A Framework for Analyzing and Testing the Performance of Software Services

Antonia Bertolino[1], Guglielmo De Angelis[1], Antinisca Di Marco[2], Paola Inverardi[2], Antonino Sabetta[1], and Massimo Tivoli[2]

[1] ISTI-CNR, Pisa, Italy
{antonia.bertolino,guglielmo.deangelis, antonino.sabetta}@isti.cnr.it
[2] Università dell'Aquila
Dipartimento di Informatica, via Vetoio, L'Aquila, Italy
{adimarco,inverard,tivoli}@di.univaq.it

Abstract. Networks "Beyond the 3rd Generation" (B3G) are characterized by mobile and resource-limited devices that communicate through different kinds of network interfaces. Software services deployed in such networks shall adapt themselves according to possible execution contexts and requirement changes. At the same time, software services have to be competitive in terms of the Quality of Service (QoS) provided, or perceived by the end user.

The PLASTIC project proposes an integrated model-based solution to the development and maintenance of services deployable over B3G networks. Notably, the PLASTIC solution includes formal techniques that combine predictive and empirical evaluation of QoS-aware services.

In this paper we provide an overview of the PLASTIC approach to the assessment of QoS properties. Referring to a complex eHealth service, we first generate and analyze performance models to establish requirements for stand-alone services. Then we use an empirical technique to test the QoS of an orchestration of services even when the actual implementations of the orchestrated services are not available.

1 Introduction

The promise of the Service Oriented Architecture (SOA) paradigm is to enable the dynamic integration between applications belonging to different, globally distributed enterprises, connected through heterogeneous B3G (Beyond 3rd Generation) networks. B3G service-oriented applications, as well as communication networks and embedded systems [1], require to consider extra-functional characteristics as a critical aspect of software development [2].

The openness of the B3G environments naturally leads the SOA paradigm to pursue mechanisms for specifying the provided levels of *Quality of Service* (QoS) and for establishing *Service Level Agreements* (SLAs) on them.

In addition, context-awareness and adaptation are key features for B3G services that are to be deployed in different environments and on hardware platforms with different characteristics. Applications must be able to react to context changes and to adapt themselves to continue to provide services within the levels of QoS that were previously agreed.

Let us consider the typical case of a service provider who is to offer a certain service (S^*) to their clients according to QoS levels ratified in form of SLAs. In order to do so, the service provider could orchestrate a number of other services ($S_1 \ldots S_n$), which may be either under their direct control, or may be provided by third parties. However, usually the life-cycle of service S^* is independent of the life-cycle of the aggregated services S_i. For example, the actual implementation of some S_i could not be used during the development or the testing of S^* because they are developed concurrently with S^* or because using them would imply additional costs or undesired side-effects (e.g., undesired writes on a DB). Furthermore, the binding between S^* and S_i is defined at run-time. Therefore, assessing the properties of a service out of the properties of its constituents is a complex task, which limits the possibility to define SLAs for the developed service.

Traditional approaches applied to the development, the deployment and the maintenance of SOAs do not provide adequate support to these needs in terms of languages, methods, and tools. Nevertheless, in recent years much research has been devoted to methodologies for QoS evaluation, including *predictive* and *empirical* techniques [3].

In this respect, the main target of the PLASTIC project [4] is an integrated model-based solution supporting both the development of services deployable over B3G networks, and the definition of their related SLAs. The key idea of the project is that the QoS characteristics of the network and of the devices should be visible at the level of services. The solution proposed in PLASTIC includes formal techniques for both predictive and empirical evaluation of QoS-aware services.

Predictive techniques span methodologies and tools that can be exploited by service providers to guide the design starting from the earliest phases of the service development. These methodologies can provide predictive assessments on whether the proposed software solution is likely to meet the expected performance goals.

Also, in order to construct an efficient and effective application, developers should test it in advance in all possible network scenarios. However, at development time it is difficult to anticipate all possible configurations in which a service will be executed. An empirical solution for the off-line validation consists in providing a testbed in which the behavior of the underlying platform and of the network can be simulated in a realistic way. In particular, starting from the performance requirements of the services orchestrated to form a composite service, it is possible to automatically validate the performance requirements of the resulting composite service hence making it possible to define appropriate SLAs for it.

In this paper we provide an overview of the PLASTIC approach to the assessment of QoS (performance) properties. We describe the predictive technique to assess the QoS properties for stand-alone services under development, and the empirical technique to test an orchestration of these services when their actual implementations are not yet available. The combination of such techniques is illustrated through the design and the implementation of an eHealth service that satisfies performance requirements.

The remainder of the paper is structured as follows: Sect. 2 introduces the PLASTIC development process; Sect. 3 describes how to model an eHealth Service within

PLASTIC; Sect. 4 and Sect. 5 respectively describe the predictive and empirical approaches used to assess the QoS of the modeled eHealth example. Conclusions and future work are given in Sect. 6.

2 Development Process

In order to address in a comprehensive way the challenges in the development of B3G applications, a new development process model has been devised [5] in the context of the PLASTIC project (see Figure 1).

