COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support

Sonia Ben Mokhtar, Nikolaos Georgantas, Valérie Issarny

COCOA: COnversation-based Service COmposition in PervAsive Computing Environments with QoS Support

Sonia Ben Mokhtar, Nikolaos Georgantas and Valérie Issarny

INRIA Rocquencourt,
Domaine de Voluceau,
BP 105, 78153 Le Chesnay Cedex, France
{*Sonia.Ben_Mokhtar,Nikolaos.Georgantas,Valerie.Issarny*}*@inria.fr*

Abstract

Pervasive computing environments are populated with networked services, i.e., autonomous software entities, providing a number of functionalities. One of the most challenging objectives to be achieved within these environments is to assist users in realizing tasks that integrate on the fly functionalities of the networked services opportunely according to the current pervasive environment. Towards this purpose, we present COCOA, a solution for COnversation-based service COmposition in pervAsive computing environments with QoS support. COCOA provides COCOA-L, an OWL-S based language for the semantic, QoS-aware specification of services and tasks, which further allows the specification of services and tasks conversations. Moreover, COCOA provides two mechanisms: COCOA-SD for the QoS-aware semantic service discovery and COCOA-CI for the QoS-aware integration of service conversations towards the realization of the user task's conversation. The distinctive feature of COCOA is the ability of integrating on the fly the conversations of networked services to realize the conversation of the user task, by further meeting the QoS requirements of user tasks. Thereby, COCOA allows the dynamic realization of user tasks according to the specifics of the pervasive computing environment in terms of available services and by enforcing valid service consumption.

1 Introduction

Pervasive computing environments are populated with networked services, i.e., autonomous software entities, providing a number of functionalities. One of the most challenging objectives to be achieved within these environments is to assist users in realizing tasks that integrate functionalities of the networked services [28], so that tasks may be requested anytime, anywhere, and realized on the fly according

to the specifics of the pervasive computing environment. To illustrate the kind of situations that we expect to make commonplace through our research, we present the following scenario (see Figure 1):
"...Today, Jerry is going to travel by train from Paris to London, where he is going to give a talk in a working seminar. At the train station, Jerry has the privilege of waiting in the V.I.P. room. In this room, besides the wonderful French buffet, a number of digital services are available, among which a streaming service used to stream digital resources on users portable devices, and a large flat screen that continuously disseminates news. Today, exceptionally, Jerry has arrived early at the train station. When he enters the V.I.P. room, nobody is there. He decides to watch a movie while waiting for his train departure. Jerry uses the e-movie application *that he has on his PDA, to which he gives the title of the movie that he wants to watch. This* e-movie application *is able to discover video servers as well as display devices available in the reach of the user and to select the most appropriate device. More precisely, if a larger screen than the user's PDA screen is found in the user's reach, and if the user's context allows it (e.g., nobody else is in the same room), this application displays the movie on that screen. Furthermore, if the user's context changes (e.g., the user leaves the room or a person enters in the room), the application is able to transfer the video stream to the user's PDA. When the train arrives at the station, Jerry gets on the train and continues to watch the movie on his PDA until the train departure..."*.

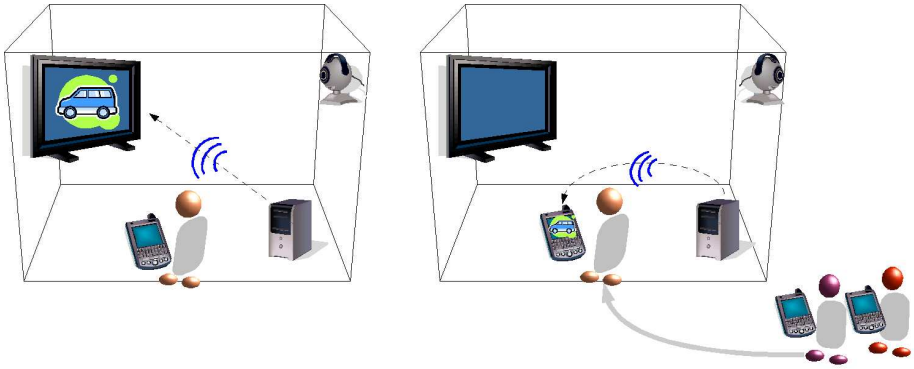

Fig. 1. Scenario

Hardware resources like displays, and software resources like multimedia streaming servers constitute networked capabilities that may conveniently be composed to realize *user tasks* enabling, for instance, the entertainment of nomadic users. Still, developing pervasive applications as user tasks that benefit from the open networking environment raises major software engineering challenges. Functional capabilities accessible in the specific networked environment must be abstracted in an adequate way so that applications may specify declaratively required functional capabilities for which concrete instances are to be retrieved on the fly. Furthermore, consumption of networked capabilities shall be achieved in a way that guarantees correctness of the application, both functionally and non-functionally. Another key requirement to indeed enable pervasive applications is for the network to be truly

open, with integration of most networked resources, without being unduly selective regarding hosted software and hardware platforms. Specifically, the pervasive computing environment shall be able to integrate most networked resources, further allowing the dynamic composition of applications out of capabilities provided by resources, when applications get either requested by users or proactively provided by the environment.

The Service-Oriented Architecture (SOA) paradigm, and its associated technologies such as Web services, appears as the right paradigm to engineer pervasive applications. Functional capabilities provided by networked resources may conveniently be abstracted as services. Specifically, a pervasive service corresponds to an autonomous networked entity, which provides a set of capabilities. A service capability then corresponds to either a primitive operation of the service or a process composing a number of operations (also referred to as conversation) [20]. Consumption of services by client applications (which may themselves realize more complex services available on the network) further requires service clients and providers to agree on both the functional and non-functional semantics of capabilities, so that they can integrate and interact in a way that guarantees dependable service provisioning and consumption. Such an agreement may be carried out at the syntactic level, assuming that clients and providers use a common service description syntax for denoting, besides service access protocols, as well, service semantics. This assumption is actually made by most software platforms for pervasive computing (e.g., Gaia [26], Aura [28], WSAMI [14]). However, such vision based on the strong assumption that service developers and clients describe services with identical terms worldwide, is hardly achievable in open pervasive environments. This raises the issue of *syntactic heterogeneity* of service descriptions. A promising approach towards addressing syntactic heterogeneity relies on semantic modeling of the services' functional and non-functional features. This concept underpins the Semantic Web [6]. Combined with Semantic Web technologies [1] , notably ontologies, for the semantic description of the services' functional and non-functional features, Web services can be automatically and unambiguously discovered and consumed in open pervasive computing environments. Specifically, ontology-based semantic reasoning enables discovering networked services whose published provided functionalities match a required functionality, even if there is no syntactic conformance between them. A number of research efforts have been conducted in the area of semantic Web service specification, which have led to the development of various semantic service description languages, e.g., OWL-S [2] , WSDL-S [3] , WSMO [4] , FLOWS [5] . Among these efforts OWL-S, which is based on

[1] Semantic Web: http://www.w3.org/2001/sw/
[2] OWL-S: http://www.daml.org/services/owl-s
[3] WSDL-S: http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/
[4] WSMO: http://www.wsmo.org/
[5] FLOWS: http://www.daml.org/services/swsf/1.0/overview/

the Web Ontology Language (OWL) [6] , a W3C recommendation, presents a number of attracting features. Indeed, OWL-S supports the concise specification of service functional capabilities in the service profile on the one hand, as well as the detailed specification of the corresponding service conversations on the other hand, which in turn provides a basis for Web service composition.

Building upon semantic Web services, and particularly OWL-S, we present COCOA, a solution for QoS-aware COnversation-based service COmposition in pervAsive computing environments. COCOA allows the dynamic realization of user tasks from networked services available in the pervasive computing environment. A preliminary effort for defining COCOA has been presented in [20]. In this article, we present the extension of COCOA with support of Quality of Service (QoS). COCOA is part of a larger effort on the development of an interoperable middleware for pervasive computing environments investigated in the IST Amigo project [7] . COCOA is composed of three major parts. First, COCOA-L, is an OWL-S based language for semantic specification of services and tasks in pervasive environments. COCOA-L allows the specification of requested and advertised service capabilities, service conversations, as well as service QoS properties. Second, COCOA-SD realizes the discovery and selection of networked services candidate to the composition. Thanks to the semantic reasoning enabled by the use of ontologies, COCOA-SD enables a thorough matching of service functionalities complemented with QoS-based matching. Finally, COCOA-CI performs dynamic QoS-aware composition of the selected services towards the realization of the target user task. The distinctive feature of COCOA-CI is the integration of services modeled as conversations, to realize a user task also modeled as a conversation. This provides a mean to deal with the diversity of services in pervasive computing environments. Indeed, as shown in Figure 2, integrating service conversations for the realization of a user task's conversation enables the same user task to be performed in different environments by means of several composition schemes (e.g., 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). Thus, the realization of the task's conversation is adaptive according to the specifics of the environment in terms of available networked services and their provided conversations. Moreover, our approach enforces a valid consumption of the composed services, ensuring that their conversations are fulfilled.

