing middleware protocols and propose a solution to their interconnection based on automated generation of interoperability software artifacts.

Then, in *Timed analysis of mobile interactions* (Section 3.2), we model timed semantics of the interaction styles identified in Section 3.1, in particular for mobile interactions, and analyze their performance.

Finally, in *QoS analysis of mobile interactions across interaction paradigms* (Section 3.3), we enrich the models and analyses of Section 3.2 with the complete semantics of the underlying mobile protocol infrastructure. Our objective is to enable the evaluation of the end-to-end QoS semantics when interconnecting heterogeneous interaction styles, which we call *interoperability effectiveness*.

3.1 Interoperability across interaction paradigms

(Publications: [74])
(Ph.D. thesis of Georgios Bouloukakis, UPMC, 2017 [43]. Co-supervision with Valérie Issarny.)

The essential issue of interoperability in distributed systems is particularly challenging in open dynamic environments. Complex applications in the current and Future Internet, including the Internet of Things and the mobile environment, may be composed from extremely heterogeneous systems. This includes, for instance, lightweight embedded systems (e.g., sensors, actuators and networks of them), mobile systems (e.g., smartphone applications), and resource-rich IT systems (e.g., systems hosted on enterprise servers and Cloud infrastructures). Assuming an underlying IP layer unifying the various access and networking technologies, these heterogeneous system domains still differ significantly in terms of *interaction paradigms*, interaction protocols and data representation models, which are most often provided by supporting middleware platforms. *Client-server (CS)*, *publish/subscribe (PS)*, *data streaming (DS)* and *tuple space (TS)* are among the most widely employed interaction paradigms, with numerous related interaction protocols, such as REST [67] and SOAP [81] Web services (CS), MQTT [17] and AMQP [76] (PS), WebSockets [65] (DS), or JavaSpaces [71] (TS).

To enable such applications, the heterogeneity between the involved system domains needs to be tackled. Existing cross-domain interoperability efforts are based on, e.g., bridging interaction protocols [23], wrapping systems behind standard technology interfaces [13], and providing common API abstractions [50, 77, 118, 142]. In particular, such techniques have been applied by open system integration

paradigms, such as service oriented architecture (SOA) and enterprise service bus (ESB) [52, 117]. Nevertheless, when it comes to integrating systems featuring heterogeneous interaction paradigms, existing interoperability solutions are typically partial, applying to or imposing specific interaction protocols. On the other hand, interaction paradigms have been widely studied, with theoretical approaches providing them with formal semantics by relying on concurrency theory, process algebras and architectural connectors (e.g., see [47]). These approaches typically identify semantics for individual paradigms but not cross-paradigm semantics, with some notable exceptions as in [4, 63]. However, these last efforts remain at the level of modeling and do not result in any interoperability solution. Finally, I point out the efforts presented in [34, 35, 37], developed by some of my fellow colleagues in the Inria ARLES and MiMove teams. These constitute a dynamic interoperability solution that deals with system heterogeneity at both the application and middleware layers in a combined way. We have focused only on the middleware layer, targeting a solution that comprehensively covers the diverse but countable set of possible semantics here; we assume a direct or otherwise provided system mapping at the application layer.

More specifically, we have introduced an interoperability solution based on abstraction and merging of the common high-level semantics of interaction paradigms, which is sufficiently general and extensible to accommodate many different, existing and potentially future, interaction protocol technologies. Our systematic approach is carried out in two stages. First, a middleware platform is abstracted under a corresponding interaction paradigm among the four base ones, i.e., CS, PS, DS and TS. To this aim, we elicit a connector model for each paradigm, which comprehensively covers its essential semantics. Then, these four models are abstracted further into a single *Generic Middleware (GM)* connector model, which encompasses their common interaction semantics. Based on GM, we introduce a solution for the automated synthesis of interoperability software artifacts that enable interconnecting the base interaction paradigms. We realize our interoperability solution as a lightweight, highly flexible and fully distributed protocol bus, the *eVolution Service Bus (VSB)*. Based on our VSB platform, we propose a comprehensive solution to the peer-to-peer integration of software entities that rely on heterogeneous interaction paradigms. These are entities of the current and Future Internet, including services and IoT Things. The integration results in complex applications that take the form of dynamic choreographies of services and Things [21,82]. Our overall approach generalizes the way to design and implement service-oriented distributed applications, where the employed interaction paradigms are explicitly represented and systematically integrated.

Connector models for base interaction paradigms. We identify the semantics of the four principal interaction paradigms, i.e., CS, PS, DS and TS, and elicit a connector model for each paradigm. Our modeling proposition is the outcome of an extensive survey of these paradigms as well as related middleware platforms in the literature.

Our models represent the essential semantics of interaction paradigms, concerning *space coupling*, *time coupling*, *concurrency* and *synchronization coupling* [4, 63]. Space coupling determines how peers interconnected via the connector identify each other and, consequently, how interaction elements (e.g., events for a PS connector) are routed from one peer to the other; in the case of PS, this may correspond to topics, queues or content filters. Time coupling determines if peers need to be present and available at the same time for an interaction or if the interaction can take place in phases occurring at different times; hence, peers in CS or DS have a strong time coupling, while PS subscribers can receive previously published events upon their reconnection to a broker. Concurrency characterizes the exclusive or shared access semantics of the virtual channel established between interacting peers; for instance, non-coordinating TS peers accessing a data space may compete for taking (and not just reading) the shared copy of some data. Finally, synchronization coupling determines whether the initiator of an end-to-end interaction blocks or not until the interaction is complete; in the former case, the interaction is executed in a synchronous way between the interacting peers, such as in the case of a synchronous CS request-response exchange. These four categories of semantics are of primary importance, because these are end-to-end semantics: when interconnecting different interaction paradigms, we seek to map and preserve these semantics.

Inside each interaction paradigm, we identify, more specifically, one or more of four *interaction types*: *one-way interaction*, *two-way synchronous interaction*, *two-way asynchronous interaction* and *two-way streaming interaction*. We also distinguish between the two *roles* involved in an interaction, such as: *sender* and *receiver* (one-way), *client* and *server* (two-way synchronous/asynchronous), *consumer* and *producer* (two-way streaming). These interaction types and roles incorporate the above semantics of end-to-end interactions. We list below the interaction types identified inside the four interaction paradigms:

1. *CS one-way*: corresponds to one-way direct messaging between a sender and a receiver without any

intermediate node supporting persistent queueing of messages;

2. *CS two-way synchronous*: corresponds to synchronous request-response interaction between a client and a server;

3. *CS two-way asynchronous*: here the response is emitted and received asynchronously at a later time with respect to the request;

4. *PS one-way*: we model here the emission of an event by a publisher (sender) and the reception of this event by a subscriber (receiver) through the broker; we assume that the subscriber is already subscribed for receiving events; queue-based messaging is also included in this interaction type;

5. *PS two-way streaming*: we model here the subscription request of a subscriber (consumer) to a broker (producer), followed by a flow of related events delivered by the broker; the event publishing part is transparent to this model;

6. *DS one-way*: we model here the emission of a data element by a producer (sender) and the reception of this data by a consumer (receiver) without any intermediate node supporting persistent queueing of data; we assume that the consumer has already set up the streaming connection with the producer;

7. *DS two-way streaming*: we model here the streaming request of a consumer to a producer, followed by a flow of data from the producer to the consumer;

8. *TS one-way*: we model here the writing of a tuple into the tuple space (receiver) by an entity accessing the tuple space (sender);

9. *TS two-way synchronous*: we model here the read or take request of an entity (client) accessing the tuple space (server), followed by a synchronous response delivering the tuple(s) corresponding to the request; the tuple writing part is transparent to this model.

We remark that the four interaction types appear across the four interaction paradigms. Indeed, the interaction type abstraction which is more fine-grained than the interaction paradigm abstraction. This already provides a hint towards our interoperability solution that we describe next.

We represent the semantics of an interaction paradigm and its supported interaction types and roles in the corresponding connector's abstract API (Application Programming Interface). This API presents the programming model supported by the connector and offered to the peers that use the connector for their interaction. The API is a set of (local and/or networked) primitives expressed as operations or functions supported by the connector. This abstract API can be refined to a specific middleware platform by mapping to the primitives and incorporating the data structures and types of the middleware platform. We further elaborate sequence diagrams that show the detailed interactions between the peers using this API.

Unifying generic connector model. Given the above four base connector models (CS, PS, DS and TS), we introduce the Generic Middleware (GM) connector model. GM is a generic connector that comprehensively abstracts and represents the semantics of various middleware protocols that follow the four base connector models, hence it can represent the majority of the existing and possibly of the future middleware protocols.

In particular, GM models the four interaction types and their associated roles supported by the base connector models in an abstract way. For this, it defines two main high-level API primitives: i) `post` employed by a peer for sending data to one or more other peers, and ii) `get` employed by a peer for receiving data. For example, a PS `publish` primitive can be abstracted by a `post` primitive. GM's API includes a number of variations of these primitives in order to precisely express the semantics of the interaction types. GM's API further specifies unifying space coupling semantics for all four base interaction paradigms. A `GM message` can represent each one of `CS message`, `PS event`, `DS data` or `TS tuple`. Additionally, the `destination` and `scope` parameters of GM's API identify the receiver(s) of a GM message. These may be mapped, e.g., to the URL and a supported `operation` of a CS server, or to the network address and a supported `topic` of a PS broker. Finally, GM introduces two timing parameters in its API that are common among the four base interaction paradigms. The `lifetime` parameter delimits the validity and availability of application data in time for an asynchronous interaction (e.g., `lifetime` of

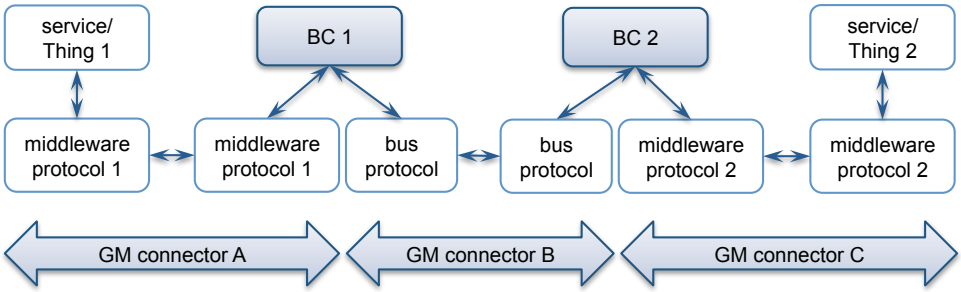


Figure 3.1: VSB architecture.

a PS event), while the `timeout` parameter delimits the interval for a completed synchronous interaction (e.g., for a two-way synchronous CS interaction).

Further to the GM API, we introduce the *Generic Interface Description Language (GIDL)* for specifying interfaces in an abstract way. A GIDL interface corresponds to an application component that employs any middleware protocol that can be abstracted into the GM connector. GIDL enables the definition of operations provided or required by an application component that follow the four interaction types and their associated roles. Besides an operation's interaction type and role, the names and data types of its input and output parameters are also specified. The description is complemented by the physical address of the component. We have specified GIDL as a meta-model in the Eclipse Modeling Framework (EMF). This allows us to develop tools that rely on the Eclipse environment and enable functionalities such as: (i) defining the GIDL interface description of an application component, and (ii) based on this description, generating an interoperability artifact that will enable interconnecting this application component with other heterogeneous application components, as we discuss next.

Interoperability bus and interoperability artifact synthesis. By relying on the GM and GIDL abstractions, we have introduced the eVolution Service Bus (VSB). Its objective is to seamlessly interconnect services & Things that employ heterogeneous interaction protocols at the middleware level, e.g., SOAP, REST, CoAP, MQTT, WebSockets, etc. VSB follows the basic concept of the Enterprise Service Bus (ESB) paradigm [52]. In this paradigm, a common intermediate bus protocol is used to facilitate interconnection between multiple heterogeneous middleware protocols: instead of implementing all possible conversions between the protocols, one only needs to implement the conversion of each protocol to the common bus protocol, thus considerably reducing the development effort. This conversion or protocol bridging is done by a component associated to the service or Thing in question and its middleware, called a *Binding Component (BC)*, as it binds the service or Thing to the service bus. The VSB architecture is depicted in Figure 3.1. Based on GM and GIDL, we introduce a systematic approach for the automated synthesis of VSB BCs, which we present in the following.