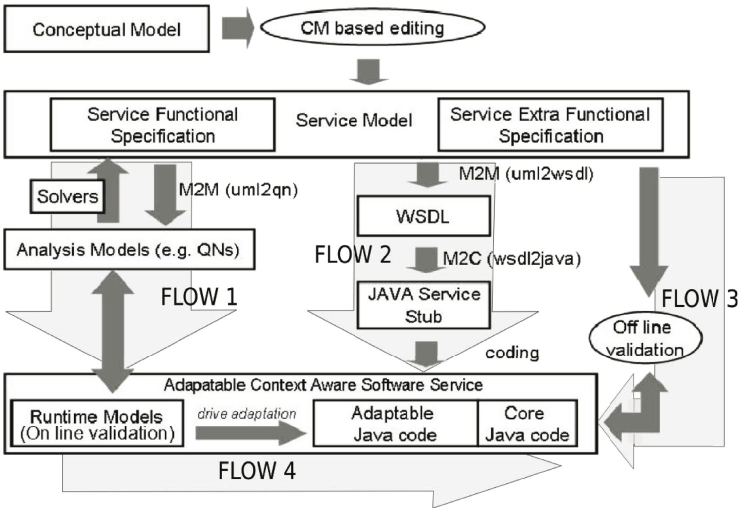


Fig. 1. The PLASTIC development process

All the activities in this process originate from the PLASTIC Conceptual Model [4,6], which provides a shared conceptual foundation for the construction of a Service Model. A Service Model involves the specification of both functional and extra-functional aspects[1] (see *Functional Service Specification* and the *Extra-Functional Service Specification* in Figure 1).

Based on such a service model, the PLASTIC process is structured into four main flows of activities (see Figure 1).

Flow 1 shows the generation of analysis models [8,9,10,11], which enable the *QoS analysis* of the service under development. This flow consists in the performance analysis process executed starting from the early phases of the software lifecycle. The aim of this activity is twofold: *(i)* to verify the service model with respect to QoS requirements, and (ii) to generate QoS models that the service can use later, at run time, in order to monitor the desired QoS and trigger adaptation when, e.g., the QoS level degrades due to possible context changes. The model-to-model (M2M) generation and

[1] We use the term *extra-functional*, as opposed to *non-functional*, following the terminology of [7].

the evaluation of the QoS models are automated and executed through a combination of tools (e.g. UML to Queueing Networks transformations engine – uml2qn), whereas the interpretation of results and feedback provision is still a human activity.

Flow 2 represents the automated generation of the skeleton implementation of the service from both M2M, and model to code (M2C) transformation engines. Specifically, this flow concerns the development of both the core code and the "adaptable" code of a service. The core code is the frozen unchanging portion of a self-adapting service (e.g., its required/provided interface). On the contrary, the adaptable code embodies a certain degree of variability making it capable to evolve (e.g., the logic of a service operation that depends on available resource constraints). This code portion is evolving in the sense that, based on contextual information, the variability can be solved with a set of alternatives (i.e., different ways of implementing a service) each of them suitable for a particular execution context. An alternative can be selected by exploiting the analysis models available at run-time.

Flow 3 on the right-hand side of the figure represents the *off-line* validation, which concerns validation at development time. In this phase services are tested in a simulated environment that reproduces functional and/or extra-functional run-time conditions.

Flow 4 concerns the *on-line* validation that consists in testing a service when it is ready for deployment and final usage. In particular, the PLASTIC validation framework supports validation during live usage stage, in which service behaviours are observed during real execution to reveal possible deviations from the expected behaviour. *On-line* validation can cover both functional and extra-functional properties.

All these four flows heavily rely on model-to-model and model-to-code automatic transformations.

The final result of this process is a deployable service code [12,13,14] that, through the support of the analysis models, has the capability to adapt to heterogeneous devices while still providing the previously agreed level of QoS. The service modeling is based on a UML profile that we have defined as a partial concrete implementation of the PLASTIC Conceptual Model.

3 Application Scenario: The eHealth Service

In this section we describe how to model an eHealth Service using the PLASTIC profile. In the following, we focus on the modelling views that contain useful information (i.e., stereotypes and tagged values) for performance analysis and testing methodologies [9,15] as we will introduce later.

We will focus on the specification of a Panic Button Scenario (PBS): the alarm is triggered when a patient's panic button is pressed in case of emergency. The eHealth Service (eHS) is in charge of handling the PBS work flow, response time and interaction among the different parties (i.e., patient, relatives or doctors) that are involved. Critical decisions, such as establishing the severity of the emergency, are taken by health specialists. An eHealth (sub-)service deployed on the patient's side monitors the vital parameters of the patient. When the patient presses the panic button, the system registers the patient's vital parameters into the eHealth database. At the same time, at the patient's side, a beeper is turned on to notify the patient that the alarm is being handled by the Service Manager. An internal counter is started to handle an "unknown" event. The eHealth database is scanned to search the most suitable supervisor to attend the patient, and a request is sent to the call center. Depending on the response of the call center, an alternative supervisor is requested or the event is assigned. The authorized supervisor is driven to set up the severity of the event by means of phone call. In some cases, the supervisor can also interact with the patient by means of cameras (e.g. the call fails, the patient cannot interact with the voice). Once the severity is set, an appropriate service must be composed to provide both, medical attention and response time according to the specific request, illness or accident.