To evaluate our approach, we have implemented a prototype of COCOA; experimental results allow us to validate the relevance of the employed paradigms in pervasive computing environments.

The remainder of this paper is structured as follows. First, we present related re-

[6] OWL: Web Ontology Language. http://www.w3.org/TR/owl-ref/

[7] Amigo: ambient intelligence for the networked home environment. http://www.extra.research.philips.com/euprojects/amigo/

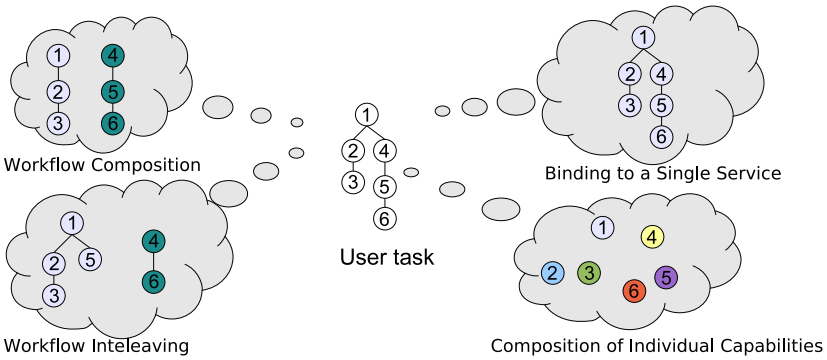

Fig. 2. Flexibility enabled by COCOA

search efforts in the area of dynamic composition of user tasks in pervasive computing environments, as well as conversation-based service composition (Section 2). Then, we introduce COCOA-L, our language for semantic, QoS-aware specification of services and tasks (Section 3). Building on COCOA-L, we present formalisms enabling the realization of COCOA in Section 4, and the mechanisms constituting COCOA in Section 5. More specifically, the latter concerns COCOA-SD our approach to semantic service discovery, and COCOA-CI our approach to conversation integration. In Section 6, we assess our approach based on its performance evaluation. Finally, we conclude with a summary of our contributions and future work in Section 7.

2 Service Composition in Pervasive Computing Environments: State of the Art

As introduced in the previous section, a user task is a software application available on the user's device that is abstractly described in terms of functionalities to be integrated. These functionalities have then to be dynamically provided by the environment. Dynamic realization of user tasks is one of the major challenges in mobile environments, as it allows users to perform potentially complex software applications opportunely according to the surrounding environment. A number of research efforts have been conducted in the area of dynamic realization of user tasks in pervasive computing environments. The Aura project [28] defines an architecture that realizes user tasks in a transparent way. The user tasks defined in Aura are composed of abstract services to be found in the environment. Gaia [26] is a distributed middleware infrastructure that enables the dynamic deployment and execution of software applications. In this middleware, an application is mapped to available resources of a specific *active space*. This mapping can be either assisted by the user or automatic. Gaia supports the dynamic reconfiguration of applications. For instance, it allows changing the composition of an application dynamically upon a user's request (e.g., the user may specify a new device providing a component that should replace a component currently used). Furthermore, Gaia supports the mobility of

applications between active spaces by saving the state of the application. Both of the previous platforms introduce advanced middleware to ease the development of pervasive applications composed out of networked resources. However, they are too restrictive regarding the networked resources that may be integrated since resources have to host the specific middleware to be known by pervasive applications. Furthermore, both approaches assume framework-dependent XML-based descriptions for services and tasks. In other words, both approaches assume that services and tasks of the pervasive computing environment are aware of the semantics underlying the employed XML descriptions. However, in open pervasive environments it is not reasonable to assume that service developers will describe services with identical terms worldwide. This raises the issue of syntactic heterogeneity of service interfaces. Indeed, while building upon service oriented architectures (e.g., Web services) resolves the heterogeneity of services in terms of employed technologies, interaction with services is based on the syntactic conformance of service interfaces, for which common understanding is hardly achievable in open pervasive computing environments. A key requirement for enabling the dynamic realization of user tasks in pervasive environments concerns expressing the semantics of services and tasks.

A promising approach addressing the semantic modeling of information and functionality comes from the Semantic Web paradigm [6]. There, information, originally comprehensible only by humans, is enriched with machine-interpretable semantics, so as to allow its automated manipulation. Such semantics of an entity encapsulate the meaning of this entity by reference to a structured vocabulary of terms *(ontology)* representing a specific area of knowledge. Ontology languages support formal description and machine reasoning on ontologies; the Web Ontology Language (OWL) [8] is a recent recommendation by W3C. These notions come from the knowledge representation field and have been applied and further evolved in the Semantic Web domain. Then, a natural evolution has been the combination of the Semantic Web and Web Services into Semantic Web Services [9]. This effort aims at the semantic specification of Web services towards automating Web services discovery, invocation, composition and execution monitoring. Hence, a number of research efforts have been proposed for the semantic specification of Web services. For instance, the latest WSDL (2.0) standard does not only support the use of XML Schema, but also provides standard extensibility features for using, e.g., classes from OWL ontologies to define Web services input and output data types. A recent proposal for the semantic specification of Web services is the Web Services Modeling Ontology (WSMO), which is specified using the Web Service Modeling Language (WSML). Besides service specification, this ontology provides support for *mediators*, which can resolve mismatches between ontologies or services. METEOR-S [25] is another proposal for enhancing Web service descriptions and enabling Web service composition. METEOR-S uses DAML+OIL [9]

[8] OWL: http://www.w3.org/TR/owl-ref/
[9] DAML+OIL: http://www.w3.org/TR/daml+oil-reference

(the direct precursor to OWL) ontologies to add semantics to WSDL and UDDI. The Web Service Semantics (WSDL-S) proposal, coming from the METEOR-S project, also annotates Web services with semantics, using references to concepts from, e.g., OWL ontologies, by attaching them to WSDL input, output and fault messages, as well as operations. The First-Order Logic Ontology for Web Services (FLOWS) is a recent proposal for the semantic specification of Web services. It has a well defined semantics in first-order logic enriched with support of Web based technologies (e.g., URIs, XML). FLOWS encloses parts of other languages and standards (e.g., WSMO, OWL-S, PSL (ISO 18629)) and supports a direct mapping to ROWS, another language from the same consortium based on logic programming (i.e., rules). OWL-S is a Web service ontology specified in OWL, which is used to describe semantic Web services. A service description in OWL-S is composed of three parts : the *service profile*, the *process model* and the *service grounding* (see Figure 3). The service profile gives a high level description of a service

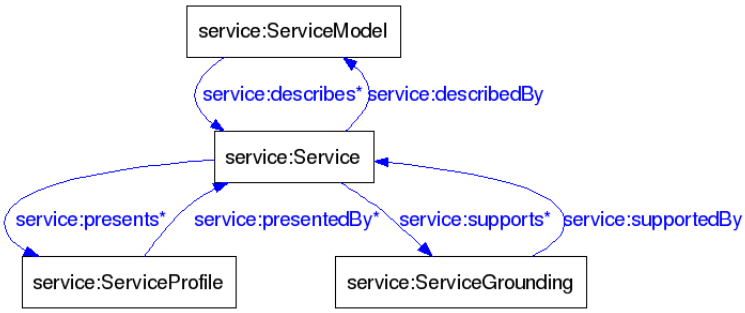

Fig. 3. OWL-S top level ontology

and its provider. It is generally used for service publication and discovery. The process model describes the services behavior as a process. This description contains a specification of a set of sub-processes coordinated by a set of control constructs. These control constructs are: *Sequence*, *Split*, *Split + Join*, *Choice*, *Unordered*, *If-Then-Else*, *Repeat-While*, and *Repeat-Until*. The sub-processes can be either composite or atomic. Composite processes are decomposable into other atomic or composite processes, while atomic ones correspond to WSDL operations. The service grounding specifies the information necessary for service invocation, such as communication protocols, message formats, serialization, transport and addressing information. The service grounding uses WSDL binding information. More precisely, it defines mapping rules to link OWL-S atomic processes to WSDL operations.