To bind a service or Thing to the service bus, a VSB BC employs the same (or symmetric, e.g., client vs. server) middleware protocol as its associated service or Thing and at the same time the bus protocol (both via third-party software libraries). Furthermore, the BC contains a *conversion logic* that maps between the primitives of the bridged protocols. To enable such mapping in a generic way, we rely on the GM connector model. More specifically, either of the bridged protocols is modeled and abstracted by an instance of the GM connector, and the conversion is performed at the GM level of abstraction. Accordingly, our approach to BC building relies on, first, introducing a *generic BC* architecture at GM level, and, then, customizing this generic architecture into a *concrete BC*. The generic BC architecture is depicted in Figure 3.2, together with a resulting concrete BC for the VSB architecture of the previous figure.

A Generic BC (GBC) performs bridging between the two associated instances of the GM connector (let them be X and Y), to each of which it connects through the GM API. The bridging functionality is implemented by the GBC conversion logic, which is a set of rules of the type:

$$\begin{aligned} &\textit{if } \texttt{get} \textit{ primitive received on GM connector } X(Y), \\ &\textit{then execute symmetric } \texttt{post} \textit{ primitive on GM connector } Y(X) \end{aligned}$$

The association between `get` and symmetric `post` primitives inside the GM API is based on the four

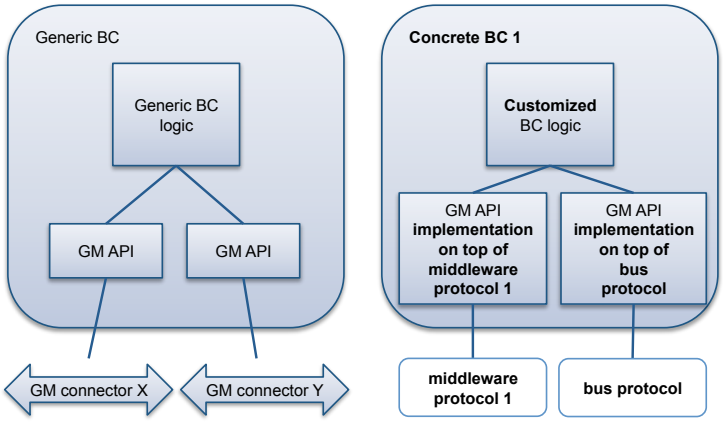


Figure 3.2: Generic BC architecture and concrete BC.

GM interaction types and their associated roles. In particular, the above simple GBC logic assumes that the GM connectors X and Y realize the same interaction type. A more complex logic could possibly bridge between different interaction types.

We customize a GBC into a concrete BC according to: i) the service or Thing to which the concrete BC is associated and its middleware protocol, and ii) the selected VSB bus protocol, or equivalently, the selected common intermediate middleware protocol of the overall application that may integrate many heterogeneous services and Things. To enable GBC customization, we develop a *protocol resource pool*. This pool contains *GM API implementations on top of concrete middleware protocols*. These are built by mapping the primitives and semantics of the concrete middleware protocols to primitives and semantics of the GM API. We develop these GM API implementations as generic code excerpts in Java. To customize the GBC into a concrete BC, we select from the protocol resource pool the two GM API implementations that correspond to the service's or Thing's middleware protocol and the VSB protocol. By attaching to them the third-party libraries that implement the two middleware protocols, we build two concrete instances of the GM connector. The concrete BC has to bridge between these two concrete GM instances. For this, the Generic BC logic is customized with the concrete data parameters of the service or Thing in question, as described in its related GIDL model.

Based on the above solution, any heterogeneous service or Thing that employs a middleware protocol following one of the CS, PS, DS and TS interaction paradigms can be bound to VSB and have its middleware protocol converted to the bus protocol. Furthermore, since the common bus protocol is abstracted based on the GM connector in the same way as any service's or Thing's protocol, different protocols can be introduced as VSB's common bus protocol. Finally, development of BCs is a tedious and error-prone process, which can highly benefit from our automated systematic support. This can help application developers integrate heterogeneous services & Things inside complex applications. Furthermore, automated BC synthesis is essential for applications relying on dynamic runtime composition of heterogeneous services & Things, where there is no human intervention.

Implementation & Assessment. We have conceived and developed VSB as an interoperability bus destined to serve dynamic choreographies of services but also Things as first-class entities [21, 82]. VSB presents the following advancements:

- VSB is a unified interoperability solution for both services and Things participating in choreographies;
- VSB provides support for all of the client-server, publish/subscribe, data streaming and tuple space interaction paradigms, hence covering the majority of the existing and possibly future middleware protocols;
- VSB's interoperability solution is based on the interaction type abstraction which is more fine-grained than the interaction paradigm abstraction; this enables a flexible mapping between interaction paradigms and hence between middleware protocols;

- Different protocols can be introduced as VSB's common bus protocol with the same easiness as for integrating support for a new middleware protocol of a service/Thing;
- VSB BCs are built and deployed as necessary; hence, no BC is needed when a service/Thing employs the same middleware protocol as the one used as common bus protocol;
- From the previous, we see that VSB is flexible and lightweight; there is no need for relying on and/or providing a full-fledged ESB platform;
- While generic and modular, VSB's architecture includes few levels of indirection in the processing of primitives when converting between protocols; this makes it simpler, lighter and faster.

VSB is utilized as core component of the *H2020 ICT CHOReVOLUTION project*[1] and enables heterogeneous interactions in services/Things choreographies. The foundations were set and a previous version of VSB was developed in the *FP7 ICT IP CHOReOS project*[2]. Currently, VSB supports the following middleware protocols: REST, CoAP, MQTT, WebSockets, SemiSpace and DPWS. It is released as open source software[3].

We have evaluated the VSB framework and runtime environment with respect to two criteria: i) the support that the VSB framework offers to developers when developing a new application – which may contain a number of heterogeneous services/Things; and ii) the runtime performance of the synthesized BCs in terms of response time and throughput.

Regarding the first criterion, we have measured the effort spent by a software developer in order to develop a choreography integrating 4 heterogeneous Things employing 4 different middleware protocols (WebSockets, REST, MQTT, SemiSpace), interconnected via CoAP employed as bus protocol. In the first test case, the developer uses the VSB Eclipse plugin to provide the GIDL descriptions of the 4 Things and the mapping between their data, and then employs the VSB BC generator to automatically synthesize the corresponding BCs. In the second test case, the developer develops "manually" the software artifacts that adapt among the primitives of the different protocols and the data of the Things. Our evaluation showed that the VSB framework reduces the application development effort considerably when compared with manual development. Particularly, the application developer was able to save 78-96% of person-hours in our scenario. Additionally, when employing the VSB framework, the developer is only required to deal with the application-level descriptions of Things, while she is spared all the complexity of the middleware-level heterogeneity and adaptation. On the contrary, building interoperability artifacts without VSB requires from the developer to be aware of the corresponding APIs and protocols. Finally, we have not measured in our evaluation VSB's effect on correct software development. Even if our evaluation sample included a single developer, the produced results are indicative of the benefit brought by VSB.

Regarding the second criterion, we have measured the introduced latency (for protocol and data conversion) inside BCs and their supported throughput for traffic ranging from low up to stress conditions. Our experimental setup comprises a variable number of WebSocket senders generating one-way message traffic towards an MQTT receiver through alternating REST, CoAP and DPWS bus protocols. We make sure to create performance bottlenecks (host CPU usage reaching 100%) at the level of BCs including the bus protocol, while keeping the rest of the entities below their stress level [61, 134]. Our experiments showed that when stress-testing the BCs (in particular the first BC encountered by the one-way message traffic), the BC conversion logic demonstrates a perfectly scalable behavior, whereas at some point the bus protocol (for all three experimented protocols) becomes the performance bottleneck. Then, each bus protocol reveals different stress-level properties in terms of response time and maximum throughput. This certainly points out the importance in the choice of the bus protocol, depending on the interconnected services/Things and their middleware protocols. Additionally, we observed that the processing latency inside the BC logic was in the order of 1/30 to 1/10 of the end-to-end response time (for all three bus protocols) concerning one-way interactions, for input traffic loads causing moderate queueing effects. Our evaluation of BCs shows that they introduce limited performance overhead into end-to-end interactions.

3.2 Timed analysis of mobile interactions

(Publications: [90])

[1] `http://www.chorevolution.eu`
[2] `http://www.choreos.eu`
[3] `https://gitlab.ow2.org/chorevolution/evolution-service-busevolution-service-bus`

(Ph.D. thesis of Georgios Bouloukakis, UPMC, 2017 [43]. Co-supervision with Valérie Issarny.)

The Generic Middleware (GM) connector, which we introduced in the previous section, models interactions where peers may have varying time coupling. For example, GM may represent a client-server interaction, where both peers should be present at the same time. In a different case, GM may represent a publish/subscribe interaction, where published events generated by publishers are stored by the broker for disconnected subscribers and are delivered to them upon their reconnection. To identify the effects of time coupling, we have analyzed GM interactions with regard to their timing behavior, more particularly in the mobile environment. For this, we introduce timing models; our modeling approach is applied at the level of the GM interaction types. We have modeled, in particular, the one-way and two-way synchronous interaction types; our approach can be extended to the other two interaction types. Our models rely on the two timing parameters of the GM's API: `lifetime` captures validity and availability of application data in time for an asynchronous interaction, while `timeout` represents the interval for a completed synchronous interaction. Additionally, we introduce the `time_on` and `time_off` parameters to capture the intermittent availability of mobile data recipients.

We study in detail the effect of these timing parameters on interactions regarding their latency and success rates. We represent and analyze the behavior of our model by relying on timed automata. In particular, we examine the conditions for successful one-way and two-way synchronous interactions, and verify reachability and safety properties, by employing the Uppaal [24] model-checker. This analysis provides us with formal conditions for successful GM interactions and their reliance on the `lifetime`, `timeout` and `time_on`/`time_off` parameters, as well as on the stochastic behavior of interacting peers. While we have modeled and formally analyzed both one-way and two-way synchronous interactions, we report in this section our results concerning the former. We further perform statistical analysis through simulation of one-way interactions over multiple runs, and study the success rate vs. latency trade-off for varying `lifetime` and `time_on`/`time_off` periods. Simulation outputs are compared with experiments run on our VSB testbed with respect to the accuracy of predicted results.

Timed automata have been applied to a variety of time-sensitive systems, such as publish/subscribe middleware [83], publish/subscribe applications [20] and distributed asynchronous real-time control systems [139]. Model checkers like Uppaal [24] and PRISM [96] (the latter supporting analysis of probabilistic timed automata) are employed to analyze timed and probabilistic properties of such systems. Similarly, a timed message-passing process algebra (an extension to the pi-calculus) [39] has been employed in [14] for analyzing the MQTT IoT protocol [17]. On the other hand, the work presented in [4] identifies and analyzes dimensions of coupling among interacting entities in communication middleware platforms, similarly to our interaction paradigm abstractions of Section 3.1. These coupling (including time coupling) semantics are modeled by using Colored Petri Nets; this however does not allow a quantified timed analysis.

Our modeling and analysis allows to study in a unified quantified manner interactions with varying time coupling employing the principal interaction paradigms. Application designers can use our analysis to compare between interaction paradigms and their instantiations into protocols, select among them, and tune the timing parameters of the overlying application effectively according to the sought trade-off between success rate and latency.