The PLASTIC Service Model for eHS is composed of several views used to structure the UML design in packages. These views span from *requirement* view to *implementation* (i.e., component-oriented implementation) and *deployment* views, through the *service* view. Due to the lack of space, in the following we detail only the views that allow performance analysis and testing. For further details we refer to [4] where the service model is completely described with minor modifications.

Service View. The definition of the services that build up the PLASTIC application is given from both the structural and the behavioral perspectives. In particular, a Structural View is given by means of Service Description Diagrams (SDescrD) that show the *ServiceDescriptions* (e.g., ServiceManager, Patient Interactor Service) that may be combined on demand (*ServiceComposition* or *ServiceUsage* dependency from composite client to composite supplier services) and collaborate to provide the mobile eHealth Service, as illustrated in Figure 2.

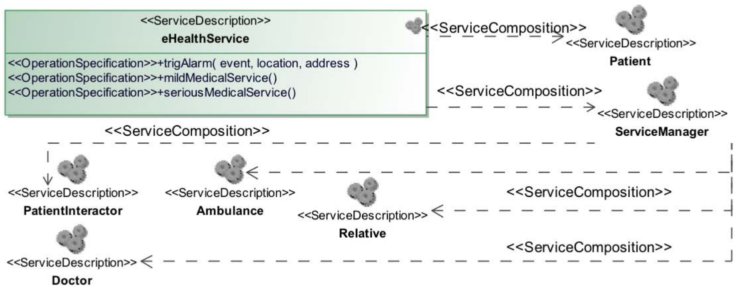


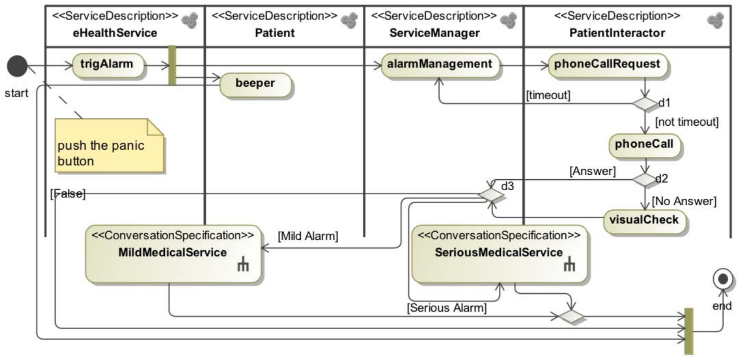
Fig. 2. The eHealth Service Description Diagram

The key concept is the *ServiceDescription*, which is the base structural unit for the description of PLASTIC applications at service level. It is a stereotype extending the UML2 Interface meta class. It provides some *OperationSpecifications* that, together, define what the user can request from its PLASTIC enabled device (e.g., doctors' or call center staff's laptop or PDA). For the sake of clarity, in Figure 2, we show the *OperationSpecifications* only for *eHealthService* and omit the ones for the other *ServiceDescriptions*.

Once all *ServiceDescriptions* have been specified, a number of business process descriptions have to be provided. Each of them describes the interactions (i.e., service orchestration) between the *ServiceDescriptions* identified in the SDescrD. These interactions model the behavior of a composite service operation. The composite service is the one obtained by composing the services in the SDescrD as specified by the set of business process descriptions. In particular, for each usage scenario of a composite service (e.g., the AlarmHandling use case of the PLASTIC eHealth service application), a Business Process Description Diagram (BPDD) has to be specified to describe the interactions (as Actions that refer to the already specified *OperationSpecification*) among the involved *ServiceDescriptions*.

As introduced above, one of the role played by the BPDD is describing the orchestration of different services. In this sense, the BPDD acts as the BPEL specification [16]. Nevertheless, BPDD also defines a well-structured set of annotations and tags that can be used in order to stereotype the elements described into the models. Differently from BPEL, such annotations can be instantiated at design time and then exploited for extra-functional analysis (e.g. performance or reliability analysis).

In Figure 3 the *ConversationSpecification* that realizes the behavior of the AlarmHandling use case is shown. At the bottom of Figure 3 there are two behaviors, Serious and Mild Medical services, that refer to two corresponding (sub-)BPDDs.

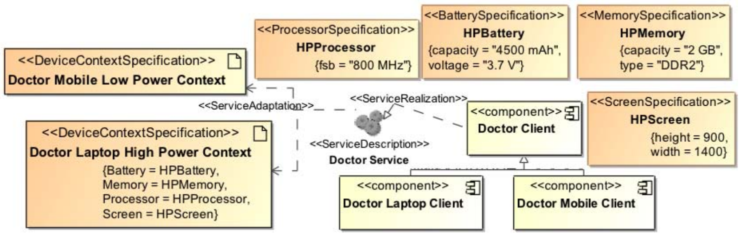


<<ServiceDescription>> eHealthService
<<ServiceDescription>> Patient
<<ServiceDescription>> ServiceManager
<<ServiceDescription>> PatientInteractor
start
trigAlarm
beeper
alarmManagement
phoneCallRequest
[timeout]
d1
[not timeout]
phoneCall
[Answer]
d2
[No Answer]
visualCheck
d3
push the panic button
[False]
<<ConversationSpecification>> MildMedicalService
[Mild Alarm]
<<ConversationSpecification>> SeriousMedicalService
[Serious Alarm]
end

Fig. 3. The BPDD representing the service orchestration

Component View. In PLASTIC, a service can be implemented by one or more software components and, in turn, a software component can be used to implement one or more services. The PLASTIC Profile provides modeling constructs aimed at describing the component-based software architecture that implements a given service. Such description is organized in a Component View in turn distinguished into Structural View and Behavioral View.