In the area of ontology-based dynamic service composition in pervasive environments, an effort based on OWL-S has been proposed in [18]. In this approach called Task Computing, services of the pervasive computing environment are described as semantic Web services using OWL-S. Each user of the pervasive computing environment carries a composition tool that discovers on the fly available services in the user's vicinity and suggests to the user all the possible compositions of these services based on their semantic inputs/outputs. While this approach validates the relevance of semantic Web technologies in pervasive computing environments, it

presents some drawbacks. For instance, suggesting to the user all the possible compositions of networked services requires that the user selects the right composition among the suggested ones, which can be inconvenient for mobile users of the pervasive computing environment. Indeed, the pervasive computing environment should minimize the users' distractions by enabling the automatic and transparent deployment and execution of user tasks. Furthermore, the services to be composed are considered as providing a single functionality, while more complex services (e.g., composite services specified with their corresponding conversation) are not considered for the composition. Such composition, involving services or realizing tasks described with their conversations, identified as conversation-based service composition, allows the realization of more complex user tasks.

In the last few years a number of research efforts have been conducted in the area of conversation-based service composition [1,2,17,4,7]. For instance, Klein *et al.* in [1] propose to describe services as processes, and define a request language named PQL (Process Query Language). This language allows finding in a process database those processes that contain a fragment that corresponds to the request. While this approach proposes a process query language to search for a process, it does not handle process integration. Thus, the authors implicitly assume that the user's request is quite simple and can be performed by a single process. On the contrary, in our approach a composition effort is made to reconstruct a task conversation by integrating services conversations.

Aggarwal *et al.* propose to describe a task conversation as a BPEL4WS [10] workflow [2]. This description may contain both references to known services (static links) and abstract descriptions of services to be integrated (service templates). At execution time, services that match the service templates are discovered, and the task's workflow is carried out by invoking the selected services. This approach proposes a composition scheme, in which a set of services are integrated to reconstruct a task's conversation. However, the services being integrated are rather simple. Indeed, each service is described at the interface level without describing the service conversation. On the contrary, we consider services as entities that can behave in a complex manner, and we try to compose these services to realize the user task's conversation.

Another conversation-based matching algorithm is proposed by Majithia *et al.* in [17]. In this approach, the user's request is specified at the interface level and is mapped to a workflow. Then, service instances that match the ones described in the workflow, in terms of inputs, outputs, pre-conditions and effects, are discovered in the network, and a concrete workflow description is constituted. As for the previous approaches, the service composition scheme that is proposed does not involve any conversation integration, as the Web services are only described at the interface-level.

[10] BPEL4WS: http://www-128.ibm.com/developerworks/library/ws-bpel/

The work proposed by Bensal and Vidal in [4] uses the OWL-S process model to match services. In their approach, the authors consider a user request in the form of required inputs/outputs, and assume a repository of OWL-S Web services. Then, they propose a matching algorithm that checks whether there is a process model in the repository that meets the desired inputs/outputs. Brogi *et al.* in [7] have proposed an enhancement of this last algorithm by performing a composition of services' process models to respond to inputs/outputs of the user's request. This last effort is close to our work, as an effort of integrating conversations is investigated. However, some differences remain. The main difference is that the authors consider that the user request can be expressed in the form of a list of inputs/outputs. While this is an interesting assumption, this implicitly prevents the user from performing complex conversations. Indeed, the algorithm composes in a pipe and filter like-way, atomic processes that are compatible in terms of provided outputs and requested inputs (signatures). While this strongly guarantees that the composed services will be able to exchange information, it weakly guarantees that the resulting composition will provide the user with the expected semantics. On the contrary, we consider that the user's request is expressed as a conversation, which guarantees that the resulting composition will indeed meet the user task's expected behavior.

The QoS-aware dynamic realization of tasks in pervasive computing environments through the integration of service conversations calls for a language that allows the semantic-aware description of services and of tasks' functional and non-functional capabilities, as well as of services' and tasks' conversations. For this purpose, we present in the following section COCOA-L, a language for specification of services and tasks of the pervasive environment.

3 COCOA-L: an OWL-S Based Service and Task Description Language

We describe herein COCOA-L, an OWL-S based language for the specification of networked services and user tasks in pervasive environments. This language extends OWL-S in order to fit the requirements of service composition in pervasive computing environments. Specifically, COCOA-L allows the specification of:

(1) Services' and tasks' advertised and requested functional capabilities;
(2) Services' and tasks' conversations for modeling their behavior; and
(3) Services' and tasks' QoS properties.

The UML diagram depicted in Figure 4 represents the main conceptual elements of COCOA-L with respect to (1), (2) and (3). These elements are further detailed in the following three sections. Note that in this diagram, colored boxes are those corresponding to reused OWL-S elements.

3.1 Requested and Advertised Capabilities

At the heart of COCOA-L, we distinguish the notion of *capability*. A capability characterizes a functionality that might be requested or advertised by a service/task. A capability is realized by the invocation of a set of *operation*s, i.e., a sequence of messages exchanged between a client and a service provider (e.g., WSDL operations). A *requested capability* has a set of provided *inputs*, a required *category*, and a set of required *outputs* and *QoS properties*; while an *advertised capability* has a set of required inputs, a provided category, and a set of provided outputs and QoS properties.

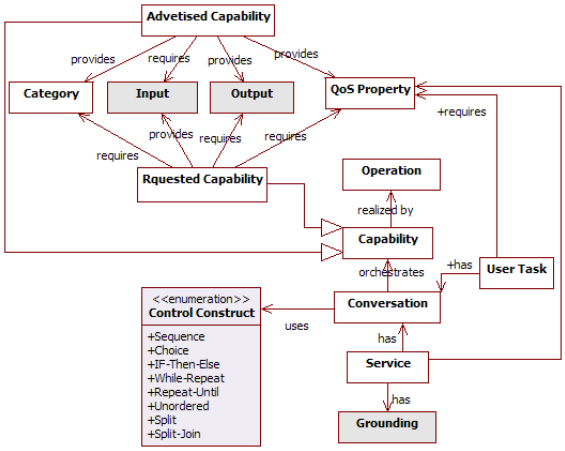

Fig. 4. COCOA-L

In COCOA-L, both *user tasks* and *services* are specified with a *conversation*, which comprises respectively requested and advertised capabilities. When a user task is being performed, its requested capabilities have to be bound to advertised capabilities of networked services.

3.2 Service Conversation Specification

A *conversation* represents the coordination of a set of capabilities by *control constructs* (e.g., Sequence, Parallel, Choice constructs). In COCOA-L, we use the OWL-S control constructs for coordinating capabilities of services and tasks. Due to their involvement in a conversation, capabilities have data and control dependencies between each other. Control dependencies are those due to the structure of the conversation. Specifically, two capabilities C_1 and C_2 are said to have a control dependency if C_1 is prior to C_2 in the conversation, and in order to enforce a valid service/task consumption C_1 must be performed before C_2 when the service/task is being performed. Nevertheless, when realizing user tasks, the interleaving of

multiple service conversations is supported as long as the services control dependencies are fulfilled. For instance, if the user task conversation is a sequence of the four capabilities $Sequence(C_1, C_2, C_3, C_4)$ and the two services to be composed S_1 and S_2 have respectively the two following conversations : $Sequence(C_1, C_3)$ and $Sequence(C_2, C_4)$, a composition that interleaves S_1 and S_2 conversations, while meeting S_1 and S_2 control dependencies, is given as follows:
$Sequence(S_1.C_1, S_2.C_2, S_1.C_3, S_2.C_4)$.

A data dependency between two capabilities C_1 and C_2 is specified when data produced by C_1 must be consumed by C_2 and only by C_2. When a data dependency is specified in a user task conversation, this means that the corresponding two capabilities must be provided by the same service. For instance, when realizing a user task comprising a booking and payment capabilities for a hotel room, one can imagine that it is not possible to use the booking capability of a hotel reservation service and the payment capability of another hotel reservation service. In the example of Figure 5, a data dependency is specified between the capabilities Browse and Get Stream of the e-movie application, which means that these two capabilities must be provided by the same networked service. If a data dependency is specified in a service conversation, this means that the corresponding capabilities must be performed in sequence without interleaving with other capabilities outside the service conversation.

COCOA-L further supports the specification of data flow between capabilities. Specifically, data flow specifies which output data produced by a capability may be consumed by another capability. Data flow specification is nevertheless different from data dependencies in the fact that it does not drive the service selection process. For instance, if the user task contains the specification of a data dependency between two capabilities, it will drive the selection of services that provide both capabilities in the same conversation, while specifying a data flow relation does not lead to any constraint in service selection. Indeed, the selection of two capabilities that belong to two different services may be performed, as long as the selected capabilities are compatible in terms of inputs/outputs to be exchanged with respect to the data flow specification.