Time model of GM interactions. We focus here on one-way interactions over a client-server (CS), publish/subscribe (PS), data streaming (DS) or tuple space (TS) connection, abstractly represented by the GM connector. In a GM one-way interaction, a sender entity uses the `post` primitive to emit a message associated to a `lifetime` period; this message can be procured using the `get` primitive by a receiver entity during its availability period `time_on`. More specifically, `lifetime` refers to emitted CS messages, PS events, DS data or TS tuples, and characterizes both availability in time, e.g., thanks to storing by a broker, and validity, e.g., for data that become obsolete as part of a data feed. Hence, `lifetime` is very short for CS or DS one-way messages, which need to be delivered immediately. `time_on` characterizes the interval during which a receiver is connected and available. During this active period, the receiver can receive one or more sent messages, events, data or tuples. Between active periods, the receiver is disconnected (`time_off`), e.g., for energy-saving or other application-related reason.

We assume that the `lifetime` and `time_on` parameters are set by application designers. `post` operations are initiated repeatedly, with a stochastic interval $\delta_{\texttt{post}}$ between two successive `post` operations. Similarly, `get` operations are initiated repeatedly, with a stochastic disconnection interval `time_off` between two successive `time_on` periods. Overall, the `post` and `get` operations are independent and have

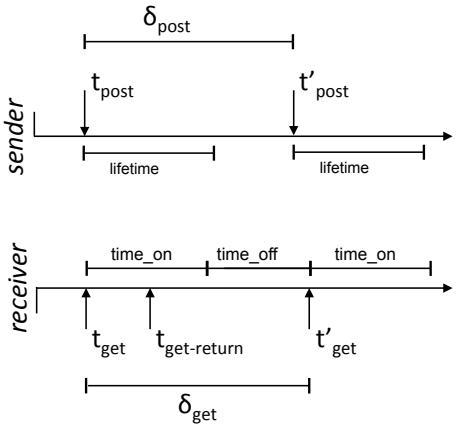


Figure 3.3: Time model of GM interactions.

individual time-stamps: we assume that the application entities undertaking the sender and receiver roles enforce their semantics independently (no coordination). Our time model is depicted in Figure 3.3.

This model allows concurrent `post` messages. Buffers of available receiving entities (including the broker and tuple space) are assumed to be infinite, hence there is no data loss due to limited buffering capacity. The data processing, transmission and queueing (due to processing and transmission of preceding data) times inside the interaction are assumed to be negligible compared to the durations of `lifetime` and `time_on` periods. With regard to network transmission, this abstracts the effect of the underlying network infrastructure, which is assumed ideal. Moreover, regarding queueing, we assume that we have no heavy load effects. This means that: all posts arriving during a `time_on` period are immediately served; all posts arriving during a `time_off` period are immediately served at the next `time_on` period, unless they have expired before. This corresponds to a $G/G/\infty/\infty$ queueing model, where there is an infinite number of on-demand servers, hence there is no queueing. We assume that the general probability distribution characterizing service times incorporates the disconnections of receiver entities.

Abstracting data processing, transmission and queueing effects in this model facilitates its formal analysis, which we present in the following. In another piece of work, we have extended this model with actual data processing, transmission and queueing; we have then performed analysis based on queueing networks; we present this other work in Section 3.3.

Timed automata-based analysis of GM interactions. We have built a timed automata model that represents the typical behavior of the GM connector and of application components using this connector for performing the interactions described in our time model. By relying on the expressive power of timed automata, we are able not only to model deterministic timing conditions for such interactions, but also to introduce basic stochastic semantics for the sender's and receiver's behaviors. In particular, we rely on timers and invariant conditions on them for enabling probabilistic transitions inside the automata. In this way, the sender and the receiver have independent stochastic behaviors.

We represent GM interactions with the connector roles *GM sender*, *GM receiver*, and with the corresponding *GM glue*, where we apply common software architecture principles. The two roles model the behavior expected from application components employing the connector, while the glue represents the internal logic of the connector coordinating the two roles.

The automaton modeling the GM sender behavior repeatedly emits a `post` message to the glue. At most one `post` operation can be active at a time. The end of a `post` operation is either a successful reception of the message or an expiration of its `lifetime`; in both cases we artificially "advance time" to `lifetime`. Between the end of a `post` operation and the initiation of a new `post` operation, there is a stochastic time interval uniformly distributed in `[0,max_delta_post-lifetime]`. With regard to our time model, we opted here for restraining the concurrency of `post` operations for simplifying the architecture of the glue. The present model (sender, receiver and glue) can be compared to one of the infinite on-demand servers of the $G/G/\infty/\infty$ model introduced above. Nevertheless, the present model is sufficient for verifying conditions for successful GM interactions. These conditions relate any `post` operation with an overlapping `get` operation; possible concurrency of `post` operations has no effect

on this. Moreover, we will see that these conditions are independent of the probability distributions characterizing the sender's and receiver's stochastic behaviors.

The automaton modeling the GM receiver behavior repeatedly emits a `get` message to the glue, with at most one `get` operation active at a time. The duration of the `get` operation is `time_on`. Before reaching `time_on`, multiple messages (posted by GM sender) may be delivered to the receiver by the glue. In the opposite case, where the whole `time_on` period elapses with no message received, a no-interaction event is generated. There is a succession of `time_on` and `time_off` intervals, with the former lasting `time_on` time and the latter lasting a stochastic time uniformly distributed in the interval `[0,max_time_off]`.

The automaton modeling the GM glue behavior determines the synchronization of the incoming `post` and `get` operations. A successful synchronization between such operations leads to a successful-interaction event, where a posted message is delivered to the receiver. The glue keeps the timing concerning the `lifetime` of a pending `post` operation. If this time expires, a failed-interaction event is generated.

Using the UPPAAL model checker, we provide and verify essential properties of our timed automata model, including formal conditions for successful GM interactions. More specifically, we verify reachability and safety properties of the combined automata GM sender, GM receiver and GM glue. We verify a set of reachability and safety properties that characterize the timings of the sender's stochastic behavior; these are precisely the timings we presented above. This verifies that the sender automaton behaves as expected. We verify similar properties that characterize the timings of the receiver's stochastic behavior. Finally, we verify conditions, as safety properties, for successful interactions using the glue automaton:

(1) A successful-interaction event implies that while a `post` operation is active a `get` event occurs, or while a `get` operation is active a `post` event occurs.

(2) A failed-interaction event means that `lifetime` is reached for an active `post` operation and no `get` operation is active. Additionally, the ongoing inactive `get` interval (`time_off`) entirely includes the terminating active `post` interval (`lifetime`).

(3) Symmetrically to (2), a no-interaction event implies that the end of the `time_on` interval is reached for an active `get` operation and no `post` operation is active. Additionally, the ongoing inactive `post` interval entirely includes the terminating active `get` interval (`time_on`).

Hence, successful interactions are determined by the durations and relative positions in time of the active/inactive `post` and `get` intervals. These depend on the deterministic parameter constants `lifetime`, `time_on` and on the stochastic parameters $\delta_{\texttt{post}}$ and `time_off`. Nevertheless, conditions are expressed in a general way, independently of the specific `post` and `get` stochastic processes. Hence, our analysis results provide application designers with general formal conditions for successful GM interactions and their reliance on observable and potentially tunable system and environment parameters. For example, from Conditions (1) & (2), an application designer may constrain `time_off` intervals of receiver processes with respect to `lifetime` intervals of posted messages or alternatively increase the latter with respect to the former, to limit failed interactions. As `time_off` intervals are stochastic, the designer may be able to guarantee a statistical upper bound for failed interactions. Similarly, from Conditions (1) & (3), the designer may tune `time_on` intervals of receiver processes so that they best adapt to the statistical properties of sender processes in order to limit 'no-interactions'; this can increase the energy efficiency of the former, e.g., for resource-constrained devices.

Simulation-based analysis of GM interactions. In order to showcase how an application designer can leverage our formal analysis results, we have performed simulation experiments over our time model of GM interactions. These experiments demonstrate how the designer can tune system parameters according to Conditions (1-3) in order to ensure a sufficient interaction success rate or a suitable tradeoff between interaction success rate and interaction latency.

We run Monte Carlo simulations in *Scilab*[4] to analyze the effect of varying `lifetime`, `time_on` and `time_off` periods on GM interactions. We assume Poisson arrivals at a specific rate for subsequent posts (hence, $\delta_{\texttt{post}}$ follows the corresponding exponential distribution). Similarly, there are exponential intervals (of varying rate) between subsequent gets. Our simulation model enables concurrent posts with no-queueing. It corresponds to an $M/G/\infty/\infty$ queueing model. The model was run for $10,000$ `get` periods to collect interaction statistics.

[4] `http://www.scilab.org`

We have measured the rates of successful interactions for various values of `lifetime`, `time_on` and `time_off` periods. As expected from our formal analysis, increasing `time_on` periods for individual `lifetime` values improves the success rate. However, we note that the success rate is severely bounded by `lifetime` periods. For the time-coupled CS or DS interactions, where the `lifetime` period is very low, the success rate, even at higher `time_on` intervals, remains much lower than for, e.g., the time-decoupled PS or TS interactions, where a certain `lifetime` is guaranteed by the broker or the tuple space. Reducing `time_off` intervals produces a significant improvement in the success rate, especially for the CS/DS case. For the PS/TS interaction paradigms, where the `lifetime` period can be varied: a higher `lifetime` period combined with higher `time_on` or lower `time_off` intervals would guarantee better success rates.

In order to study the trade-off between interaction latency and interaction success rate, we additionally produce cumulative latency distributions of interactions for various values of `lifetime` and `time_on` periods, and for a specific rate of the exponential intervals between subsequent gets. All failed interactions are pegged to the value `lifetime` concerning their latency. We have observed that lower `lifetime` periods produce markedly improved latency. Whereas, with higher levels of `lifetime` periods (typically PS/TS), we have noticed higher success rates, but also higher latency. On the other hand, increasing `time_on` intervals and decreasing `time_off` intervals also results in improved latency, but at the same time in better success rates.

In the same way and by using our produced results, an application designer may configure the `time_on` period of connectivity of a receiver entity, where we assume given values for `post` arrivals, `post` lifetimes and `get` frequencies. By selecting a specific value for `time_on`, the application designer is able to predict the mean success rate of the posts, along with a statistical upper bound for the latency of the same posts expressed with a certain probability. If the resulting values are not satisfactory, the application designer may increase the `time_on` period to achieve a higher success rate and/or lower latency.

Comparison with prototype implementation. In order to validate our simulation-based analysis (and further our underlying time model and formal analysis), we have implemented realistic GM interactions. Specifically, we employ: i) an implementation of the DPWS [145] CS middleware for GM interactions of very low `lifetime`; and (ii) an implementation of the JMS [122] PS middleware for GM interactions of a certain guaranteed `lifetime`. We apply the same settings as for our simulations. Note that in our implementation setup we have concurrent posts and queueing. This corresponds to an M/G/1/∞ queueing model; however, we make sure to avoid heavy load effects that would lead to high queueing delays in our experiments. We compare the results of simulated and real/monitored interaction success rates for various values of `post` lifetimes and `get` frequencies. The absolute deviation between the two is no more than 10%. This deviation may be attributed to data processing, transmission and queueing delays in the running of our prototype, which have been abstracted in our model.

As already pointed out, we present in the next section an extension of our model and analysis that takes into account queueing effects and other additional factors of realistic middleware interactions.

3.3 QoS analysis of mobile interactions across interaction paradigms

(Publications: [44–46])
(Ph.D. thesis of Georgios Bouloukakis, UPMC, 2017 [43]. Co-supervision with Valérie Issarny.)

In the previous section, we modeled in a unifying way the timing behavior of diverse system interactions where systems have varying functional and QoS semantics, in particular due to different interaction paradigms (CS, PS, DS and TS) and QoS configurations. Based on this modeling, we evaluated and compared the latencies (or end-to-end response times) and success rates of such system interactions. The considered QoS semantics concern data validity/availability as well as the intermittent availability of mobile data recipients for energy saving purposes or other application-related reason. In these semantics, we assumed that the effect of the end-to-end protocol infrastructure is negligible. In the present section, we introduce further QoS semantics by modeling the end-to-end protocol infrastructure of mobile applications. In particular, the performance of such applications may be constrained by queueing delays and message losses due to heavy traffic load, as well as due to low network bandwidth or even network disconnections occurring in the mobile protocol infrastructure.