Service Specification Diagrams (SSD) are introduced for defining the components implementing a *ServiceDescription*. Such diagrams are extensions of UML2 Class Diagrams and a number of new modeling constructs are provided, as detailed in Figure 4. The *ServiceRealization* stereotype is introduced to link *ServiceDescription* stereotyped interface and *ComponentSpecification* stereotyped components to describe how services are implemented in terms of software components. Moreover, by means of the SSD the designer can specify the contexts in which the service will be able to adapt. In particular *DeviceContextSpecification* elements are used to describe the possible devices (e.g., doctor's mobile or laptop). Each tag of such stereotypes refers to an available resource



<<DeviceContextSpecification>>
Doctor Mobile Low Power Context
<<ProcessorSpecification>>
HPProcessor
{fsb = "800 MHz"}
<<BatterySpecification>>
HPBattery
{capacity = "4500 mAh", voltage = "3.7 V"}
<<MemorySpecification>>
HPMemory
{capacity = "2 GB", type = "DDR2"}
<<ServiceAdaptation>>
<<ServiceRealization>>
<<component>>
Doctor Client
<<ScreenSpecification>>
HPScreen
{height = 900, width = 1400}
<<DeviceContextSpecification>>
Doctor Laptop High Power Context
{Battery = HPBattery, Memory = HPMemory, Processor = HPProcessor, Screen = HPScreen}
<<ServiceDescription>>
Doctor Service
<<component>>
Doctor Laptop Client
<<component>>
Doctor Mobile Client

Fig. 4. The Service Specification Diagram for the Doctor Service

specification of the Resource package. The *DeviceContextSpecification* is then linked to adaptable services by means of *ServiceAdaptation* relationships.

Once the *ComponentSpecifications* of the components implementing the service being modeled have been given, their interactions have to be specified. The PLASTIC profile provides the designer with Elementary Service Dynamics Diagrams (ESDD) to model the interactions among the involved components (specified in the structural view by means of *ComponentSpecification* elements). Each ESDD is a suitably stereotyped UML sequence diagram annotated with information useful for performance analysis purposes, e.g., *latency*, *worst-case execution time*, *reliability (probability of failure)*, or *maximum number of simultaneous invocations* of a component operation.

Figure 5 shows the Elementary Service Dynamics Diagram, i.e., the interaction between component instances providing the AlarmManagement uml.Action defined in Figure 3. Additional information (i.e., stereotypes with their own tags) is introduced for the sake of performance analysis.

4 Performance Model Generation and Analysis

In PLASTIC a service can be either a composition of other services or a basic one implemented by an assembly of components. For a composite service, the performance analysis can be conducted both at the service composition level (abstract view of the service) and at the component level (detailed view of the service). For a stand-alone service, instead, the analysis can be only conducted at the component level.

The analysis process is composed by three steps: *(i)* generation of performance models from the Service Model through Model-to-Model transformation; *(ii)* evaluation of the generated performance models through solvers to obtain performance indices (e.g., response time); *(iii)* interpretation of the performance indices and possible production of feedbacks on the Service Model to improve the performance. The performance model that the service may use at run time is the last one generated during the analysis process.

In the following we first briefly recall what the SAP•one methodology [9] is (see Sect. 4.1) and then we show how to use it to analyze service performance at the component level. On the result of this analysis a provider might base the definition of the SLA of a service operation. In Sect. 4.2 we describe the analysis process of the alarmManagement operation of the ServiceManager service.