Our objective is to realize user tasks based on their conversation specification through the integration of services also specified with their conversation. This integration has to fulfill both data and control dependencies of user tasks and services.

3.3 Service QoS Specification and Measurement

3.3.1 QoS Specification

QoS specification associated with the dynamic composition of user tasks is concerned with capturing the user tasks QoS requirements as well as services QoS

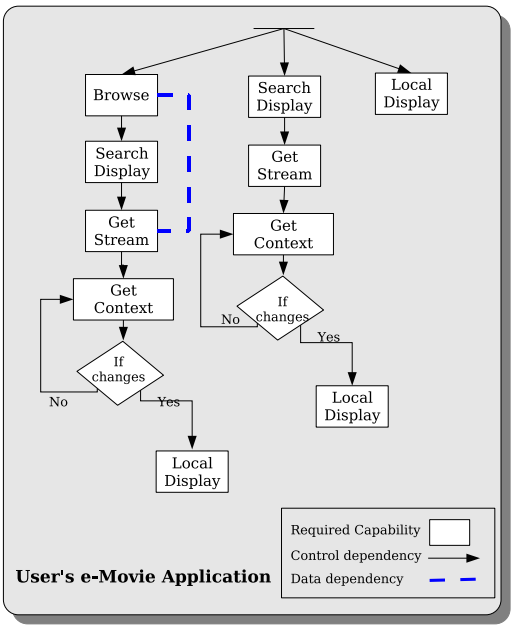

Fig. 5. The e-movie application

properties. QoS specification should: (1) allow the description of both quantitative (e.g., service latency) and qualitative (e.g., CPU scheduling mechanism) QoS attributes; and (2) be declarative in nature, that is, specify only what is requested, but not how the requirements should be implemented by services [3].

In the following, QoS *category* refers to a specific non functional property of a service that we are interested in (e.g., *performance*). Every category consists of one or more *dimensions*, each representing one attribute of the category. For instance, *latency* defines a dimension of the performance category. Quantitative dimensions in QoS specification, also referred to as *metrics*, measure specific quantifiable attributes of the service. Qualitative dimensions, referred to as *policies*, dictate the behavior of the services. These dimensions are described in COCOA-L with references to ontology concepts. Sabata *et al.* further classify the metrics into categories of *performance*, *security levels*, and *relative importance*. Policies are divided into categories of *management* and *level of service* [27].

Based on the aforementioned work, and the work introduced in [23], we introduce a base QoS specification of services depicted in the UML diagram of Figure 6, which is adapted to pervasive environments. Specifically, we notice that although more QoS parameters yield more detailed description, the gain has to be put up against the increased overhead. Usually, a small number of parameters (i.e., ≤ 5) is sufficient to capture the dominant QoS properties of a system [10]. Along with the factor of limited resources on mobile devices, we only take into account the most dominant and descriptive dimensions in our base QoS specification, instead of trying to incorporate every possible applicable dimension. However, it can be easily extended with more dimensions, if requested by specific services or tasks, by supporting the new dimensions in a way similar to the ones discussed in this

section.

In the latter diagram, dark colored boxes represent qualitative dimensions, whereas light colored boxes represent quantitative ones. A *QoS Property*, is described based on QoS dimensions and expressed as a boolean expression using the following operators: *and, or, not, equal, not-equal, is-a, is-exactly-a, is-not-a, more-than, less-than, max-value-of, min-value-of*. The operators *is-a, is-exactly-a* and *is-not-a* are used to compare qualitative properties, while *equal, not-equal, more-than, less-than, max-value-of, min-value-of* operators are used to compare quantitative properties. Finally, the *and, or* and *not* operators are used to define composite properties.

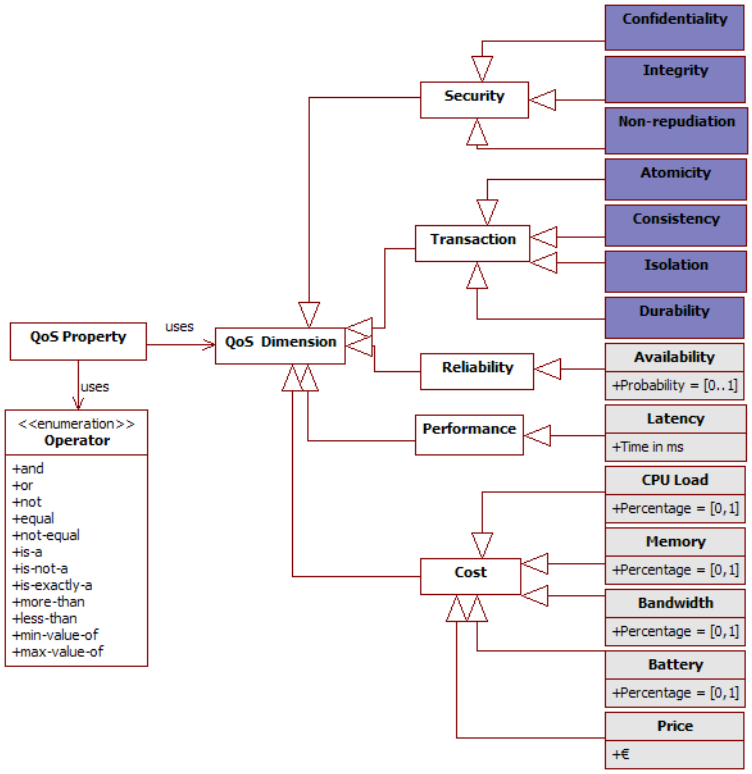

Fig. 6. QoS Specification

According to our service model depicted in Figure 4, a user task has two kinds of required QoS properties: QoS properties specified at the level of capabilities expressing *local QoS requirements*, and QoS properties specified at the level of the whole task expressing *global QoS requirements*. Local QoS requirements have to be satisfied by individual advertised capabilities of services, whereas global QoS requirements has to be satisfied by the resulting service composition. The mechanisms used to check the fulfillment of local and global QoS properties of user tasks are further detailed in Sections 5.1 and 5.2, respectively.

3.3.2 Measurement of Quantitative QoS Dimensions

The specification of quantitative QoS dimensions in service requests and advertisements requires providing dimension measuring as accurate as possible. Service-level dimensions can be measured easily (e.g., off-line measurements using available quality analysis tools). Resource-related measures for the services are also easy to obtain *after* service execution, using available utilities (e.g., pathchar [11] for bandwidth measurement). However, providing accurate metrics measures for the selection of services *prior* to their execution requires special care, since this relates to predicting the service's resource consumption. The prediction of service metrics can be carried out based on histories [11,24,13], which has been proved to be accurate and efficient [13].

In our case, while evaluating the QoS of a service composition, we provide two 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. Actually, we consider both the previous estimations, which depend on the user's task requirement (e.g., deterministic or probabilistic) in the user's request. For example, 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. Further details about how we perform these estimations are given in section 4.2. Moreover, we use *relative importance* to characterize both the users' preferences among the various QoS dimensions and the criticality of the hosts' resources. Further details about the use of *relative importance* among QoS dimensions are given in Section 5.2.

4 Formalisms for QoS-aware Dynamic Service Composition

In this section we introduce two formalisms enabling the integration of services' conversations for realizing user tasks with support of QoS.

