Business Process Adaptability Metrics for QoS-Based Service Compositions

Raffaela Mirandola, Diego Perez-Palacin, Patrizia Scandurra, Michele Brignoli, Andrea Zonca

Business process adaptability metrics for QoS-based service compositions*

Raffaela Mirandola[1] Diego Perez-Palacin[1] Patrizia Scandurra[2]
Michele Brignoli[2] Andrea Zonca[2]

[1] Politecnico di Milano, Italy
{raffaela.mirandola,diego.perez}@polimi.it
[2] Università degli Studi di Bergamo, DIGIP, Dalmine (BG), Italy
patrizia.scandurra@unibg.it {m.brignoli,a.zonca}@studenti.unibg.it

Abstract. Modern service-oriented software applications, like those envisioned in cloud computing scenarios, operate in highly dynamic and often unpredictable environments that can degrade their quality of service. Therefore, it is increasingly important to efficiently and effectively manage the adaptation of such service compositions while guaranteeing quality attributes, such as availability, performance or cost. Within this context, software metrics to quantify the adaptability of a business process in orchestrating distributed services are highly demanded in conjunction with techniques for evaluating other system quality attributes. This paper proposes a set of software metrics to quantify the adaptability of a service-oriented application when services are composed dynamically trough a business process. The paper also proposes an approach for analyzing tradeoffs between the application adaptability and a quality of service such as availability. The feasibility of the approach is illustrated through a case study carried out with a tool we have developed.

1 Introduction

The SOA promise of agility and flexibility is recognized in the ability to change the business processes as market changes. To this end, it introduces a separate layer in the architecture, the *Business Process Layer*, for business processes and flows. This layer covers process representation and composition, and provides building blocks for orchestrating or choreographing the set of required atomic or composite services from the underlying *Service Component Layer*. Loosely-coupled services are aggregated to constitute a business process aligned with business goals and able to rapidly change as the market condition changes [5].

Modern service-oriented software applications, like those envisioned in cloud computing scenarios, are increasingly reliant on business processes built from multiple distributed software services that must be suitably composed to meet some specified functional and non functional requirements. These service-oriented

* This work is supported in part by the Italian Ministry of Research within the PRIN project GenData 2020 and by the EU-FP7-ICT-610531 SeaClouds project.

applications are often embedded in dynamic contexts, where requirements, environment assumptions, and usage profiles continuously change. Ergo, a key requirement for software is becoming the capability to adapt its behavior dynamically, in order to keep providing the required quality of service (QoS). Without adaptation, an application is prone to degrade performance because of faulty components, messages lost between services or delays due to an increasing number of users. Using adaptation, the application can change, for example, some of the services it uses or its overall service composition [3,2,9]. However, guaranteeing software adaptability can influence other quality attributes such as performance, reliability or maintainability and in the worst case, improving the adaptability of the system could decrease other quality attributes. A key challenge for the software engineering community is therefore how to efficiently and effectively manage such dynamic service compositions while guaranteeing QoS. Within this context, software metrics to quantify the adaptability of a business process[3] in orchestrating distributed services are needed in conjunction with techniques for evaluating other system quality attributes, like availability, reliability, performance, cost, etc. In [11], a set of software adaptability metrics are defined at the architectural level of a service-oriented software application to quantify the adaptability of a static assembly (or architecture) of service-oriented components. An approach for evaluating tradeoffs between the system adaptability and other system quality attributes, is also presented to fulfill also global QoS.

This paper introduces a new set of metrics that complement the previous ones defined in [11] in order to quantify the adaptability of a service-oriented application when services are composed dynamically trough a business process. Besides the metrics that enable comparison of process-based service compositions, we also studied a possible relationship between the business process adaptability and the satisfiability of a given quality requirement. If such relationship exists, then service compositions offering best trade-off, between adaptability and the target requirement, can be chosen. This approach allow us to evaluate different concrete business processes in order to select the one that best fits the quality requirements. In this paper, we present the results of such a study by considering availability as target quality. The architecture of a supporting tool and a case study are also presented to exemplify the overall approach.

The remainder of this paper is organized as follows. Section 2 proposes a set of metrics for quantifying SaaS adaptability at the business process level. Section 3 presents our approach for relating adaptability and a single quality attribute. Section 4 presents an example of service-oriented application used to exemplify the proposed approach. Section 5 describes a a tool to automate metrics calculation. Section 6 reviews the works related to our approach. Finally, Section 7 concludes the paper.