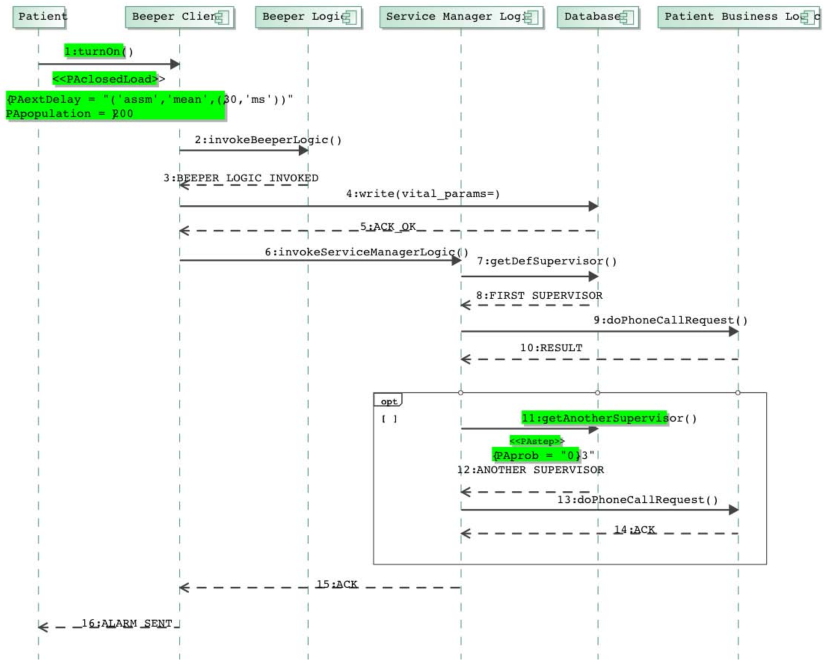


Fig. 5. The Elementary Service Dynamics Diagram for the Alarm Management

We used the approach several times to define the SLA concerning the mean response time for the operations of the services that have been composed (see Figure 3) into the *PLASTIC eHealth composite service* in order to implement the AlarmHandling functionality.

4.1 The Used Analysis Approach and Tools

The performance analysis is carried out by means of two tools: MOSQUITO and WEASEL.

MOSQUITO (MOdel driven conStruction of QUeuIng neTwOrks) [17] is a *model transformation* tool that generates Queuing Networks (QNs) starting from the PLASTIC Service Model. The model creation in MOSQUITO is based on two different methodologies: SAP•one [9] and Prima-UML [8]. In this work we use only the SAP•one methodology, hence details on the Prima-UML approach are omitted.

The SAP•one methodology, implemented by MOSQUITO, defines translation rules that map UML architectural patterns (identified in the Component View) into QN patterns. The target model is generated by composing the identified QN patterns suitably instantiated according to the particular scenario. To carry on the performance analysis, additional information generally missing in the software architecture description needs to be annotated on the software system model. Such data are strictly related to the performance aspects and are used both in the QN parameterization and in the workload definition. They are: the operational profile of the system that models the way the system will be used by the users (i.e. the distribution of frequencies of invocation of service's use cases by the service consumer); the workload entering the system as the estimated number of requests made to system components (modelled as service centers); the service demand of a request to the system components; the performance characterization of the system components represented by attributes such as service rate, scheduling policy, waiting queue capacity.

SAP•one associates each QN service center to a software component, and the QN customers represent the different requests that users make to the software system. The QN topology reflects the one of the Service Specification Diagram. Each ESDD is processed to lump the behavior that it represents into a class of jobs of the QN (i.e. a chain). In other words, a job traverses the network following the behavior described by the diagram it comes from. The workload of each chain is extracted from the annotations in the Use Case Diagram.

After that, by using MOSQUITO (hence following the SAP•one methodology), a QN model has been built, the WEASEL tool is used to solve the generated QN model in order to predict performance indexes. WEASEL [18] (a WEb service for Analyzing queueing networkS with multiplE soLvers) offers a Web Service that solves QN models specified in PMIF [19] format, using several off-the-shelf QN solvers (e.g., MVA-QFP [20] and SHARPE [21]). The performance measures are presented to the client as a text file in the original output format of the selected tool.

4.2 Performance Analysis of the alarmManagement

The alarmManagement action in Figure 3 must satisfy the following performance requirement[2]: *the average response time of alarmManagement must not exceed 10 seconds when the triggered alarms in the system are less than 100.*

To perform the analysis, we have used the SAP•one methodology. This means that the service has been considered at the component level where the alarmManagement action is implemented by components' interactions as specified in the Component View of the service model [4].

We generated the performance model (at the software architecture level) of the alarmManagement design by means of MOSQUITO using the SAP•one approach.

The obtained queuing network has been then evaluated via WEASEL where the selected solution technique was Exact MVA implemented in the MVA Queuing Formalism Parser [20].

For the *FirstDesign*, we analyze the mean response time as the number of alarm requests arriving to the system grows from 50 to 300. The proposed design did not satisfy the requirement. The system response time reaches 10 seconds with only 65 alarms. Moreover, the analysis highlighted that a database component is the bottleneck of this system design, hence to improve the system response time we should lighten the load offered to the database.

We produced the second design alternative by modifying the alarmManagement design as follows. In the dynamics model of Figure 5, the *Service Manager Logic* accesses the database twice to retrieve information on the two supervisors of the patient in trouble. This can be optimized by introducing in the database interface a new method that

[2] This requirement has been agreed by the customer together with the domain experts.

retrieves the information of all the supervisors of the patient. The call of this method substitutes the first call of the method used to get information about only one single supervisor, while the second call can be removed. In this way we reduce the load to the database.

On this design alternative, i.e., *SecondDesign*, we repeated the analysis and the requirement was satisfied since for the *alarmManagement* and the *visualCheck* operations we predicted a mean response time respectively equal to $7.25s$ and to $4.83s$ when the number of triggered alarms is 100. On the other hand, the mean system response time was 10 seconds when the *alarmManagement* operation handles 126 concurrent alarms.

5 Performance Testing

Following the design and analysis stages described above, the subsequent step in developing B3G services consists in early testing them within a simulated environment, which we referred to in Figure 1 as *off-line* validation. When developing a service orchestration, the composition of the external services must be tested both to validate that the implementation respects the functional contracts in place, and to evaluate if it actually meets the expected quality levels. Clearly the QoS offered by a composition not only depends on its implementation, but is also affected by the quality levels of the composed services. Furthermore, when the interaction happens through complex middlewares, the application of analytical techniques such as the one described above to derive the exposed extra-functional properties is not always feasible, since the modeling of such infrastructures is particularly difficult and error prone. This task becomes even harder when the analytical models of the platform have to be defined from scratch.