4.1 Modeling Service Conversations as FSA

In order to ease service composition by enforcing control and data dependencies of services/tasks, we propose to model services and tasks conversations using finite state automata. Other approaches to formalizing Web services conversations and composition have been proposed in the literature based on Petri nets [29], process algebras [16] or finite state machines [12]. Figure 7 describes the mapping

[11] http://www.caida.org/tools/utilities/others/pathchar

rules that we have defined for translating an OWL-S process model to a finite state automaton. In this model, automata symbols correspond to capabilities described using COCOA-L. The initial state corresponds to the beginning of the conversation, and final states correspond to the end of a client/service interaction. Each control

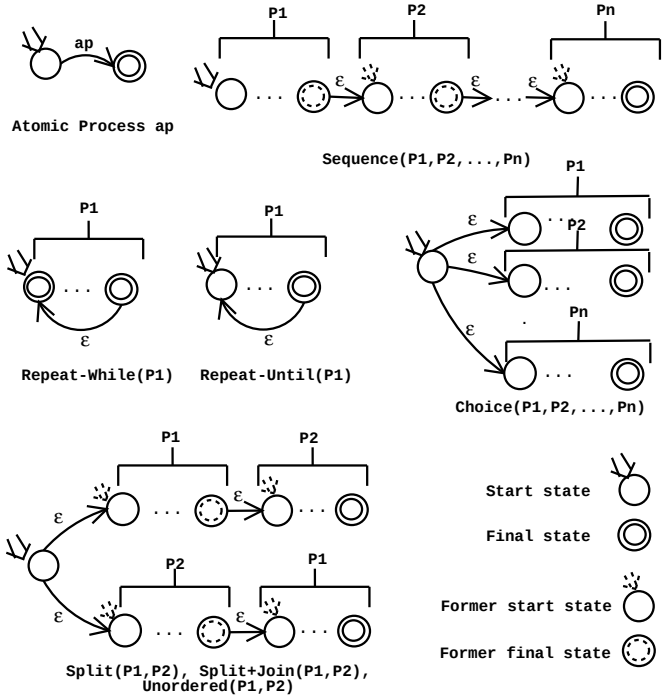

Fig. 7. Modeling OWL-S processes as finite state automata

construct involved in a conversation is mapped to an automaton using the rules depicted in Figure 7. Then, these automata are linked together in order to build a global automaton. Further details about modeling OWL-S processes as automata can be found in [5]. Figure 8 shows the automaton representing the e-movie application. Both user tasks and networked services are modeled as finite state au-

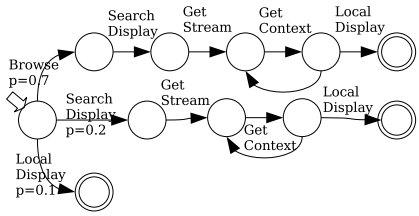

Fig. 8. Automaton of the e-movie application

tomata. However, the user task's automaton is enriched with additional information in some of its transitions, i.e., the probability for this transition to be selected. More precisely, a probability value is introduced in the case of a Repeat-While, Repeat-Until and Choice constructs. For the first two constructs (loops), the information added is the probability for the corresponding process to be executed once again. In the case of the Choice control construct, a probability is attached to each possible choice of this construct. This information is necessary to calculate a probabilistic

QoS estimation of a composition, as further discussed in the following section. For instance, if a composition involves a loop, the QoS of this composition depends on how many times the user will execute this loop. All these probabilities are evaluated based on histories and are updated each time the user task is executed. In addition to these probabilities, some other information is needed to estimate the worst case value of QoS parameters. This information is attached to each loop construct in the task's process, and gives the maximum number of times the loop can be carried out during the execution of a user task.

4.2 Evaluating the QoS of composed User Tasks

In our approach, as the task is abstract, i.e., do not refer to specific services, we need to extract the QoS formulae corresponding to each QoS metric. These formulae are extracted in advance and stored with the task's description. Then, during the composition, each time an element is being composed, these formulae are used to check the fulfillment of the task's QoS requirements. A number of research efforts propose reduction rules to compute the QoS of a workflow [8,19]. We use the model proposed by J.Cardoso *et al.* in [8] to extract the formula of each QoS dimension, corresponding to the task's automaton structure. In this approach, a mathematical model is used to compute QoS for a given workflow process. More precisely, an algorithm repeatedly applies a set of reduction rules to a workflow until only one atomic node remains. This remaining node contains the QoS formula for each considered metric, corresponding to the workflow under analysis. The algorithm uses a set of six reduction rules: (1) sequential, (2) parallel, (3) conditional, (4) fault-tolerant, (5) loop and (6) network. However, as our automata model is an abstraction of the OWL-S workflow constructs, we only need to keep the reduction rules for sequential, conditional, and loops systems.

As introduced earlier, we provide two estimations for each QoS dimension: (1) a history-based probabilistic estimation and (2) a pessimistic estimation. Figure 9 and Tables 1 and 2 show how we perform these estimations. Figure 9 describes the reduction rules to be applied for sequence, choice and both simple and dual loop constructs. In this figure capabilities represented on each transition (named o_i) provides some QoS attributes (e.g., Availability (noted a_i), Latency (noted l_i), Cost [12] (noted c_i)). Besides these attributes, some capabilities, i.e. those involved in the choice and loops constructs, have additional information, i.e., the probability to be selected (p_i). These probabilities are only used in the case of a probabilistic average estimation of QoS. The formulae to be applied in this case are described in Table 1. Note that in this Table, for each loop case, the probabilities p_i described in Figure 9, are changing to p'_i after reduction, where: $p'_i = \frac{p_i}{1-p}$. On the other hand, evaluating a worst case estimation of QoS requires the use of the above reduction rules,

[12] In the following we refer cost to any cost-related dimension, e.g., CPU load, memory.

by applying the formulae described in Table 2. In this case another information is required for both loop cases, which is the maximum number of times a loop can be executed, as described earlier. This information is represented by N in Table 2.

We focus on the QoS of a service composition with respect to the three dimensions: availability, latency and cost, because they are considered as important QoS dimensions of user tasks (e.g., [8]) and other quantitative dimensions can be calculated in a similar way. For qualitative dimensions, their evaluation is trivial since it only needs to ensure that the policy of each composed service is no weaker than the user's request. This is done by reasoning on the semantic concepts describing the required policies and the provided ones.

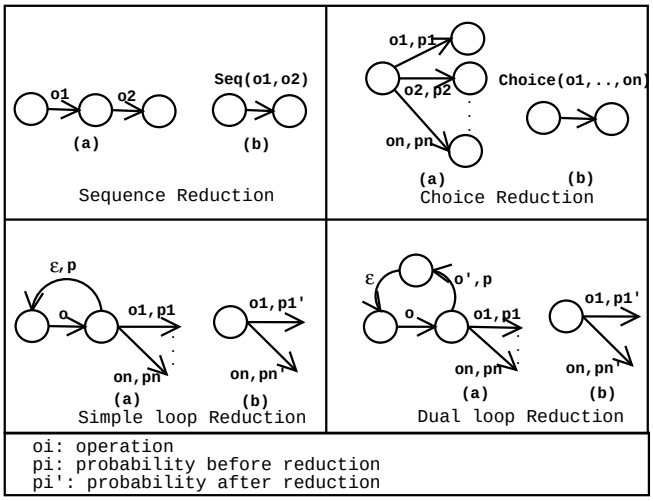

Fig. 9. Workflow Constructs

	Seq	**Choice**	**Simple Loop**	**Dual Loop**
Availability	$a_1 * a_2$	$\sum a_i p_i$	$\frac{(1-p)*a_o}{1-pa_o}$	$\frac{(1-p)*a_o}{1-a_o a_{o'}}$
Latency	$l_1 + l_2$	$\sum l_i p_i$	$\frac{l_o}{1-p}$	$\frac{l_o+l_{o'}-(1-pl_{o'})}{1-p}$
Cost	$c_1 + c_2$	$\sum c_i p_i$	$\frac{c_o}{1-p}$	$\frac{c_o+c_{o'}-(1-p)c_{o'}}{1-p}$

Table 1
History-based probabilistic average QoS evaluation

	Seq	**Choice**	**Simple Loop**	**Dual Loop**
Availability	$a_1 * a_2$	$Min(a_i)$	$N * a_o * Min(a_i)$	$N * a_o * a_{o'} * Min(a_i)$
Latency	$l_1 + l_2$	$Max(l_i)$	$N * l_o + Max(l_i)$	$N * (l_o + l_{o'}) + Max(l_i)$
Cost	$c_1 + c_2$	$Max(c_i)$	$N * c_o + Max(c_i)$	$N * (c_o + c_{o'}) + Max(c_i)$

Table 2
Pessimistic QoS evaluation

Having these two formalisms introduced, we present in the following two sections the two mechanisms constituting COCOA, i.e., QoS-aware service discovery

(COCOA-SD) in Section 5.1, and QoS-aware conversation integration (COCOA-CI) in Section 5.2.

5 Mechanisms for QoS-aware Dynamic Service Composition

5.1 QoS-aware Service Discovery: COCOA-SD

Service discovery allows finding in the pervasive environment, at the specific time and place, service advertised capabilities that match service requested capabilities towards the realization of user tasks. Service discovery decomposes into service matching and service selection as described below.