In [60, 62, 72], the trade-off between interaction response times and success rates when using key IoT protocols is evaluated (e.g., for CoAP). However, the employed methods are protocol-specific and

do not cover the introduction of a new IoT protocol. Several existing efforts concerning the design and evaluation of mobile systems aim at satisfying QoS requirements under several constraints (e.g., intermittent availability, limited resources, etc.). The evaluation methods used are derived mainly from the field of Queueing Theory. For instance, 2-Dimensional (2-D) Markov chains and quasi-birth-death (QBD) processes have been used in [98,107,108,141] to model the changing connectivity of mobile users when offloading computation or data to the Cloud through either 3G or WiFi connections.

In this section, we model the performance of middleware protocols by relying on Queueing Network Models (QNMs) [22,97]. Based on QNMs, middleware nodes (clients, servers, brokers, etc.) are represented as queues, called service centers or queueing centers, and the exchanged messages as jobs served. QNMs enable the isolation and separate analysis of each service center from the rest of the network. The solution of the entire network can be then formed by combining these separate solutions. Queueing networks have been extensively applied to represent and analyze communication and computer systems and have proved to be simple and at the same time powerful tools for system designers with regard to system performance evaluation and prediction [89, 119]. In particular, queueing networks have simple analytical solutions and can achieve a favorable balance between accuracy and efficiency, when compared with other modeling approaches like Queueing Petri Nets and Performance Petri Nets, even if there is always a trade-off between modeling expressiveness and capacity of solving models of realistic size and complexity [94,104,137].

As part of our approach, we model and solve analytically in particular a service center that represents an intermittently connected data receiver. This is modeled explicitly as an *ON/OFF queueing center*. This further makes part of a set of different service centers that we identify for modeling end-to-end mobile interactions. We develop simulation models for these service centers. By employing further these service centers as building elements, we introduce a set of performance modeling patterns for the various interaction types supported by the CS, PS, DS and TS interaction paradigms; these are differentiated for reliable and unreliable mobile interactions. Furthermore, we combine these performance modeling patterns to model heterogeneous end-to-end mobile interactions; these involve different interaction paradigms interconnected via the VSB. We perform simulation-based analysis of these interactions.

By using our models and analyses, application designers are able to analyze and possibly configure certain QoS semantics (protocol QoS features, network and user connectivity, message `lifetime` periods, allocated system resources) in order to provide appropriate interaction response times and success rates in complex heterogeneous applications.

ON/OFF queueing center model for mobile interactions. To model intermittently connected mobile peers, we employ the intermittent (ON/OFF) queueing center. Besides Poisson arrivals with rate λ and exponentially distributed service times with rate μ, we assume that the server of the queueing center is subject to an ON-OFF procedure. More specifically, the server remains in the ON-state for exponential time with parameter θ_{ON} ($\theta_{\text{ON}} = 1/T_{\text{ON}}$), during which it serves messages (if any). Upon the expiration of this time, the server enters the OFF-state, during which it stops serving messages, also for exponential time with parameter θ_{OFF} ($\theta_{\text{OFF}} = 1/T_{\text{OFF}}$). Without loss of generality, we make the assumption that if T_{ON} expires and there is a message currently being served, the server interrupts its processing and will continue from the point it stopped in the next T_{ON} period.

We have elaborated an analytical solution for the mean response time of the ON/OFF queueing center. This is the time that a message remains in the queueing center (waiting in the queue and being served). Queueing centers with service interruptions or vacations and varying features have been a topic of research for several decades, with a variety of analytical and numerical solutions of the resulting Markov chains [7,64,112,138]. The interest of our approach is that it relies on common, mean value, assumptions, such as: i) the PASTA property, meaning Poisson messages encounter the mean queue upon arrival [1]; ii) the memoryless property of the exponential distribution; and iii) Little's Law. Additionally, we rely on queueing centers with multiple classes of customers and priority queueing [51].

In our solution, we introduce, besides the regular flow (class) of arriving messages with rate λ, a second, *virtual flow* λ_{OFF} of OFF messages that correspond to the T_{OFF} intervals of the server. More specifically, an OFF message arrives at the server exactly when the latter goes into its T_{OFF} interval. We represent the server's inactivity during the T_{OFF} interval with the virtual service time of the OFF message. To ensure this punctuality, we assume that an OFF message arrives in the system exactly at the beginning of a T_{OFF} interval and has *preemptive priority* over regular messages. Hence, it also reaches the server at the beginning of the T_{OFF} interval. This modeling allows mapping the ON/OFF queueing

center to one with continuous service. Hence, we specify our model as a queueing model with two classes of customers and preemptive priority of one class over the other [51]. The singularities in solving this model are the following:

- The flow of OFF messages is not Poisson: during the T_{OFF} interval no new OFF message is allowed to arrive. Thus, the PASTA property does not hold for this flow.
- The average arrival rate of the OFF flow takes two alternating rate values: 0 during T_{OFF} intervals and $1/T_{\mathrm{ON}}$ during T_{ON} intervals (exactly 1 OFF message arrives at the end of a T_{ON} interval).

We employ the ON/OFF queueing center in the following as a component inside the QNMs that we introduce for modeling end-to-end mobile interactions. Besides the ON/OFF queueing center and certainly the continuous service queueing center (M/M/1), we also employ two other models of queueing centers in our QNMs. The first one includes finite buffer capacity for the queue. The second one includes message expirations due to limited `lifetime`. In particular, we assume that a message may be checked for its residual `lifetime` just prior to its regular processing; expired messages exit the queueing center without regular processing. In a similar way, we consider that the request and response messages of a two-way synchronous interaction share a common *cumulative* `lifetime` equal to the `timeout` parameter that represents the interval for a completed interaction. This may be checked in the same way as for a single message `lifetime`. We apply the finite capacity and message expiration features individually or jointly on both continuous service and ON/OFF queues. We have developed simulation models for all these queueing centers; we provide more details about our simulator later in this section.

Performance modeling patterns for mobile interactions. We introduce *performance modeling patterns* for the four base interaction paradigms; actually, more finely, for their supported interaction types identified in Section 3.1. We include in our modeling the end-to-end protocol infrastructure, which we assumed ideal in Section 3.2, thus taking into account more realistic concerns of mobile interactions. In particular, we model the level of end-to-end communication reliability in the face of disconnections of mobile peers or intermediate protocol entities. Our modeling of Section 3.2 was at the level of the GM interaction types, abstracting the interaction paradigms. Being more detailed, the present performance modeling distinguishes among both interaction types and interaction paradigms.

Let us consider a direct one-way interaction between two mobile peers. In this interaction, a mobile sender produces messages generated by an application. The application disconnects from the network from time to time (e.g., for energy saving purposes or other user-related reason), and the produced messages are buffered until the next connection, upon which they are forwarded to the underlying middleware layer. The middleware layer transmits these messages over the network to the mobile receiver. The middleware layer may be shared with other senders (applications) running on the same host; hence, it may serve multiple message flows having possibly different remote destinations. At the other side, the middleware layer of the mobile receiver is able to receive messages from the network coming from multiple remote senders, including the mobile sender above. This middleware layer delivers the destined messages to the overlying application whenever the application gets connected (e.g., the user launches it). The middleware layer may be shared with other receivers (applications) running on the same host; hence, it distributes the messages it receives from the network to possibly multiple applications. As already mentioned, the middleware instances of the two mobile peers take care of sending and receiving the application messages over the network. Inside the network, disconnections may occur due to several reasons, such as wireless devices moving out of network coverage, router crash/reboot, etc. We identify these disconnections at a higher layer, as middleware disconnections, where effectively the end-to-end connectivity between the two middleware instances is broken.

We model the interaction between the two mobile peers as a QNM. This is depicted in Figure 3.4. We employ in our modeling the continuous service and intermittent (ON/OFF) queueing centers, with message expirations where necessary. Assuming finite buffer capacity for the queueing centers is optional, e.g., if heavy load effects and resource constraints need to be taken into account. We model the mobile sender as a sequence of two queueing centers. The first one (SQ1) represents the buffering of application messages by the application itself and their intermittent processing by the local middleware whenever the application is connected. The second one (SQ2) represents the buffering and transmission by the middleware of application messages on a possibly intermittent network connection. We model also the mobile receiver as a sequence of two queueing centers. The first one (RQ1) represents the buffering

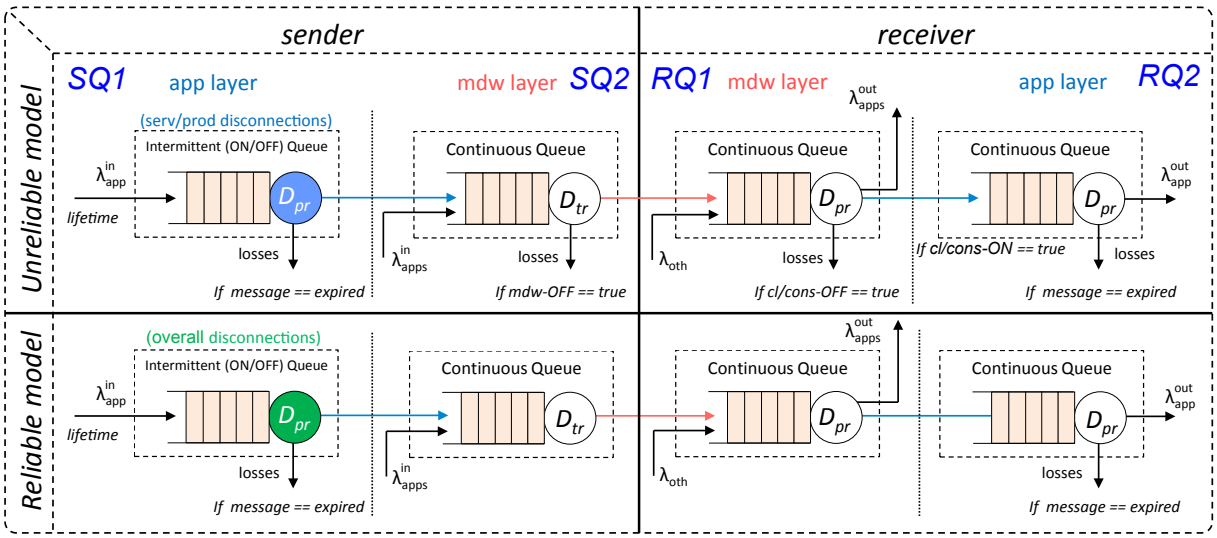


Figure 3.4: Performance modeling pattern for direct one-way interactions.

and processing by the middleware of application messages that are received from the network; messages destined for the application are delivered to it if it is connected. The second one (RQ2) represents the intermittent buffering and processing by the application (whenever it is connected) of its destined messages that are delivered by the local middleware. The complete QNM is built from the above four queueing centers in sequence: SQ1-SQ2-RQ1-RQ2.

We distinguish between two cases for this QNM, depending on the level of *communication reliability* supported by the modeled middleware protocol and more generally by the end-to-end protocol infrastructure. Existing middleware protocols can be categorized into: i) unreliable protocols, which do not provide any guarantees for the delivery of messages; and ii) reliable protocols, where the delivery of messages is either confirmed or reported as failed to the sender application. Unreliable middleware protocols typically build on top of UDP, which is itself an unreliable transport protocol. On the other hand, reliable middleware protocols usually build on top of TCP. TCP guarantees reliable message delivery by using acknowledgements, timeouts and retransmissions. Middleware protocols may further introduce their own additional reliability mechanisms.