Testers may rely on empirical approaches when all the composed services are available, and can be also arbitrarily accessed at development time for testing purposes. However, in general this solution is applicable only in few lucky cases. In fact, commonly at least some of the external services are either not available at all (for instance simply not implemented, yet), or their usage comes along with unwanted side-effects (for instance utilization fees or database modifications). To circumvent this problem, in PLASTIC we provide support to the automatic derivation of testbeds to be used in the place of the real composed service.

In the following we present the proposed approach for the empirical evaluation of QoS properties of a composite B3G service and its application to the eHealth example described in Sect. 3. The approach relies on the specification of reasonable agreements on the extra-functional properties.

In a global view of the PLASTIC process, the performance bounds expected at design time, as derived by the analytic approach presented in Sect. 4.2, are exploited to infer the agreements used for testing the implementation of the orchestration described in Figure 3.

5.1 Puppet

As discussed in Sect. 4, predictive approaches are crucial during the design and the development of a software system, to shape the quality of the final product [22]. But increasingly modern applications are deployed over complex platforms (i.e., the middleware), which introduce many factors influencing the QoS and not always easy to model in advance. In such cases, empirical approaches, i.e., evaluating the QoS via run-time measurement, could help smoothing platform-dependent noise. However, such approaches require the development of expensive and time consuming prototypes [23], on which representative benchmarks of the system in operation can be run.

For example, testers may be interested in assessing that a specific service implementation can afford the required level of QoS (e.g., latency and reliability) when playing one of the roles in a specified choreography or when used in composition with other services (orchestration).

As we discussed in [15], there is large room for the adoption of empirical approaches when model-based *code-factories* can be used to automatically generate a running prototype from a given specification. In particular, as we argued in [15,24,25], given the high availability of standardized computer processable information, Web Services (WSs) and related technologies (e.g. WSDL, WS-BPEL, WS-CDL, WS-Agreement, WSLA) yield very promising opportunities for the application of empirical approaches to QoS evaluation.

In this direction, PUPPET (Pick UP Performance Evaluation Testbed) [15] is a *code-factory* which realizes the automatic derivation of testbeds for evaluating the desired QoS characteristics for a service under development, before it is deployed.

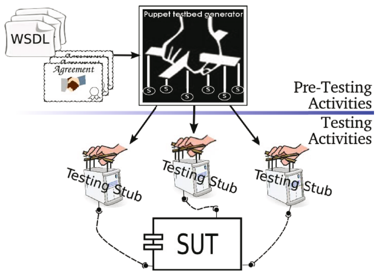


Fig. 6. PUPPET: The approach

PUPPET relies on the availability of the QoS specification of both the service under evaluation and the interacting services. Such assumption is in line with the increasing adoption of formal contracts to establish the mutual obligations among the involved parties and the guaranteed QoS parameters, which is referred to as the SLA for the WSs. For example, Figure 6 depicts the case when 3 different stubs are generated during pre-testing activities. Each stub is derived from a model describing the public interface of the remote services (WSDL), and the contracted SLA. During the testing activities, testers can bind the resulting stubs to the Service Under Test (SUT) using them as a testbed.

In [24], PUPPET was extended with a module able to include into the stubs also the emulation of the supposed functional behavior. In this case, the functional behavior of a service is described by means of the *Symbolic Transition System* (STS) models as described in [26]. Specifically, for each received invocation, the service stub can query the STS model and choose one of the possible functionally correct results, sending it back to answer the service client request.

Also, possible dynamic transformation of the network topology and, consequently, of the configuration of the environment must be taken into account when developing a networked service, especially in the off-line testing phase. In B3G, the most typical context change is due to the movement of nodes hosting services. Correspondingly, latest work extends PUPPET adding a module that plugs into the generated stubs the mobility emulation of the node hosting the service. The detailed description of this module is given in [25].

5.2 Performance Testing of the eHealth Service

Let us consider again the BPDD in Figure 3, and let us assume that the orchestration it describes is going to be implemented in parallel with the development of the four services it composes (i.e. eHealthService, Patient, ServiceManager, PatientInteractor). The problem that we want to solve here is how to test the performance of the orchestrated service even when just *models* of the composed services are available but the actual implementations are not (or we do not want to access to avoid undesired side-effects).