5.1.1 Service Matching

Service matching allows identifying services that provide semantically equivalent capabilities with those of the user task's conversation. Furthermore, these capabilities should fulfill the QoS properties required in the task's requested capabilities. We use the matching relation $Match(Adv, Req)$ to match an advertised capability Adv against a requested capability Req. This relation extends the relation defined in [21] with the matching of QoS properties. Specifically, the $Match$ relation is defined using the function $distance(concept_1, concept_2)$, hereafter denoted by $d(concept_1, concept_2)$, which gives the semantic distance between two concepts, $concept_1$ and $concept_2$, as given in the classified ontology to which the concepts belong. Precisely, if $concept_1$ does not subsume [13] $concept_2$ in the ontology to which they belong to, the distance between the two concepts does not have a numeric value, i.e., $d(concept_1, concept_2) = NULL$. Otherwise, i.e., if $concept_1$ subsumes $concept_2$, the distance takes as value the number of levels that separate $concept_1$ from $concept_2$ in the ontology hierarchy obtained after ontology classification. In this relation, we consider the case where $concept_1$ is subsumed by $concept_2$ as a mismatch and we assign the value $NULL$ to the relation d because such matching implies that a client may be provided with an advertised capability that is more specific than the requested capability, which may lead to a malfunction of the advertised capability. For instance, if the advertised capability translates only Latin languages into other Latin languages, and the client provides in its requested capability the concept Language as input, which subsumes both Greek and Latin languages, the advertised capability will not work if the client invokes the corresponding service with a text in Greek as input. Moreover, as we aim at the automatic realization of user tasks we opt for the selection of only capabilities that are

[13] Subsumption means the fact to incorporate something under a more general category.

equivalent or more generic than the requested capabilities, thus avoiding the risk of malfunctioning capabilities.

Formally, let the advertised capability Adv be defined by the set of required inputs $Adv.In$, a set of provided outputs $Adv.Out$, a provided category $Adv.Cat$, and a set of provided QoS properties $Adv.P$. On the other hand, let the requested capability Req be defined by a set of provided inputs $Req.In$, a set of required outputs $Req.Out$, a required category $Req.Cat$ and a set of required QoS properties $Req.P$.

The relation $Match$ is then defined as:

$$\begin{aligned} Match(Adv, Req) = & \forall in' \in Adv.In, \exists in \in Req.In : d(in', in) \geq 0 \text{ and} \\ & \forall out' \in Req.Out, \exists out \in Adv.Out : d(out, out') \geq 0 \text{ and} \\ & d(Adv.Cat, Req.Cat) \geq 0 \\ & \forall p' \in Req.P, \exists p \in Adv.P : (p \Rightarrow p') \end{aligned}$$

From the above, the relation $Match(Adv, Req)$ holds if and only if all the required inputs of Adv are matched with inputs provided by Req; all the required outputs of Req are matched with outputs provided by Adv; the category required by Req is matched with the category provided by Adv and all the required properties of Req are matched with properties provided by Adv.

5.1.2 Service Selection

Service selection allows identifying which services from those that offer semantically equivalent capabilities to the capabilities of the user task are potentially useful for the composition. The selection of services is based on the control dependencies that are inherent to their conversation specification. For instance, a service that provides a semantically equivalent capability to one of the requested capabilities of the user task, could not be useful for the composition if the latter capability has data or control dependencies with capabilities that are not requested at all in the user task. To perform this selection we use regular expressions. Specifically, we extract from the task automaton the regular expression that represents the language generated by this automaton. For each term of this regular expression, which corresponds to a capability from the task description, we introduce the quantifier $?$ that indicates that there is **0** or **1** occurrence of this term. For example, the regular expression extracted for the automaton of the e-movie application presented in Figure 8 is given by :

$(Browse)?\ (SearchDisplay)?\ (GetStream)?\ (GetContext)^*\ (LocalDisplay)?$
$|$
$(SearchDisplay)?\ (GetStream)?\ (GetContext)^*\ (LocalDisplay)?\ |$
$(LocalDisplay)?$

Let's note by L the language generated by the extracted regular expression and by $L_1, L_2, ..., L_n$ the languages generated by the automata of the pre-selected services $S_1, S_2, ..., S_n$ respectively. COCOA-SD selects all the services S_i such that $L \cap L_i \neq \emptyset$. For example, a service that provides a sequence of capabilities that match semantically the capabilities Browse and GetStream of the user task, is selected.

This allows the selection of services that meet the control dependencies of the user task by enabling the potential interleaving of their conversations. Furthermore, if a data 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 selection is also based on QoS specifications. Particularly, if local QoS requirements are specified in some capabilities of the user task, service capabilities that do not fulfill the latter requirements are not selected for the composition.

5.2 QoS-aware Conversation Integration: COCOA-CI

Once semantic-aware service discovery is achieved, the next step towards dynamic composition of user tasks, is the integration of the conversations of the selected services. COCOA-CI integrates the conversations of services selected using COCOA-SD, to realize the conversation of the target user task. Moreover, COCOA-CI supports interleaving of these conversations. COCOA-CI integrates the conversations of discovered services to realize the user task, based on associated state automata.

COCOA-CI first integrates all the automata of 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. Consider the automaton representing the conversation of the target user task depicted in Figure 10, left higher corner, and the automata representing the conversations of the selected services, Figure 10, right lower corner. In this figure, all the automata of the selected services are connected in a global automaton, in which all the added transitions are represented with dashed lines.

The next step of COCOA-CI is to parse each state of the task's 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's automaton a state of the global automaton that can *simulate* it, i.e., a task's automaton state is simulated by a global automaton state when for each incoming symbol [14] of the

[14] Incoming symbols of a state correspond to the labels of the next transitions of this state.

former there is at least one semantically equivalent [15] incoming symbol of the latter. For example, in Figure 10, the state t_1 of the task's automaton can be simulated by the initial state of the global automaton because the set of incoming symbols of t_1, is a subset of the set of incoming symbols of the global automaton initial state.

COCOA-CI allows finding service compositions with possible interleaving of conversations of the involved services. Indeed, this is done by managing service sessions. A service session characterizes the execution state of a service conversation. A session is opened when a service conversation starts and ends when this conversation finishes. Several sessions with several networked services can be opened at the same time. This allows interleaving the interactions with distinct networked services. Indeed, a session opened with a service A can remain opened (temporary inactive) during the interaction of the client with another service B. An example of managing sessions is given in Step (1) of the composition. In this step, the capability Browse of the task's automaton has been matched against the capability Browse of the global automaton. The next step is to find the capability Search Display of the task's automaton (Step (2)). However, this capability is not available in the Video Streaming Service. This leads to open another session with the Display Service as this service provides the sought capability. In Step (3) after matching the capability Search Display, the capability Get Stream is sought. A semantically equivalent capability, i.e., the Send Stream capability, is accessible in the Video Streaming Service from the previously opened session.

An important condition that has to be observed when managing sessions is that each opened session must be 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 others 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).

In addition to checking for each state the equivalence between incoming capabilities, a verification of the conformance to the QoS constraints of the user task is performed. This is done by using the QoS formulae that have been extracted from the task's automaton structure as described in Section 4.2. Thus, we start with the QoS formula for each QoS dimension, in which we initially assume that all capabilities will provide the best value of the considered QoS dimension (for example, latency = 0, availability = 1). Then, each time we examine a service capability, we replace the corresponding best value in the formula of the considered dimension, with the real QoS value of the capability. This allows evaluating at each step of the integration the values of all QoS dimensions in the case that the current capability is selected. These values are then compared to the corresponding values required

[15] We recall that equivalence relationship between capabilities is a semantic equivalence that have already been checked by COCOA-SD.