We model an unreliable end-to-end interaction as follows. SQ1 is an ON/OFF queueing center to represent the intermittent connectivity of the sender application. SQ2, RQ1, RQ2 are continuous service queueing centers: any middleware/network disconnection or receiver application disconnection results in message loss but does not delay message transmission additionally (no retransmissions). Lost messages are represented by a side output flow of the queueing centers besides the delivered message flow. Additional message loss may result from message expiration; this is checked by the sender and receiver applications at SQ1 and RQ2.

We model a reliable end-to-end interaction as follows. SQ1 is again ON/OFF, but it now represents connectivity along the entire interaction path. The connectivity pattern of SQ1 results from the intersection of the connectivity patterns of the sender application, the middleware and the receiver application. The meaning of this is that the sender application is reliably informed about the successful or failed end-to-end transmission of a message. SQ2, RQ1, RQ2 are continuous-service: they represent only the interaction latency along the end-to-end path, but they include no message losses due to disconnections. The only message loss may result from message expiration; this is treated as in the unreliable case.

The above analyzed modeling of a direct one-way interaction between two mobile peers applies in the case of a CS or DS one-way interaction. In a similar way, we model an indirect one-way interaction through an intermediate protocol node, as in the case of a PS interaction where an event produced by a publisher reaches a subscriber through a possibly also mobile broker. In this case, the interaction can be split in two parts, publisher-broker interaction and broker-subscriber interaction, which may take place in a time-decoupled way. Furthermore, we model two-way interactions, where the two parts of the interaction have opposite directions and are modeled in sequence. In the case of the two-way synchronous interaction, the `timeout` parameter is used instead of the `lifetime` parameter in the way already described in the

previous. The complete set of our performance modeling patterns is as follows:

- *Reliable/unreliable CS one-way, two-way synchronous, two-way asynchronous*;
- *Reliable/unreliable PS one-way, two-way streaming*;
- *Reliable/unreliable DS one-way, two-way streaming*;
- *Reliable/unreliable TS one-way, two-way synchronous.*

This corresponds to the list of the interaction types identified for each interaction paradigm that we introduced in Section 3.1. These patterns can be reused inside bigger compositions, thus modeling the end-to-end performance of complex systems.

End-to-end performance modeling of heterogeneous interactions. We have leveraged our performance modeling patterns to model the end-to-end performance of services/Things interactions that employ heterogeneous middleware protocols possibly offering different levels of communication reliability.

We interconnect such services/Things through VSB, as introduced in Section 3.1. Accordingly, any such interconnection can be abstracted as a sequence of three GM connectors, where the two end connectors abstract the middleware protocols of the two interacting peers and the middle connector abstracts the employed VSB protocol. The bridging between each end connector and the middle connector is ensured by a BC. We map to each GM connector a specific performance modeling pattern by taking into account the interaction paradigm and further the interaction type that the specific protocol abstracted by GM follows, as well as its reliability and other QoS semantics. The resulting end-to-end model is the composition of the three selected patterns.

Moreover, when composing two patterns, we merge the application layers represented in the patterns at the point of their bridging. We possibly then refine the merged application layer: it should correspond to the conversion logic of the BC that performs the bridging. This application layer takes into account the constituent application layers (e.g., continuous-service or ON/OFF queues) but also the fact that a BC is assumed to be always connected. This assumption is desirable in order to provide seamless interoperability and can be achieved, e.g., by the deployment of BCs in the Cloud.

Performance simulator for mobile interactions. We have developed a simulator that implements the queueing center models and performance modeling patterns presented in the previous. We leverage our simulator for analytical model validation, as well as for creating simulation models that represent our proposed performance modeling patterns. Using the latter models, we perform statistical analysis and evaluate the trade-off between response times and success rates for interactions represented by the models. Our simulator, *MobileJINQS*[5], is an open-source library for building simulations encompassing constraints of mobile applications. MobileJINQS is an extension of JINQS, a Java simulation library for multi-class queueing networks [66]. JINQS provides a suite of primitives that allow developers to rapidly build simulations for a wide range of QNMs [97]. MobileJINQS retains the generic model specification power of JINQS, while it provides additional features of interest to mobile or other systems such as: i) `lifetime` limitations for messages entering a queueing network; ii) intermittently available (ON/OFF) queue servers representing the intermittent connectivity of mobile peers; iii) arrival and connectivity rates derived from real data traces; iv) ON/OFF queueing centers with buffers of finite capacity; and v) selective queue output flows and related sink nodes that collect messages lost due to middleware disconnections or message expirations.

ON/OFF queueing center model validation using real data traces. We have performed three experiments with MobileJINQS in order to validate our analytical model for the ON/OFF queueing center.

In our first experiment, we calculate a set of mean response time values with the analytical model. These are derived for a selected mean value of message service times, various mean durations of ON and OFF intervals, and different mean message arrival rates, from low ones to ones bringing the system close to saturation. At the same time, we have used MobileJINQS to implement the ON/OFF queueing center (with infinite queue capacity and no message expirations). Arrivals and service times for individual

[5] `http://xsb.inria.fr/d4d.php#mobilejinqs`

messages, as well as individual ON and OFF intervals are derived from the probability distributions that were assumed in our solution of the analytical model. We run the simulator by setting for these distributions the same mean values as for the analytical model calculations, and obtain steady state mean response times. We have plotted together the response times produced by the analytical and simulation models. The resulting curves are almost identical for low and medium arrival rates. We noticed some small differences for very high rates. This is acceptable, since the system is close to saturation at these rates.

We have performed a second validation of our analytical model for the ON/OFF queueing center by using message arrival workloads derived from real data traces. We relied on the *D4D dataset*, which we obtained from Orange Labs in the context of the *D4D challenge*[6] [44]. The D4D dataset contains Call Detail Records (CDRs) and SMS message sending records of users subscribed to Sonatel, the principal telecommunications provider of Senegal, which is linked to Orange. This data was collected over the whole country for a period of 50 weeks from January to December 2013.

From this dataset, we built data traces for each base station (or antenna) of the Sonatel mobile network. Each antenna trace provides the number of connections (emitted calls and SMS messages) made to the specific antenna by users per interval of 10 minutes and for the whole 50 week period. We assume that the resulting antenna input workload and related user access pattern to mobile telecommunication services can also be used as a representative message arrival workload for our ON/OFF queueing center modeling mobile interactions. More precisely, we map the number of connections N_{i} for each 10 min interval i at the selected antenna to an equal number of message arrivals N_{i} at the ON/OFF queueing center in the same time interval i. This mapping results in a non-homogeneous Poisson arrival process with rate parameter λ piecewise constant in each 10 min interval i and equal to $\lambda_{\mathrm{i}} = N_{\mathrm{i}}/10$ arrivals per min, where we cover the whole 50 week period. Finally, we include in our experiments an antenna of low load (calculated as the mean arrival rate over the whole period) and an antenna of high load among the set of antennas.

We extended our MobileJINQS model of the ON/OFF queueing center with non-homogeneous Poisson arrivals and parameterized the arrival process with the set of rates resulting from the data traces of the two selected antennas (low-load and high-load). We further parameterized the probability distributions for service times and ON and OFF intervals with selected parameter values. By running the simulator with the above settings, we obtain mean response times per interval of two weeks over the 50-week period for the high and low loads. Subsequently, we apply the same parameter values to the analytical model for each 10 min interval. We aggregate the produced mean response times per interval of two weeks over the 50-week period. Note that we made sure to avoid the saturation (per 10 min interval; if it was let run with this arrival rate) of the ON/OFF queueing center when selecting our parameters and given the rates of the non-homogeneous Poisson arrival process. Otherwise this would make the analytical model inapplicable. By comparing the plotted curves for the simulated and analytical response times, we noticed that the deviation between the two is no more than 5%. This shows that the analytical model can produce reliable response times for message arrival workloads that are not homogeneous and are derived from real data traces. Nevertheless, we made the assumption that the individual arrival rates of the 10 min intervals do not exceed the limit that would bring the ON/OFF queueing center to saturation (if it was let run with each one of these rates). Moreover, our produced mean response times are per interval of two weeks, which may be too coarse-grained. We intend to do a more fine-grained analysis including studying the transient behavior of the ON/OFF queueing center.

In our third validation experiment, we apply ON/OFF interval durations derived from real data traces. In particular, we have created a dataset concerning Internet connectivity of mobile users in the metro [16]. For this, we developed an Android application, named *Metro Cognition*, which aims at collecting connectivity data for metro passengers in Paris and Delhi. The application allows a user to identify her itinerary inside the metro system by entering her departure and destination metro stations. Then, data are collected during her journey. Every 30 seconds and/or each time the connectivity status changes a record is created along with timestamp information. At the end of the journey, the collected records are stored on a cloud server.

For practical reasons, we focused on two specific itineraries inside the Paris metro (from station "Cité Universitaire" to station "Dugommier", named CUD; and return, named DCU) for which we have collected sufficient amounts of data: 34 journeys of a total duration of 15.18 hours, where the average journey duration is 26.8 min, for CUD, and a bit lower numbers for DCU. Our data show that metro

[6] `http://www.d4d.orange.com/en/`

travelers lose and recover their Internet connection for intervals ranging from several seconds to 5 minutes. In average, connection intervals are 1.5 times longer than disconnection intervals. More specifically, we measured 146.38 sec and 96 sec in average for connection and disconnection intervals while traveling in the CUD itinerary. Numbers for DCU are 155.88 sec and 72.15 sec, respectively.

We consider that the Internet connectivity patterns we measured in the metro are very relevant for our ON/OFF queueing center modeling mobile interactions. Hence, we directly derive ON/OFF interval durations from the measured connection and disconnection intervals. As a first step, in our analysis, we tried to identify the best fit of our collected data to existing probability distributions. It was interesting to observe that the resulting complementary cumulative distribution functions (CCDFs) of connection and disconnection intervals fit best with exponential distributions. We then apply our connectivity data to our MobileJINQS model of the ON/OFF queueing center. We also apply two different service times and, for each, various arrival rates, from low ones to ones bringing the system close to saturation. We run the simulator and obtain steady state mean response times. At the same time, we apply the same parameter values to the analytical model, including the average ON/OFF values resulting from our dataset. We calculate mean response times with the analytical model. We have plotted and compared the response times provided by the analytical model and the trace-based simulation. Results match with high accuracy for low and medium arrival rates. For higher rates, there is a quite good match between the two with a maximum difference of about 10%. This is still acceptable accuracy for the analytical model, given that it relies on the assumption of exponential distributions for the ON/OFF intervals, which is an approximation for the ON/OFF intervals from the real data traces.

Performance analysis of mobile interactions. We have performed simulation experiments that demonstrate how an application designer can leverage our performance modeling patterns for analyzing mobile interactions that follow the various interaction types supported by the CS, PS, DS and TS interaction paradigms. Our approach here is similar to our simulation-based analysis of GM interactions in Section 3.2. However, as already pointed out, the performance modeling patterns allow a much more realistic modeling of mobile interactions than the time model of GM interactions of Section 3.2, by including queueing delays, network disconnections, communication reliability and other factors of the end-to-end protocol infrastructure. We develop simulation models in MobileJINQS that are parameterized with a variety of message arrival rates, message service times, as well as $T_{\text{ON}}/T_{\text{OFF}}$ periods and message `lifetime` periods. We demonstrate that varying these periods has a significant effect on the rate of successful interactions. Furthermore, we evaluate the trade-off between interaction success rates and interaction response times.