A possible solution to this problem is to use PUPPET to build stubs of the orchestrated services. As mentioned in Sect. 5, PUPPET ensures by construction that the extra-functional behavior exhibited by each generated stub conforms to the guaranteed levels expressed in a SLA.

```
1  ...
2  <wsag:GuaranteeTerm ... wsag:Obligated="
        ServiceProvider">
3   <wsag:ServiceScope wsag:ServiceName="
         ServiceManager">
4    <puppetScope:PuppetScope>
5     <puppetScope:Method>
6      <NameMethod>alarmManagement</
           NameMethod>
7     </puppetScope:Method>
8    </puppetScope:PuppetScope>
9   </wsag:ServiceScope>
10  ...
11  <wsag:ServiceLevelObjective>
12   <puppetSLO:PuppetSLO>
13    <puppetSLO:Latency>
14     <value>14500</value>
15     ...
16    </puppetSLO:Latency>
17   </puppetSLO:PuppetSLO>
18  </wsag:ServiceLevelObjective>
19 ...
```

–A–

```
1  ...
2  public class ServiceManagerSoapBindingImpl {
3  ...
4   public void alarmManagement ( ...
5   ...
6    Density D = new Density();
7    Double sleepValue = D.gaussian(14500-(
         System.currentTimeMillis()-
         cOmMoNinvocationTime));
8     if (sleepValue >= 0)
9      try {
10      Thread.sleep(sleepValue.longValue());
11     } catch (InterruptedException e) {}
12 ...
```

–B–

Fig. 7. `alarmManagement` : SLA and Generated Code into the Service Stub

Figure 7.A shows an example on how the QoS indexes obtained by the performance analysis prediction can be instantiated in an SLA. Specifically, `line 2` asserts that the term of the SLA is a service-side constraint that is applicable to the service ServiceManager (`line 3`) on the operation `alarmManagement` (`line 6`). Figure 7.B depicts the portion of the stub that PUPPET automatically generates with respects to the given SLA. The operational semantic we give in PUPPET to emulate the clauses on latency is defined in terms of a random and normally distributed sleeping period drawn in the range between 0 and the maximum expected time by the operation [15,24]. In this example, the stub emulates a service guaranteeing that its mean elapsed time conforms to the index defined in Sect. 4. Thus in Figure 7.A at `line 14` we imposed the max elapsed time as $14500ms$ which means emulating the mean response time in $7250ms$.

Note that in the stubs either the combined emulation of different QoS properties (e.g. latency with reliability), or the emulation of other aspects of the service (e.g. the functionality or the mobility) may affect the emulation of the latency clause. PUPPET solves these issues reducing at run-time the maximum latency time with the time elapsed emulating other aspects. Such delta is calculated as the difference between the timestamp executing instructions at `line 7` of Figure 7.B and the timestamp marked when the operation is called.

The goal of the approach presented here is limited to evaluating technical constraints that form the basis on which a SLA can be defined. However, in a more general setting, a SLA is more than just (a set of) technical constraints. Indeed, a number of non-technical aspects (legal clauses, penalties for violations, business strategies) play an important role in making contractually-agreed service provision a viable solution. A more general discussion of the problems related to establishing and enforcing a SLA as a *legal contract* is beyond the scope of this work, but can be found in [27].

6 Conclusion

B3G networks are characterized by a distributed, heterogeneous and mutable nature, which poses difficult problems in developing service-oriented systems. An additional challenge arises when such systems must meet precise QoS requirements.

Several European research projects, such as ASG [28], COMET [29], MADAM [30], MUSIC [31], SeCSE [32], recognized that it is certainly no longer possible to propose solutions without adequate specification and validation of QoS features, especially in heterogeneous and networked services contexts.

In particular, the SeCSE project exploits service specifications describing semantics and QoS information in order to guide the test phase, proposing tools for the automatic generation and execution of test cases. The exploitation of QoS-awareness in the context of highly dynamic systems is also a key feature of the MADAM project. The objectives of the project include the development of an adaptation theory and a set of reusable adaptation strategies and mechanisms to be enacted at run-time. Context monitoring is used as the basis for decision making about adaptation, which is managed to a large extent by generic middleware components.

The PLASTIC project tackled these challenges by defining a platform and introducing a comprehensive process for developing lightweight and QoS-aware services. This process spans the design, the predictive analysis and the validation of services, taking into account both functional and extra-functional characteristics.

With respect to the problem of the assessment of QoS properties, this paper shows how to fruitfully combine predictive and analytical approaches with empirical ones. In particular, it described how to link the results of those phases that are typical of the earliest stages of the service development process (i.e. performance model generation and analysis) with the input of those phases that characterized the latest parts of the development process (i.e. testing techniques and testing support tools).

It is important to remark that such integration was possible because all the common entities, the relations among the entities, as well as the artifacts and the extra-functional properties they model were formally defined and structured in the PLASTIC conceptual model [4,6]. In such a way it is ensured that the information captured and defined starting from the early phases of the software lifecycle can be referred and reused at each stage of the software development.

The application of the described approaches to a real world case study will further permit to refine and validate the whole framework. Specifically, in collaboration with the industrial partners of the PLASTIC project, we are applying the PLASTIC development process on wider and more complex case studies.

Acknowledgements. The authors wish to thank Andrea Polini for his contribution to the research on PUPPET, and Luca Berardinelli for his contribution in modeling the eHealth example. This work was supported in part by the PLASTIC Project (EU FP6 STREP n. 26955) and in part by the TAS^3 Project (EU FP7 CP n. 216287).