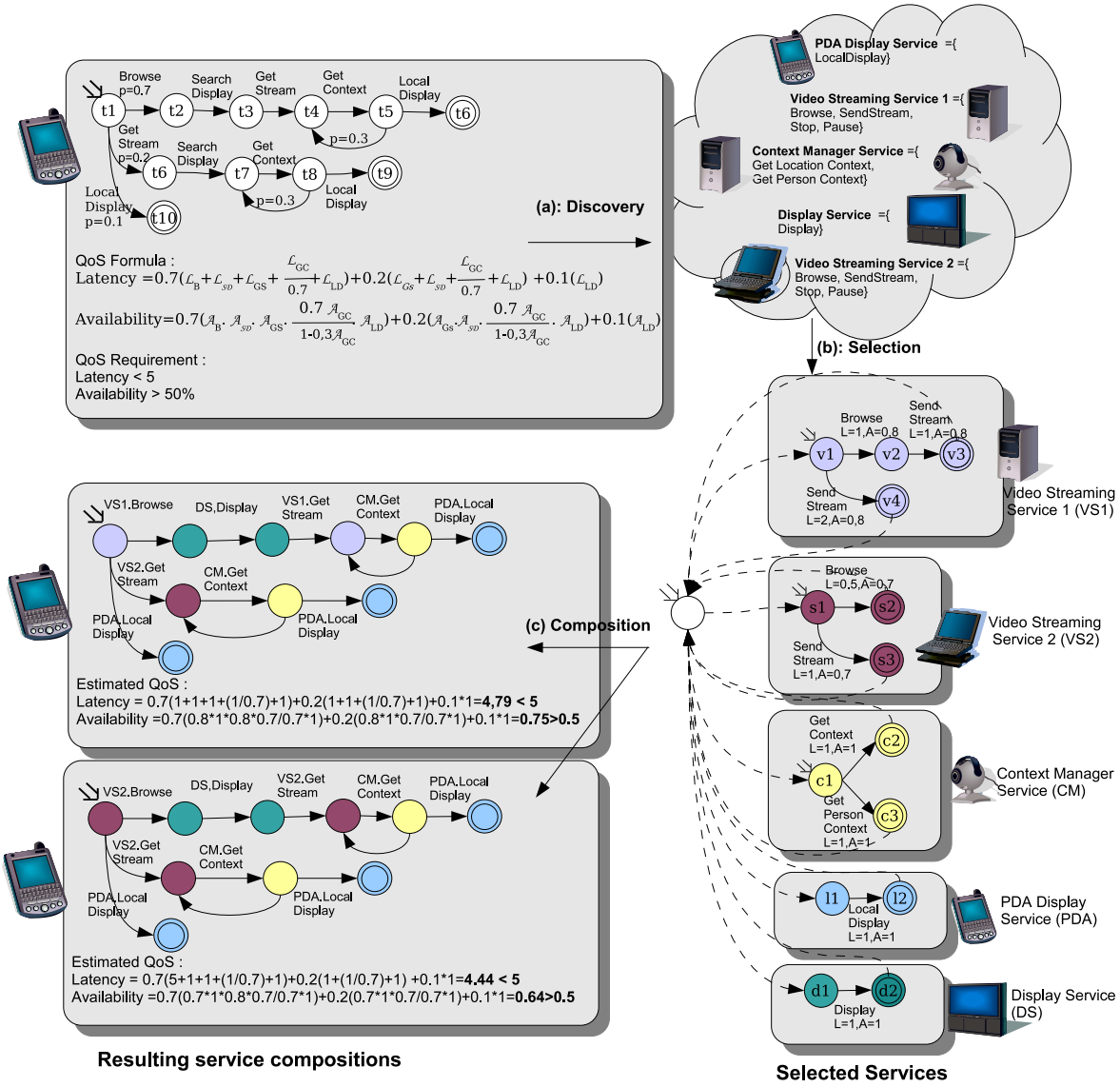

Fig. 10. Conversation Integration

by the user task, and if the constraints are not met, the path in the global automaton that includes this capability is rejected.

COCOA-CI gives a set of sub-automata from the global automaton that conforms to the task's automaton structure (two sub-automata are depicted in the left lower corner of Figure 10). Each of these automata is a composition of networked services that conforms to the conversation of the target user task, further enforcing valid service consumption.

Once the set of possible compositions is given (See Figure 10 where two compositions are given by COCOA-CI), a last stage is to choose the best among resulting compositions, on the basis of provided QoS. However, since different dimensions are in different units, data normalization is needed. In our case, we apply standard

deviation normalization on the various dimensions as in [15]:

$$d'(c_i) = \begin{cases} 2 & if(d(c_i) - m(d) > 2 * \delta(d)) \\ 0 & if(d(c_i) - m(d) < -2 * \delta(d)) \\ \frac{d(c_i) - m(d)}{2 * \delta(d)} + 1 & otherwise \end{cases} \quad (1)$$

where $d(c_i)$ is the value of dimension d for the service composition c_i, and $m(d)$ and $\delta(d)$ are the mean value and standard deviation for dimension d, respectively. Note that for QoS parameters that are stronger with smaller values (e.g., latency), $d'(c_i)$ is further transformed by $d''(c_i) = 2 - d'(c_i)$, so that stronger values are normalized to greater values.

With every dimension normalized, every service composition is evaluated based on a benefit function like in [15]:

$$\text{Overall Benefit} = \sum_{i=1}^{n} (d(c_i) * w_i) / \text{Service Composition Cost} \quad (2)$$

where w_i is the relative importance of the considered dimension.

Using the service composition that has been selected, the conversation description of the user task is complemented with information coming from the composed services. Specifically, each capability of the user task is replaced with the corresponding capability of the networked services. This capability may correspond to either one single or a sequence of client/service interactions. Furthermore, a grounding description for the user task, which contains the binding information of the composed services is generated.

The complemented task's description and the generated grounding are sent to an execution engine that performs the user task by invoking the appropriate networked services.

6 Prototype Implementation and Performance Evaluation

COCOA decomposes into two main mechanisms, COCOA-SD for discovering component services and COCOA-CI for integrating the conversations of selected services. COCOA-SD relies on semantic reasoning on ontologies used to infer relations between semantic descriptions, which we have identified as a costly mechanism [22]. Nevertheless, semantic discovery of service capabilities can be performed efficiently in pervasive computing environments upon the deployment of

appropriate solutions. Indeed, in [21] we present an efficient semantic service discovery protocol for pervasive computing environments. Results show that rich, semantic service discovery can be performed with response times comparable to the syntactic WSDL-based service discovery. Furthermore, we define mechanisms for structuring service repositories based on the semantic specification of services, which increases the scalability of our protocol. Further details about efficient semantic service discovery in pervasive computing environments can be found in [21].

In this article we are primarily interested in evaluating the performance of COCOA-CI, which is at the heart of the composition process, as well as the impact of supporting QoS awareness.

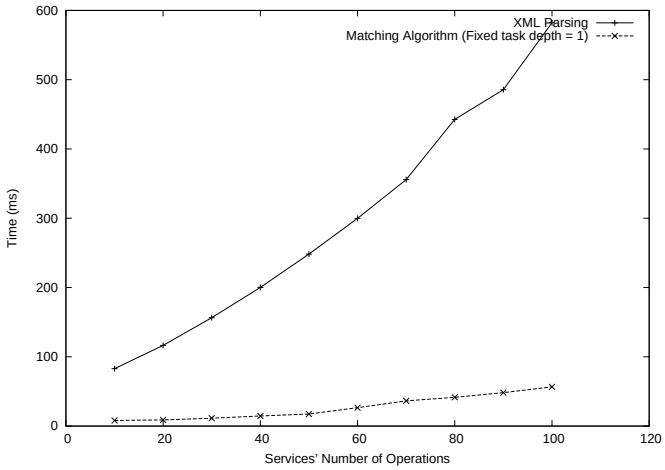

Fig. 11. Performance of COCOA-CI (increasing the number of semantically equivalent capabilities provided by services)

We have implemented COCOA-CI in Java, on a Linux platform running on a laptop with an Intel Pentium 4, 2.80 GHz CPU and 512 MB of memory. The performance of COCOA-CI is proportional to the complexity of the task 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 semantically equivalent capabilities provided by networked services; (2) the number of capabilities requested in the task's conversation. We have carried out two experiments, each evaluating the impact of each factor on the performance of COCOA-CI. In both experiments, each value is calculated from an average of 10 runs.

Figure 11 considers the first factor. In this figure, the number of capabilities provided by networked services is increasing from 10 to 100 capabilities that are semantically equivalent. We compare the performance of COCOA-CI with the XML parsing of the services and task descriptions, which is inherent to the use of Web

services and semantic Web technologies. The resulting curves show that the cost of our algorithm is negligible compared to the XML parsing time. Figure 12 considers the second factor. In this figure, the number of capabilities provided by the networked services is fixed to the worst case coming from the previous experiment, i.e., 100 semantically equivalent capabilities, while the number of capabilities requested in the task's conversation is increasing from 1 to 20. The experiment that is depicted in this figure corresponds to the comparison of the performance of COCOA-CI with the XML parsing of the services and the task conversation descriptions. The figure shows 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 equal to: 100^{nb} in each case, where nb is the number of capabilities requested in the task's conversation. We can see that for a number of possible compositions less than 100^{10}, our algorithm takes less time than the XML parsing time. In realistic cases, both the user task and networked 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 reasonable for the pervasive computing environment. Indeed, we have applied our algorithm in a real case example in which the task's conversation contains twenty requested capabilities and the selected services provide thirty 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 milliseconds to find the two resulting compositions among 36 intermediate compositions, against 152 milliseconds for the XML parsing time.