We have analyzed the reliable PS one-way interaction pattern, where an event produced by a mobile publisher is reliably transmitted to a mobile subscriber through a possibly also mobile broker. This pattern is depicted in Figure 3.5. In the reliable PS one-way pattern, losses occur only due to message expirations, i.e., there are no losses due to disconnections. We parameterize the QNM of the pattern as follows. We set a specific (Poisson) message arrival rate to the first queueing center of the QNM, and select proper (exponential) service times for all the queueing centers; these represent processing and transmission times of messages. We alternate among different `lifetime` periods for messages. Regarding the publisher-broker part of the interaction, there is one ON/OFF queueing center, which represents the overall connectivity along the interaction path between the publisher and the broker. This includes the connectivity of the publisher and the middleware, whereas we assume that the broker is fixed and reliable. Experiments are performed while varying the (exponential) T_{ON} and T_{OFF} periods of this queueing center. Similarly, regarding the broker-subscriber part of the interaction, there is one ON/OFF queueing center, which represents the overall connectivity between the broker and the subscriber. This includes the connectivity of the middleware and the subscriber. Experiments are performed while varying the (exponential) T_{ON} and T_{OFF} periods also of this queueing center. Using MobileJINQS, we perform around 700000 interactions for each experiment.

We have measured the rates of successful interactions for various values of message `lifetime` and $T_{\text{ON}}/T_{\text{OFF}}$ periods for both the publisher-broker and broker-subscriber parts of the interaction. As expected, increasing T_{ON} periods over T_{OFF} periods (of either part of the interaction) for individual `lifetime` values improves the interaction success rate. Similarly, increasing `lifetime` periods also results in higher success rates. In order to study the trade-off between interaction response times and interaction success rates, we have additionally produced cumulative response time distributions for various values of message `lifetime` and $T_{\text{ON}}/T_{\text{OFF}}$ periods for both parts of the interaction. We include response

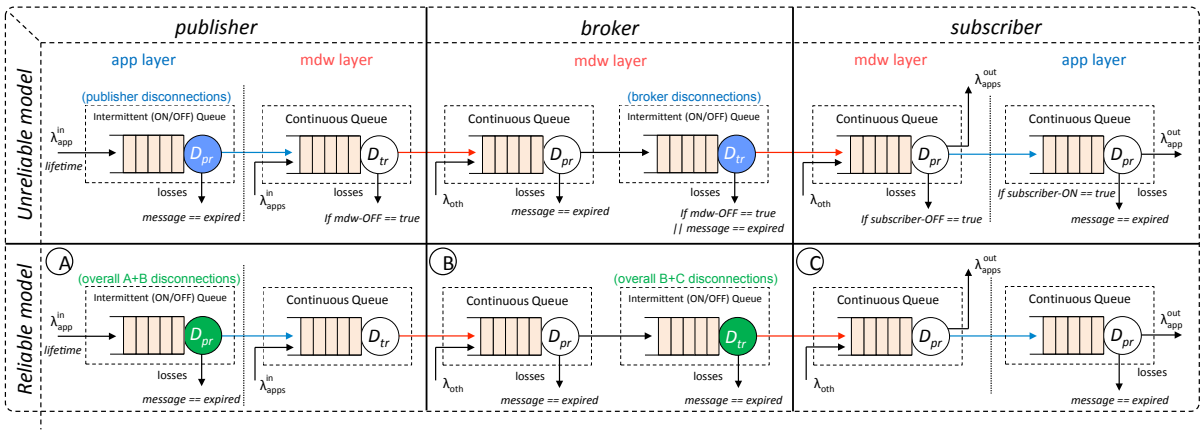


Figure 3.5: Performance modeling pattern for PS one-way interactions.

times only for successful interactions, i.e., response times lower than the `lifetime` period. We have remarked that lower `lifetime` periods produce clearly improved response times. Whereas, by increasing the `lifetime`, we get higher success rates, but with increased response times. On the other hand, increasing T_{ON} periods with respect to T_{OFF} periods results in both improved response times and higher success rates.

The results of the present analysis are similar to those of the simulation-based analysis of Section 3.2. This means that the abstract time model of GM interactions introduced there remains valid, at least when assuming a reliable protocol infrastructure. Nevertheless, besides providing the same general observations as the previous analysis, our analysis based on the performance modeling patterns can provide much more accurate quantitative results that take into account a rich set of features of the modeled real systems. By employing this analysis, an application designer is able to: (i) compare and select middleware protocols according to their QoS features, and (ii) configure certain system parameters (e.g., message lifetimes, application connection/disconnection periods, message service rates), in order to predict/ensure mean values and/or statistical bounds for the interaction success rates and interaction response times inside the designed system. Furthermore, by composing appropriate performance modeling patterns as introduced in the previous, an application designer is able to analyze, predict and possibly ensure the end-to-end performance of heterogeneous mobile interactions where the involved systems (services and Things) are interconnected via VSB.

Chapter 4

Conclusion and Outlook

This manuscript provided a synthetic view of my principal research efforts in these last years. These efforts focused on the study of mobile distributed systems that rely on the enabling features of service oriented computing. The presentation was structured around two major aspects of this area, namely Composition and Interoperability. The presented results include several enablers that fit into or complement the SOA architectural style. These enablers respond to the demanding requirements of the mobile environment and further its direct extension to the global Internet of Services and Things. My current and future research aims at generalizing and extending these results in the way discussed in the following.

Summarizing the properties of the mobile environment as discussed until now, we can state that this computing environment is characterized by high *uncertainty*, resulting from its openness and constant evolution. Uncertainty calls for designing mobile distributed systems that are able to run in a beforehand unknown, ever-changing context. Nevertheless, the complexity of such change cannot be tackled at system design-time. *Emergent mobile distributed systems* are systems that, due to their automated, dynamic, environment-dependent composition and execution, emerge in a possibly non-anticipated way and manifest emergent properties, i.e., both systems and their properties take their complete form only at runtime and may evolve afterwards. This contrasts with the typical software engineering process, where a system is finalized during its design phase.

Emergent mobile distributed systems present challenging research questions. My objective is to study and ensure fundamental distributed system properties, both functional and non-functional, such as (i) interoperability, (ii) realizability and trustworthiness in particular with respect to correctness against requirements, (iii) performance versus resource efficiency, and (iv) scalability, for emergent mobile distributed systems. Assurance of such properties is highly challenged by the dynamism, heterogeneity and uncertainty inherent in the emergent systems and their environment. Hence, my emphasis is on conditions and limitations for realistic emergent systems. I will focus on the following research directions:

Cross-layer impact on emergent properties. Functional and non-functional properties of mobile distributed systems are dependent on the application, middleware, network, and system resources layers, which are all highly dynamic and heterogeneous in today's global networking and computing environment. Hence, when studying emergent properties for such systems, none of these layers should be abstracted and taken for granted. My research will focus on addressing emergent properties in a cross-layer fashion. Such properties relate to application interfaces and interaction protocols, application data and data streams, middleware interaction paradigms and protocols, as well as application- and middleware-level QoS. Modeling and analysis of cross-layer properties will be based on formal multi-level abstractions and stepwise refinement between them.

Evolution of emergent properties in heterogeneous space and in time. Emergent mobile distributed systems arise as dynamic compositions of heterogeneous constituents (i.e., *heterogeneous space*) that may evolve over time. Global emergent properties, both functional and non-functional, result from the individual properties of the constituent systems. My objective is to study the *composability* of properties when heterogeneous systems are composed together, and their *replaceability* when a system is substituted by another system within the composition. The resulting modeling complexity is further exacerbated by uncertainty, since the exact functional and non-functional behavior of the constituent systems and of their environment cannot be known at design-time or can only be estimated. I intend to

deal with the modeling of emergent global properties over space and time, as well as with the modeling of global requirements, across heterogeneous systems.

Trustworthiness of emergent heterogeneous mobile distributed systems. Ensuring the trustworthiness of emergent mobile distributed systems is challenging due to their heterogeneity and other dimensions of uncertainty. I principally identify here trustworthiness as correctness against system requirements, functional and non-functional. Given the unplanned variability of emergent systems and their environment, requirements and even system invariants have to be adaptive, i.e., expressed as a range of acceptable possibilities for a system's behavior. In this context, my objective is to assure global system correctness that additionally accommodates the heterogeneous nature of the requirements of the individual constituent systems. Besides trustworthiness, I aim to assess more generally the feasibility of *realistic* emergent systems. This includes the effectiveness, performance and scalability of adaptation enactment in realistic computing and networking environments. It also concerns the ability of such systems to provide useful and satisfying service to their users, as opposed to degenerated service due to their emergence and execution uncertainty.

Last but not least and to demonstrate the generality of the identified research directions, we have been working lately on the interoperability and composition of computer-mediated social communication services, commonly found in widely used social networking platforms, across these different platforms [8, 9]. There, besides technical considerations of the systems, user-centered and social considerations need to be taken into account. All this opens up exciting new possibilities of research.

Bibliography

[1] Ivo Adan and Jacques Resing. *Queueing Theory*. Eindhoven University of Technology. Dep. of Mathematics and Computing Science, 2002.

[2] H. Al-Helal and R. Gamble. Introducing replaceability into web service composition. *IEEE Transactions on Services Computing*, 7(2):198–209, April 2014.

[3] Eyhab Al-Masri and Qusay H. Mahmoud. Discovering the best web service. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1257–1258, New York, NY, USA, 2007. ACM.

[4] Lachlan Aldred, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. *On the Notion of Coupling in Communication Middleware*, pages 1015–1033. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[5] Mohammad Alrifai, Thomas Risse, and Wolfgang Nejdl. A hybrid approach for efficient web service composition with end-to-end qos constraints. *ACM Trans. Web*, 6(2):7:1–7:31, June 2012.

[6] Mohammad Alrifai, Dimitrios Skoutas, and Thomas Risse. Selecting skyline services for qos-based web service composition. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 11–20, New York, NY, USA, 2010. ACM.

[7] Eitan Altman and Uri Yechiali. Analysis of customers' impatience in queues with server vacations. *Queueing Systems*, 52(4):261–279, Apr 2006.

[8] Rafael Angarita, Nikolaos Georgantas, and Valérie Issarny. USNB: Enabling Universal Online Social Interactions. In *IEEE International Conference on Collaboration and Internet Computing*, San Jose, United States, October 2017.

[9] Rafael Angarita, Nikolaos Georgantas, Cristhian Parra, James Holston, and Valérie Issarny. Leveraging the Service Bus Paradigm for Computer-mediated Social Communication Interoperability. In *International Conference on Software Engineering (ICSE), Software Engineering in Society (SEIS) Track*, Buenos Aires, Argentina, May 2017.

[10] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, June 2007.

[11] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[12] The OWL Coalition at the World Wide Web Consortium. *OWL Web Ontology Language*, 2004. W3C Recommendation, `http://www.w3.org/TR/owl-ref/`.

[13] E. Avilés-López and J.A. García-Macías. TinySOA: A Service-oriented Architecture for Wireless Sensor Networks. *Service Oriented Computing and Applications*, 3(2):99–108, 2009.

[14] Benjamin Aziz. A formal model and analysis of an iot protocol. *Ad Hoc Networks*, 36(Part 1):49 – 57, 2016.

[15] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[16] G. Bajaj, G. Bouloukakis, A. Pathak, P. Singh, N. Georgantas, and V. Issarny. Toward enabling convenient urban transit through mobile crowdsensing. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 290–295, Sept 2015.

[17] Andrew Banks and Rahul Gupta. MQTT Version 3.1.1. OASIS Standard, OASIS, October 2014. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`.

[18] L. Barakat, S. Miles, and M. Luck. Efficient correlation-aware service selection. In *2012 IEEE 19th International Conference on Web Services*, pages 1–8, June 2012.

[19] L. Baresi, N. Georgantas, K. Hamann, V. Issarny, W. Lamersdorf, A. Metzger, and B. Pernici. Emerging research themes in services-oriented systems. In *2012 Annual SRII Global Conference*, pages 333–342, July 2012.

[20] L. Baresi, C. Ghezzi, and L. Mottola. On accurate automatic verification of publish-subscribe architectures. In *IEEE Intl. Conf. on Software Engineering*, 2007.

[21] Adam Barker, Christopher D. Walton, and David Robertson. Choreographing web services. *IEEE Trans. on Services Computing*, 2:152–166, 2009.

[22] Forest Baskett, K Mani Chandy, Richard R Muntz, and Fernando G Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2):248–260, 1975.