References

1. Bertolino, A., Bonivento, A., De Angelis, G., Sangiovanni Vincentelli, A.: Modeling and Early Performance Estimation for Network Processor Applications. In: Proc. of 9th MoDELS. Springer, Heidelberg (2006)
2. Ludwig, H.: WS-Agreement Concepts and Use – Agreement-Based Service-Oriented Architectures. Technical report, IBM (2006)
3. Woodside, M., Franks, G., Petriu, D.: The future of software performance engineering. In: FOSE 2007: 2007 Future of Software Engineering, pp. 171–187. IEEE Computer Society Press, Los Alamitos (2007)
4. PLASTIC Project: (EU FP6 STREP n. 26955), `http://www.ist-plastic.org`
5. Autili, M., Berardinelli, L., Cortellessa, V., Di Marco, A., Di Ruscio, D., Inverardi, P., Tivoli, M.: A development process for self-adapting service oriented applications. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 442–448. Springer, Heidelberg (2007)
6. Autili, M., Cortellessa, V., Di Marco, A., Inverardi, P.: A conceptual model for adaptable context-aware services. In: WS-MaTe 2006 (2006)
7. Bass, L., Clements, P., Kazman, R.: Quality Attributes. In: Software Architecture in Practice, ch. 4, pp. 75–91. Addison-Wesley, Reading (1998)
8. Cortellessa, V., Mirandola, R.: PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams. Science of Computer Programming 44(1), 101–129 (2002)
9. Di Marco, A.: Model-based Performance Analysis of Software Architectures. PhD thesis, University of L'Aquila (2005)
10. Di Marco, A., Mascolo, C.: Performance Analysis and Prediction of Physically Mobile Systems. In: ACM WOSP, Buenos Aires (Argentina) (2007)
11. Cortellessa, V., Singh, H., Cukic, B.: Early reliability assessment of UML based software models. In: ACM WOSP, pp. 302–309 (2002)

12. Inverardi, P., Mancinelli, F., Nesi, M.: A declarative framework for adaptable applications in heterogeneous environments. In: ACM SAC (2004)
13. SEA Group: (The Chameleon Project), `http://www.di.univaq.it/chameleon/`
14. Autili, M., Di Benedetto, P., Inverardi, P., Mancinelli, F.: A resource-oriented static analysis approach to adaptable Java applications. In: Proc. of CORCS 2008 (IEEE/COMPSAC 2008). IEEE Computer Society Press, Los Alamitos (to appear, 2008)
15. Bertolino, A., De Angelis, G., Polini, A.: A QoS Test-bed Generator for Web Services. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 17–31. Springer, Heidelberg (2007)
16. IBM: BPEL4WS, Business Process Execution Language for Web Services, v.1.1 (2003)
17. MOSQUITO: (User manual), `http://sealabtools.di.univaq.it/SeaLab/MosquitoHome.html`
18. WEASEL: (User manual), `http://sealabtools.di.univaq.it/SeaLab/Weasel/`
19. Smith, C.U., Llado, C.M.: Performance model interchange format (pmif 2.0): XML definition and implementation. In: QEST 2004 Proceedings, pp. 38–47. IEEE Computer Society Press, Los Alamitos (2004)
20. Chereddi, C.: Mean Value Analysis for Closed, Separable, Multi Class Queueing Networks with Single Server & Delay Queues (2006)
21. Sahner, R.A., Trivedi, K.S.: SHARPE: Symbolic Hierarchical Automated Reliability and Performance Evaluator, Introduction and Guide for Users (2002)
22. Smith, C., Williams, L.: Performance Solutions: A practical Guide To Creating Responsive, Scalable Software. Addison Wesley, Reading (2001)
23. Liu, Y., Gorton, I.: Accuracy of Performance Prediction for EJB Applications: A Statistical Analysis. In: Gschwind, T., Mascolo, C. (eds.) SEM 2004. LNCS, vol. 3437, pp. 185–198. Springer, Heidelberg (2005)
24. Bertolino, A., De Angelis, G., Frantzen, L., Polini, A.: Model-based Generation of Testbeds for Web Services. In: Suzuki, K., Higashino, T., Hasegawa, T., Ulrich, A. (eds.) TestCom/-FATES 2008. LNCS, vol. 5047, pp. 266–282. Springer, Heidelberg (2008)
25. Bertolino, A., De Angelis, G., Lonetti, F., Sabetta, A.: Let The Puppets Move! Automated Testbed Generation for Service-oriented Mobile Applications. In: Proc. of the 34th €μ-SEAA, Parma, Italy. IEEE Computer Society Press, Los Alamitos (to appear, 2008)
26. Frantzen, L., Tretmans, J., Willemse, T.A.C.: A Symbolic Framework for Model-Based Testing. In: Havelund, K., Núñez, M., Roşu, G., Wolff, B. (eds.) FATES 2006 and RV 2006. LNCS, vol. 4262, pp. 40–54. Springer, Heidelberg (2006)
27. Skene, J., Skene, A., Crampton, J., Emmerich, W.: The Monitorability of Service-Level Agreements for Application-Service Provision. In: Proc. of WOSP 2007, pp. 3–14 (2007)
28. ASG: (EU IST FP6), `http://asg-platform.org/`
29. COMET: (EU IST FP6), `https://www.comet-consortium.org/`
30. MADAM: (EU IST FP6), `http://www.ist-madam.org`
31. MUSIC: (EU IST FP6), `http://www.ist-music.eu/`
32. SeCSE: (EU IST FP6), `http://secse.eng.it`