[3] Hereafter, we use the term "process adaptability" to denote the variability degree of a process in selecting concrete services. This vision is different from the broader concept of process adaptability in the context of self-adaptive systems [4,6,13].

2 Business process adaptability quantification

This section defines a set of metrics to quantify the adaptability of a business process and its constituents. We adopt the OASIS BPEL (Business Process Execution Language)[4] as the de facto standard to specify business processes and realize them concretely. We assume the reader is familiar with BPEL constructs.

2.1 Process activity tree

For defining and computing metrics, we first introduce a tree-structure representing a BPEL process. Let p be the business process for a compound service. We define the *Process Activity Tree* of p as the structure $T_p = (V_p, E_p)$ where V_p is the set of nodes representing the BPEL activities of p and E_p is the set of edges representing the nesting relationships among the BPEL activities. Specifically, an internal node represents a BPEL structured activity in the set $\{sequence, switch, while, flow, pick\}$. Similarly, a leaf node is associated to a BPEL atomic action in the set $\{invoke, assignment, receive, reply\}$.

Let n be the number of different elementary services $s_i | i = 1, ..., n$ orchestrated by a process p. For each invocation of an elementary service s_i (i.e., an *invoke* leaf node), either in an asynchronous or synchronous manner, several *concrete services* (or *service instances*) that have been defined as partners may exist that match the description of s_i. We assume that all the instances available for a service s_i are functionally compliant with it, i.e., each instance provides at least all the capabilities provided by s_i and require at most all the capabilities required by s_i. Instances of the same service may differ for QoS values (such as cost and availability characteristics).

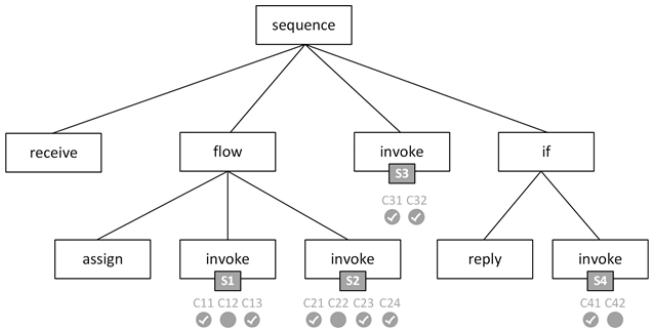

Fig. 1. Example of a BPEL activity tree

Figure 1 shows an example of BPEL tree for a compound service realized by the orchestration of four elementary software services ($n = 4$). The activity tree

[4] `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

also shows the service instances available for each service invoked. We assume the existence of n sets of "used" service instances UC_i in the process p, where service instances in each set UC_i are the ones that provide s_i ($UC_1 = \{C11, C12\}$, $UC_2 = \{C21, C23\}$, $UC_3 = \{C31, C32, C34\}$ and $UC_4 = \{C41\}$, in Fig. 1); the existence of n sets of service instances C_i, each C_i includes the instances that can provide s_i ($C_1 = UC_1$, $C_2 = UC_2 \cup \{C22\}$, $C_3 = UC_3 \cup \{C32\}$ and $C_4 = UC_4 \cup \{C42\}$ in Fig. 1).

2.2 Process Adaptability Index

The proposed metric measures the adaptability of a business process in terms of the average number of service choices made per each activity. We make the assumption that the services orchestrated by the process are stateless. We postpone as future work the extension of such a metric for stateful services.

Process adaptability index (PAI) is a metric that quantifies the degree of adaptability of a BPEL process definition. It measures the adaptability of a process by relating the number of service instances used to make up the process with the number of service instances that the most adaptable process would use:

$$PAI \in \mathbb{Q}\{0..1\} \quad | \quad PAI_p = \frac{EAI_{root(T_p)}}{EAI_{root(T_{map})}}$$

where $EAI_{root(T_p)}$ and $EAI_{root(T_{map})}$ are, respectively, the **Element Adaptability Index (EAI)** for the root of T_p and the one for the root of the activity tree T_{map} of the most adaptable process *map*.

The value of the metric PAI ranges between zero and one. A value of one means that the process is using all existing instances for each service, and then its adaptability is already to the maximum. A value close to zero means that the market offers few choices to increase the process adaptability.