[23] Françoise Baude, Imen Filali, Fabrice Huet, Virginie Legrand, Elton Mathias, Philippe Merle, Cristian Ruz, Reto Krummenacher, Elena Simperl, Christophe Hammerling, and Jean-Pierre Lorre. ESB Federation for Large-scale SOA. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 2459–2466, New York, NY, USA, 2010. ACM.

[24] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal 4.0. Technical report, Aalborg University, Denmark, 2006.

[25] Nebil Ben Mabrouk. *QoS-aware Service-Oriented Middleware for Pervasive Environments*. PhD Thesis, Université Pierre et Marie Curie - Paris VI, April 2012.

[26] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie Issarny. QoS-aware Service Composition in Dynamic Service Oriented Environments. In Brian F. Cooper Jean M. Bacon, editor, *Middleware 2009 - ACM/IFIP/USENIX, 10th International Conference*, Urbana, IL, United States, November 2009. Springer.

[27] Nebil Ben Mabrouk, Nikolaos Georgantas, and Valerie Issarny. Set-based Bi-level Optimisation for QoS-aware Service Composition in Ubiquitous Environments. In *Proceedings of the 22nd IEEE International Conference on Web Services (ICWS)*, New York, United States, June 2015.

[28] Nebil Ben Mabrouk, Nikolaos Georgantas, and Valérie Issarny. Multi-Objective Service Composition in Ubiquitous Environments with Service Dependencies. *International Journal of Services Computing (IJSC)*, 4, April 2016.

[29] Sonia Ben Mokhtar. *Semantic Middleware for Service-Oriented Pervasive Computing*. PhD Thesis, Université Pierre et Marie Curie - Paris VI, December 2007.

[30] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. Ad Hoc Composition of User Tasks in Pervasive Computing Environments. In *Software Composition*, pages 31–46, Edinburgh, United Kingdom, 2005.

[31] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. *Journal of Systems and Software*, 80(12):1941–1955, 2007.

[32] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valérie Issarny. Efficient Semantic Service Discovery in Pervasive Computing Environments. In *Middleware*, pages 240–259, Melbourne, Australia, 2006.

[33] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valérie Issarny, and Yolande Berbers. EASY: Efficient semAntic Service discoverY in pervasive computing environments with QoS and context support. *Journal of Systems and Software*, 81(5):785–808, 2007.

[34] A. Bennaceur and V. Issarny. Automated synthesis of mediators to support component interoperability. *IEEE Transactions on Software Engineering*, 41(3):221–240, March 2015.

[35] Amel Bennaceur, Emil Andriescu, Roberto Speicys Cardoso, and Valérie Issarny. A Unifying Perspective on Protocol Mediation: Interoperability in the Future Internet. *Journal of Internet Services and Applications*, page 14, 2015.

[36] Amel Bennaceur, Gordon Blair, Franck Chauvel, Nikolaos Georgantas, Paul Grace, Falk Howar, Paola Inverardi, Valérie Issarny, Massimo Paolucci, Animesh Pathak, Romina Spalazzese, B. Steffen, Bertrand Souville, and Huang Gang. Towards an architecture for runtime interoperability. In *ISoLA 2010 - 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, pages 206–220, Crete, Greece, 2010.

[37] Amel Bennaceur and Valérie Issarny. Layered Connectors: Revisiting the Formal Basis of Architectural Connection for Complex Distributed Systems. In *ECSA'14 - The 8th European Conference on Software Architecture*, Vienna, Austria, August 2014. Springer.

[38] Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 613–624. VLDB Endowment, 2005.

[39] Martin Berger and Kohei Honda. The two-phase commitment protocol in an extended ĬĂ-calculus. *Electronic Notes in Theoretical Computer Science*, 39(1):21 – 46, 2003. EXPRESS'00, 7th International Workshop on Expressiveness in Concurrency (Satellite Workshop from CONCUR 2000).

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

[41] Gordon Blair, Amel Bennaceur, Nikolaos Georgantas, Paul Grace, Valérie Issarny, Vatsala Nundloll, and Massimo Paolucci. The Role of Ontologies in Emergent Middleware: Supporting Interoperability in Complex Distributed Systems. In Fabio Kon and Anne-Marie Kermarrec, editors, *12th International Middleware Conference (MIDDLEWARE)*, volume LNCS-7049 of *Middleware 2011*, pages 410–430, Lisbon, Portugal, December 2011. Springer. Part 8: Run-Time (Re)configuration and Inspection.

[42] Gordon Blair, Massimo Paolucci, Paul Grace, and Nikolaos Georgantas. Interoperability in Complex Distributed Systems. In Marco Bernardo and Valerie Issarny, editors, *11th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Connectors for Eternal Networked Software Systems*. Springer, 2011.

[43] Georgios Bouloukakis. *Enabling Emergent Mobile Systems in the IoT: from Middleware-layer Communication Interoperability to Associated QoS Analysis*. PhD Thesis, Université Pierre et Marie Curie - Paris VI, August 2017.

[44] Georgios Bouloukakis, Rachit Agarwal, Nikolaos Georgantas, Animesh Pathak, and Valerie Issarny. Leveraging CDR datasets for Context-Rich Performance Modeling of Large-Scale Mobile Pub/Sub Systems. In *WiMob 2015 - 11th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Abu Dhabi, United Arab Emirates, October 2015.

[45] Georgios Bouloukakis, Nikolaos Georgantas, Ajay Kattepur, and Valérie Issarny. Timeliness Evaluation of Intermittent Mobile Connectivity over Pub/Sub Systems. In *ICPE 2017 - 8th ACM/SPEC International Conference on Performance Engineering* , ICPE 2017, L'Aquila, Italy, April 2017.

[46] Georgios Bouloukakis, Ioannis Moscholios, Nikolaos Georgantas, and Valérie Issarny. Performance Modeling of the Middleware Overlay Infrastructure of Mobile Things. In *IEEE International Conference on Communications*, Paris, France, May 2017.

[47] Nadia Busi and Gianluigi Zavattaro. A Process Algebraic View of Shared Dataspace Coordination. *The Journal of Logic and Algebraic Programming*, 75(1):52–85, 2008.

[48] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281 – 308, 2004.

[49] Yves Caseau. Efficient handling of multiple inheritance hierarchies. In *OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 271–287, New York, NY, USA, 1993. ACM Press.

[50] Matteo Ceriotti, Amy L. Murphy, and Gian Pietro Picco. Data Sharing vs. Message Passing: Synergy or Incompatibility?: An Implementation-driven Case Study. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 100–107, New York, USA, 2008.

[51] Wei Chang. Preemptive priority queues. *Operations research*, 13(5):820–827, 1965.

[52] David Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[53] N. Chen, N. Cardozo, and S. Clarke. Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, PP(99):1–1, 2017.

[54] Y. Chen, J. Huang, C. Lin, and J. Hu. A partial selection methodology for efficient qos-aware service composition. *IEEE Transactions on Services Computing*, 8(3):384–397, May 2015.

[55] P. ChL'tel, J. Malenfant, and I. Truck. Qos-based late-binding of service invocations in adaptive business processes. In *2010 IEEE International Conference on Web Services*, pages 227–234, July 2010.

[56] The DAML Services Coalition. Bringing semantics to web services: The owl-s approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC'04)*, July 2004.

[57] The OWL-S Coalition. *OWL-S: Semantic Markup for Web Services*, 2004. W3C Member Submission, `http://www.w3.org/Submission/OWL-S`.

[58] Diana Comes, Harun Baraki, Roland Reichle, Michael Zapf, and Kurt Geihs. *Heuristic Approaches for QoS-Based Service Selection*, pages 441–455. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[59] Ion Constantinescu and Boi Faltings. Efficient matchmaking and directory services. In *Proceedings of the IEEE International Conference on Web Intelligence (WI'03)*, 2003.

[60] Niccolò De Caro, Walter Colitti, Kris Steenhaut, Giuseppe Mangino, and Gianluca Reali. Comparison of two lightweight protocols for smartphone-based sensing. In *2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, pages 1–6. IEEE, 2013.

[61] Stein Desmet, Bruno Volckaert, S VAN ASSCHE, D VAN DER WEKEN, Bart Dhoedt, and Filip De Turck. Throughput evaluation of different enterprise service bus approaches. In *Proceedings of SERP2007, the 2007 International Conference on Software Engineering Research & Practice (part of the 2007 World Congress in Computer Science, Computer Engineering, and Applied Computing)*, pages 378–384, 2007.

[62] Lars Durkop, Bjorn Czybik, and Jurgen Jasperneite. Performance evaluation of M2M protocols over cellular networks in a lab environment. In *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, pages 70–75. IEEE, 2015.

[63] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[64] Awi Federgruen and Linda Green. Queueing systems with service interruptions. *Oper. Res.*, 34(5):752–768, October 1986.

[65] Ian Fette and Alexey Melnikov. The WebSocket Protocol. RFC 6455, IETF, December 2011. `https://www.rfc-editor.org/info/rfc6455`.

[66] Tony Field. JINQS: An extensible library for simulating multiclass queueing networks, v1. 0 user guide, 2006.

[67] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[68] J. G. Pereira Filho and M. van Sinderen. Web service architectures - semantics and context-awareness issues in web services platforms. Technical report, Telematica Instituut, 2003.

[69] Emna Fki, Said Tazi, and Khalil Drira. Automated and flexible composition based on abstract services for a better adaptation to user intentions. *Future Generation Computer Systems*, 68(Supplement C):376 – 390, 2017.

[70] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*, ASE'03, pages 152–161, Piscataway, NJ, USA, 2003. IEEE Press.

[71] Freeman, E. and Arnold, K. and Hupfer, S. *JavaSpaces Principles, Patterns, and Practice*. Addison-Wesley Longman Ltd. Essex, UK, UK, 1999.

[72] K. Fysarakis, I. Askoxylakis, O. Soultatos, I. Papaefstathiou, C. Manifavas, and V. Katos. Which IoT Protocol? Comparing Standardized Approaches over a Common M2M Application. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7, Dec 2016.

[73] Nikolaos Georgantas, Sonia Ben Mokhtar, Ferda Tartanoglu, and Valérie Issarny. Semantics-Aware Services for the Mobile Computing Environment. In Rogerio de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems III*, pages 1–35. Springer, 2005.

[74] Nikolaos Georgantas, Georgios Bouloukakis, Sandrine Beauche, and Valérie Issarny. Service-oriented Distributed Applications in the Future Internet: The Case for Interaction Paradigm Interoperability. In Kung-Kiu Lau, Winfried Lamersdorf, and Ernesto Pimentel, editors, *ESOCC 2013 - European Conference on Service-Oriented and Cloud Computing*, volume 8135 of *Lecture Notes in Computer Science*, pages 134–148, Malaga, Spain, September 2013. Springer.

[75] Çagdaş Evren Gerede, Richard Hull, Oscar H. Ibarra, and Jianwen Su. Automated composition of e-services: Lookaheads. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 252–262, New York, NY, USA, 2004. ACM.

[76] Robert Godfrey, David Ingham, and Rafael Schloming. Advanced Message Queuing Protocol (AMQP) Version 1.0 Part 0: Overview. OASIS Standard, OASIS, October 2012. `http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html`.

[77] Paul Grace, Gordon S. Blair, and Sam Samuel. A Reflective Framework for Discovery and Interaction in Heterogeneous Mobile Environments. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(1):2–14, 2005.

[78] I. Graja, N. Guermouche, A. H. Kacem, and K. Drira. An approach for multiple-instance based service composition. In *2015 IEEE International Conference on Services Computing*, pages 435–442, June 2015.

[79] D. Grigori, J. C. Corrales, M. Bouzeghoub, and A. Gater. Ranking bpel processes for service discovery. *IEEE Transactions on Services Computing*, 3(3):178–192, July 2010.

[80] Michael Grüninger, Richard Hull, and Sheila McIlraith. A short overview of FLOWS: A first-order logic ontology of web services. *IEEE Data Eng. Bull.*, 31:3–7, 01 2008.