Figure 12 shows also another important result, which is the impact of introducing QoS in our integration algorithm. This impact is amounts to a small increase in the XML parsing time, which is due to the addition of XML tags for describing QoS, while at the same time to a considerable decrease of the execution time of our algorithm. 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.

7 Conclusion

The pervasive computing vision is increasingly enabled by the large success of wireless networks and devices. In pervasive environments, heterogeneous software and hardware resources may be discovered and integrated transparently towards assisting the performance of users' daily tasks. Building upon the service oriented architecture paradigm and particularly Web services allows having a homogeneous view of the heterogeneous services populating pervasive environments, as services have standard descriptions and communicate using standard protocols. However, realizing such a vision still requires dealing with the syntactic heterogeneity of service descriptions. Most existing solutions to dynamic composition of networked services in pervasive environments poorly deal with such heterogeneity, since they

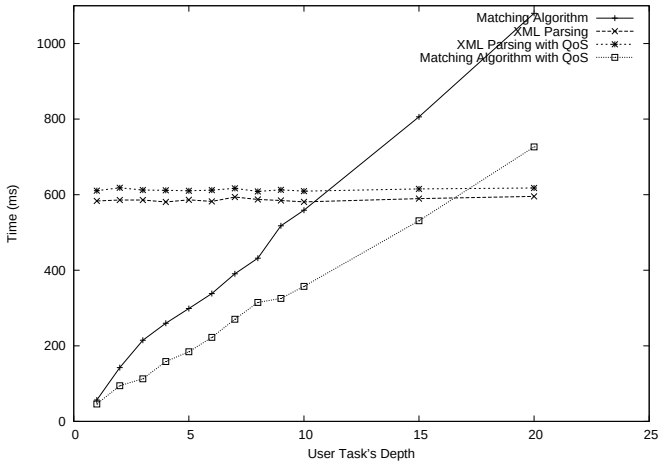

Fig. 12. Performance of COCOA-CI with and without QoS (fixed number of service capabilities, increasing the task's number of capabilities)

assume that components being integrated have been developed to conform syntactically in terms of interfaces.

Building upon semantic Web services, we presented in this article COCOA, our solution to dynamic service composition in pervasive computing environments. COCOA presents a number of attractive features. Indeed, COCOA enables the integration of services having a complex behavior for the realization of user tasks that also have complex behaviors. Specifically, the realization of the user task varies each time a user task is performed according to the specifics of services available in the current pervasive environment. This realization may vary from the integration of individual service capabilities, to the interleaving of potentially complex service conversations. Furthermore, COCOA allows meeting QoS requirements of user tasks.

For the QoS-aware dynamic realization of tasks, we first presented COCOA-L, an OWL-S based language enabling the specification of service advertised and requested capabilities, service conversations, as well as the specification of QoS properties. Then, we presented COCOA-SD, which enables QoS-aware semantic service discovery and COCOA-CI, for the QoS-aware dynamic integration of the selected service conversations.

To perform such a composition, COCOA introduces an abstraction of OWL-S based conversations as finite state automata. This translates the difficult issue of conversation integration to an automata analysis problem by further enabling the assessment of services and tasks data and control dependencies. Furthermore, for enabling QoS-awareness, COCOA-L allows the specification of both local and global QoS requirements of user tasks. Task's local QoS requirements are those related to particular requested capabilities of the user task, they are checked by COCOA-SD when selecting service advertised capabilities that semantically match requested capabilities of the user task. On the other hand, global QoS requirements are checked

by COCOA-CI when integrating service conversations and require the aggregation of QoS properties coming from the multiple advertised capabilities to be integrated.

We further presented in this article a prototype implementation and evaluation of COCOA. In this article, we have been primarily interested in evaluating the performance of COCOA-CI and the impact of introducing QoS-awareness in the composition process. Indeed, a preliminary solution for efficient semantic service discovery in pervasive environment, has previously been introduced in [21]. For evaluating a prototype implementation of COCOA-CI, we have compared its response time against the time spent for the XML parsing of services and task descriptions, which is inherent to the use of Web services and semantic Web technologies. Results show that in more realistic cases, COCOA overhead is negligible compared to XML parsing. We have further done experiments for evaluating the impact of introducing QoS-awareness in COCOA. Results show the introduction of QoS constraints improves the performance of COCOA-CI. Our ongoing research efforts include the deployment of COCOA-CI on top of an existing semantic service discovery protocol for pervasive environments (e.g., [21]), such that the composition of user tasks can be performed transparently and in a distributed manner by a set of collaborating service directories of the pervasive computing environment.

References

[1] Bernstein A. and Klein M. Towards high-precision service retrieval. In *Proceedings of The First International Semantic Web Conference(ISWC'02)*, 2002.

[2] Rohit Aggarwal, Kunal Verma, John Miller, and Willie Milnor. Dynamic web service composition in meteor-s. Technical report, LSDIS Lab, Computer Science Dept., UGA, 2004.

[3] C. Aurrecoechea, A. T. Campell, and L. Hauw. A survey of QoS architectures. *ACM/Springer Verlag Multimedia Systems Journal, Special Issue on QoS Architecture*, 3(6):138–151, May 1998.

[4] Sharad Bansal and Jose M. Vidal. Matchmaking of web services based on the daml-s service model. In *Proceedings of the second international joint conference on Autonomous agents and multi-agent systems*, 2003.

[5] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valerie Issarny. Ad hoc composition of user tasks in pervasive computing environments. In *Proceedings of the 4th Workshop on Software Composition (SC 2005). Edinburgh, UK, April 2005. LNCS.*

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

[7] A. Brogi, S. Corfini, and R. Popescu. Composition-oriented service discovery. In *Proceedings of the 4th Workshop on Software Composition (SC'05)*, 2005.

[8] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and Web service processes. *Journal of Web Semantic*, 2004.

[9] 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)*, 2004.

[10] H. V. Dijk, K. Langendoen, and H. Sips. ARC: a bottom-up approach to negotiated QoS. In *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2000)*, December, 2000.

[11] J. Flinn, S. Y. Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *Proceedings of IEEE ICDCS*, 2002.

[12] Howard Foster, Sebastian Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *IEEE International Conference on Automated Software Engineering*, 2003.

[13] S. Gurun, C. Krintz, and R. Wolski. NWSLite: a light-weight prediction utility for mobile devices. In *Proceedings of ACM MobiSys*, 2004.

[14] Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Francoise Sailhan, Rafik Chibout, Nicole Levy, and Angel Talamona. Developing ambient intelligence systems: A solution based on web services. *Journal of Automated Software Engineering*, 2004.

[15] Valerie Issarny Jinshan Liu. Qos-aware service location in mobile ad-hoc networks. In *IEEE International Conference on Mobile Data Management (MDM'04)*, 2004.

[16] M. Koshkina and F. van Breugel. Verification of business processes for Web services. Technical report, York University, 2003.

[17] 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.

[18] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing - the semantic web meets pervasive computing. In *2nd International Semantic Web Conference (ISWC2003)*, 2003.

[19] D. Menasce. Composing Web services: A QoS view. *IEEE Internet Computing*, Vol. 8.(No. 6), November/December 2004.

[20] Sonia Ben Mokhtar, Nikolaos Georgantas, and Valerie Issarny. Cocoa: Conversation-based service composition in pervasive computing environments. In *Proceedings of the IEEE International Conference on Pervasive Services (ICPS'06)*, 2006.

[21] Sonia Ben Mokhtar, Anupam Kaul, N. Georgantas, and Valerie Issarny. Efficient semantic service discovery in pervasive computing environments. In *Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06)*, 2006.

[22] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valerie Issarny. Towards efficient matching of semantic web service capabilities. In *Proceedings of the workshop of Web Services Modeling and Testing (WS-MATE'06)*, 2006.

[23] Sonia Ben Mokhtar, Jinshan Liu, Nikolaos Georgantas, and Valerie Issarny. Qos-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05)*, 2005.

[24] D. Narayanan and M. Satyanarayanan. Predictive resource management for wearable computing. In *Proceedings of ACM MobiSys*, 2003.

[25] Abhijit A. Patil, Swapna A. Oundhakar, Amit P. Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th conference on World Wide Web*, 2004.

[26] Manuel Roman, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mobile Computing and Communication Review*, 6(4), 2002.

[27] B. Sabata, S. Chatterjee, M. Davis, J. J. Sydir, and T. F. Lawrence. Taxonomy for QoS specification. In *Proceedings of Workshop on Object-oriented Real-time Dependable Systems (WORDS 97)*, Newport Beach, California, USA, 1997.

[28] Joao Pedro Sousa and David Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture*, 2002.

[29] W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: Yet another workflow language. *Information Systems*, 2004.