[81] Gudgin, M. and Hadley, M. and Mendelsohn, N. and Moreau, J.J. and Nielsen, H.F. and Karmarkar, A. and Lafon, Y. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, W3C, April 2007. `http://www.w3.org/TR/soap12-part1/`.

[82] Dominique Guinard, Stamatis Karnouskos, Vlad Trifa, Bettina Dober, Patrik Spiess, and Domnic Savio. Interacting with the SOA-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Trans. on Services Computing*, 3:223–235, 2010.

[83] Fei He, Luciano Baresi, Carlo Ghezzi, and Paola Spoletini. Formal analysis of publish-subscribe systems by probabilistic timed automata. In *Formal Techniques for Networked and Distributed Systems (FORTE)*, pages 247–262, 2007.

[84] S. Y. Hwang, E. P. Lim, C. H. Lee, and C. H. Chen. Dynamic web service selection for reliable web service composition. *IEEE Transactions on Services Computing*, 1(2):104–116, April 2008.

[85] Valérie Issarny, Georgios Bouloukakis, Nikolaos Georgantas, and Benjamin Billet. Revisiting Service-oriented Architecture for the IoT: A Middleware Perspective. In *14th International Conference on Service Oriented Computing (ICSOC)*, Banff, Alberta, Canada, October 2016.

[86] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadis, Marco Autili, Marco Aurelio Gerosa, and Amira Ben Hamida. Service-Oriented Middleware for the Future Internet: State of the Art and Research Directions. *Journal of Internet Services and Applications*, 2(1):23–45, May 2011.

[87] W. Jiang, S. Hu, and Z. Liu. Top k query for qos-aware automatic service composition. *IEEE Transactions on Services Computing*, 7(4):681–695, Oct 2014.

[88] Jun Jin, Yu Zhang, Yuanda Cao, Xing Pu, and Jiaxin Li. *ServiceStore: A Peer-to-Peer Framework for QoS-Aware Service Composition*, pages 190–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[89] Debessay Fesehaye Kassa, Klara Nahrstedt, and Guijun Wang. Analytical models of short-message reliability in mobile wireless networks. In *Proc. of the 14th ACM Intl. Conf. on Modeling, analysis and simulation of wireless and mobile systems*, pages 369–376. ACM, 2011.

[90] Ajay Kattepur, Nikolaos Georgantas, Georgios Bouloukakis, and Valerie Issarny. Analysis of Timing Constraints in Heterogeneous Middleware Interactions. In *ICSOC'15 - International Conference on Service Oriented Computing*, Goa, India, November 2015.

[91] A. Klein, F. Ishikawa, and S. Honiden. Sanga: A self-adaptive network-aware approach to service composition. *IEEE Transactions on Services Computing*, 7(3):452–464, July 2014.

[92] J. Kopeck, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, 11(6):60–67, Nov 2007.

[93] Mariya Koshkina and Franck van Breugel. Verification of business processes for web services. Technical Report CS-2003-11, Department of Computer Science, York University, Canada, 2003.

[94] S. Kounev. Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, July 2006.

[95] Andreas Krall, Jan Vitek, and Nigel Horspool. Near optimal hierarchical encoding of types. In Mehmet Aksit and Satoshi Matsuoka, editors, *11th European Conference on Object Oriented Programming (ECOOP'97)*, pages 128–145, Finland, 1997. Springer.

[96] Marta Kwiatkowska, Gethin Norman, and David Parker. *PRISM 4.0: Verification of Probabilistic Real-Time Systems*, pages 585–591. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[97] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. *Quantitative system performance: computer system analysis using queueing network models.* Prentice-Hall, Inc., 1984.

[98] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong. Mobile data offloading: How much can wifi deliver? *IEEE/ACM Transactions on Networking*, 21(2):536–550, April 2013.

[99] Jing Li, Yongwang Zhao, Min Liu, Hailong Sun, and Dianfu Ma. An adaptive heuristic approach for distributed qos-based service composition. In *The IEEE symposium on Computers and Communications*, pages 687–694, June 2010.

[100] D. Liu, Z. Shao, C. Yu, and G. Fan. A heuristic qos-aware service selection approach to web service composition. In *2009 Eighth IEEE/ACIS International Conference on Computer and Information Science*, pages 1184–1189, June 2009.

[101] Jinshan Liu and Valerie Issarny. Qos-aware service location in mobile ad-hoc networks. In *Proceedings of IEEE International Conference on Mobile Data Management (MDM'04)*, pages 224–235, 2004.

[102] Shalil Majithia, David W. Walker, and W. A. Gray. A framework for automated service composition in service-oriented architecture. In *1st European Semantic Web Symposium*, 2004.

[103] Axel Martens. *Analyzing Web Service Based Business Processes*, pages 19–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[104] Tomáş Martinec, Lukáş Marek, Antonín Steinhauser, Petr Tůma, Qais Noorshams, Andreas Rentschler, and Ralf Reussner. Constructing performance model of jms middleware platform. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 123–134, New York, NY, USA, 2014. ACM.

[105] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. *Task Computing – The Semantic Web Meets Pervasive Computing*, volume 2870 of *Lecture Notes in Computer Science*, pages 866–881. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[106] S. A. McIlraith, T. C. Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, Mar 2001.

[107] F. Mehmeti and T. Spyropoulos. Is it worth to be patient? analysis and optimization of delayed mobile data offloading. In *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, pages 2364–2372, April 2014.

[108] Fidan Mehmeti and Thrasyvoulos Spyropoulos. Performance analysis of "on-the-spot" mobile data offloading. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 1577–1583. IEEE, 2013.

[109] D. A. Menasce. Composing web services: A qos view. *IEEE Internet Computing*, 8(6):88–90, Nov 2004.

[110] S. B. Mokhtar, N. Georgantas, and V. Issarny. Cocoa : Conversationbased service composition for pervasive computing environments. In *2006 ACS/IEEE International Conference on Pervasive Services*, pages 29–38, June 2006.

[111] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valérie Issarny. *Ad Hoc Composition of User Tasks in Pervasive Computing Environments*, pages 31–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[112] Philippe Nain. Queueing systems with service interruptions: An approximation model. *Performance Evaluation*, 3(2):123 – 129, 1983.

[113] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[114] M. Pantazoglou and A. Tsalgatidou. A generic query model for the unified discovery of heterogeneous services. *IEEE Transactions on Services Computing*, 6(2):201–213, April 2013.

[115] Massimo Paolucci, Takahiro Kawamura, Terry R.Payne, and Katia Sycara. Semantic matching of Web services capabilities. *Lecture Notes in Computer Science*, 2342:333–347, 2002.

[116] 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, Nov 2007.

[117] Mike P. Papazoglou and Willem-Jan Heuvel. Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, 16:389–415, July 2007.

[118] Peter Pietzuch, David Eyers, Samuel Kounev, and Brian Shand. Towards a Common API for Publish/Subscribe. In *Proceedings of the 2007 Inaugural International Conference on Distributed Event-based Systems*, pages 152–157, New York, USA, 2007. ACM.

[119] Thadpong Pongthawornkamol, Klara Nahrstedt, and Guijun Wang. Probabilistic qos modeling for reliability/timeliness prediction in distributed content-based publish/subscribe systems over best-effort networks. In *Proc. of the 7th Intl. Conf. on Autonomic computing*, pages 185–194. ACM, 2010.

[120] Davy Preuveneers and Yolande Berbers. Prime numbers considered useful: Ontology encoding for efficient subsumption testing. Tech. Rep. CW464, Department of Computer Science, Katholieke Universiteit Leuven, Belgium, 2006.

[121] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, and Koenraad De Bosschere. Towards an extensible context ontology for ambient intelligence. In *EUSAI*, pages 148–159, 2004.

[122] Mark Richards, Richard Monson-Haefel, and David A. Chappell. *Java Message Service*. O'Reilly, second edition, 2009.

[123] P. Rodriguez-Mier, M. Mucientes, and M. Lama. Hybrid optimization algorithm for large-scale qos-aware service composition. *IEEE Transactions on Services Computing*, 10(4):547–559, July 2017.

[124] 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, July 2016.

[125] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel. Web service modeling ontology. *Appl. Ontol.*, 1(1):77–106, January 2005.

[126] Z. Shen and J. Su. Web service discovery based on behavior signatures. In *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, volume 1, pages 279–286 vol.1, July 2005.

[127] Evren Sirin, Bijan Parsia, and James Hendler. Template-based composition of semantic web services. In *AAAI Fall Symposium on Agents and the Semantic Web*, 2005.

[128] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Adding owl-s to uddi, implementation and throughput. In *Proceedings of the Workshop on Semantic Web Service and Web Process Composition*, 2004.

[129] T. G. Stavropoulos, S. Andreadis, N. Bassiliades, D. Vrakas, and I. Vlahavas. The tomaco hybrid matching framework for sawsdl semantic web services. *IEEE Transactions on Services Computing*, 9(6):954–967, Nov 2016.

[130] C. Surianarayanan and G. Ganapathy. An approach to computation of similarity, inter-cluster distance and selection of threshold for service discovery using clusters. *IEEE Transactions on Services Computing*, 9(4):524–536, July 2016.

[131] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[132] Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27–46, 2003.

[133] David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In *Proceedings of the first Semantic Web Working Symposium, (SWWS)*, pages 447–461, 2001.

[134] K. Ueno and M. Tatsubori. Early capacity testing of an enterprise service bus. In *2006 IEEE International Conference on Web Services (ICWS'06)*, pages 709–716, Sept 2006.

[135] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.

[136] Sebastien Verel, Arnaud Liefooghe, and Clarisse Dhaenens. Set-based multiobjective fitness landscapes: A preliminary study. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 769–776, New York, NY, USA, 2011. ACM.

[137] Mary Vernon, John Zahorjan, and Edward D. Lazowska. A comparison of performance Petri nets and queueing network models. Tech. Rep. 669, Department of Computer Science, University of Wisconsin-Madison, USA, 1986.

[138] Kuo-Hsiung Wang and Ying-Chung Chang. Cost analysis of a finite m/m/r queueing system with balking, reneging, and server breakdowns. *Mathematical Methods of Operations Research*, 56(2):169–180, Nov 2002.

[139] Libor Waszniowski, Jan Krakora, and Zdenek Hanzalek. Case study on distributed and fault tolerant system modeling based on timed automata. *The Journal of Systems and Software*, 82:1678–1694, 2009.

[140] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. In *e-Technology, e-Commerce and e-Service, 2004. EEE '04. 2004 IEEE International Conference on*, pages 359–368, March 2004.

[141] Huaming Wu and Katinka Wolter. Tradeoff analysis for mobile cloud offloading based on an additive energy-performance metric. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, pages 90–97, 2014.

[142] Daniel Wutke, Daniel Martin, and Frank Leymann. Facilitating Complex Web Service Interactions through a Tuplespace Binding. In *Proceedings of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 275–280, 2008.

[143] Amy Moormann Zaremski and Jeannette M. Wing. Signature matching: a tool for using software libraries. *ACM Transactions on Software Engineering and Methodology*, 4(2):146–170, 1995.

[144] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 1997.

[145] E. Zeeb, A. Bobek, H. Bohn, and F. Golatowski. Service-oriented architectures for embedded systems using devices profile for web services. In *Advanced Information Networking and Applications Workshops, 2007, AINAW '07. 21st International Conference on*, volume 1, pages 956–963, May 2007.

[146] H. Zheng, W. Zhao, J. Yang, and A. Bouguettaya. Qos analysis for web service compositions with complex structures. *IEEE Transactions on Services Computing*, 6(3):373–386, July 2013.

[147] Yoav Zibin and Joseph Yossi Gil. Efficient subtyping tests with pq-encoding. In *Proceedings of the 16th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '01, pages 96–107, New York, NY, USA, 2001. ACM.

[148] E. Zitzler, L. Thiele, and J. Bader. On set-based multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 14(1):58–79, Feb 2010.