To complete the definition of the metric PAI, we define the adaptability index for the nodes of the process activity tree. Starting from the root node of a process activity tree, a recursive traversal calculates the EAI of every node depending on the node type and handles a leaf node (at the bottom) as the base case.

Node *invoke*. The EAI index of a leaf node *invoke* for a service s_i can be expressed mathematically as follow:

$$EAI \in \mathbb{N} \quad | \quad EAI_{invoke\ s_i} = |UC_i|$$

where $|UC_i|$ denotes the number of concrete service instances used to provide the service s_i. This index corresponds to the metric *Absolute adaptability of a service (AAS)* defined in [11] that uses a natural number to quantify how much adaptable a service is by counting the different alternatives to execute the service (1 no adaptable, >1 adaptable), where the service adaptability grows according to the number of concrete service instances able to provide it. Referring to the example in Fig. 1, we observe that $EAI_{invoke\ s_1} = 2$, $EAI_{invoke\ s_2} = 3$, $EAI_{invoke\ s_3} = 2$, and $EAI_{invoke\ s_4} = 1$.

Node n **in** $\{assignment, receive, reply\}$**.** $EAI_n = 0$
These node types are neutral w.r.t. the adaptability quantification. Referring to the example in Fig. 1: $EAI_{receive} = 0$, $EAI_{reply} = 0$, and $EAI_{assign} = 0$.

Node *flow***.** The *flow* construct provides a kind of parallelism of interaction activities. The EAI index of a node *flow* can be expressed mathematically as follow:

$$EAI \in \mathbb{Q}^+ \quad | \quad EAI_{flow} = \frac{\Sigma_{j=1}^{m} EAI_{a_j}}{m}$$

where m is the number of child nodes and $a_j | j = 1, ..., m$ denotes the j-th child activity within the scope of the flow node. Referring to the example in Fig. 1:

$$EAI_{flow} = \frac{EAI_{assign} + EAI_{invoke\ s2} + EAI_{invoke\ s3}}{3} = \frac{0+2+3}{3} = 1.67.$$

Node *switch***.** The *switch* construct expresses conditional behavior. The EAI index of a node *switch* can be expressed as:

$$EAI \in \mathbb{Q}^+ \quad | \quad EAI_{switch} = \sum_{j=1}^{m} p_j \cdot EAI_{a_j}$$

where m is the number of child nodes, $a_j | j = 1, ..., m$ denotes the j-th activity in a conditional branch of the switch construct, and p_j is the probability of executing the activity a_j. In our view, adaptability of interaction within a switch construct is related to the probability of the occurrence of each of its conditions. Thus, the EAI of a switch construct is calculated as the summation of the probability of each condition occurrence multiplied with the EAI of the interaction activity within that condition. At design time, we assume that the probability of execution for branches is equivalent: $p_j = \frac{1}{m}$. It must hold: $p_j \geq 0$ for all $j = 1 \ldots m$ and $\sum_{j=1}^{m} p_j = 1$. At runtime, the probability of execution for every single conditional branch may differ from the other branches. These probabilities can be estimated from the *operational profile*[5] [10].
A node *if-else* is considered equivalent to a node *switch* with two conditional branches. Referring to the example in Fig. 1, we observe at design time:

$$EAI_{if-else} = 0.5 \cdot EAI_{reply} + 0.5 \cdot EAI_{invoke\ s4} = 0.5 \cdot 0 + 0.5 \cdot 1 = 0.5$$

Node *pick***.** The construct *pick* is used to wait for the occurrence of one of a set of events (message events or alarm events) and then perform an activity associated with the event. The semantics of a *pick* construct is similar to that of a *switch*. The EAI index of a node *pick* is therefore:

$$EAI \in \mathbb{Q}^+ \quad | \quad EAI_{pick} = \sum_{j=1}^{m} p_j \cdot EAI_{a_j}$$

[5] Environmental data about the business process collected from domain experts.

where m is the number of child nodes, $a_j|j = 1, ..., m$ denotes the j-th activity associated with an event e_j (a message event or an alarm event), and p_j is the probability that event e_j occurs. Also in this case, at design time, we assume $p_j = \frac{1}{m}$ as for a *switch* construct.

Node *while*. The EAI index of a node *while* is:

$$EAI \in \mathbb{Q}^+ \quad | \quad EAI_{while} = \frac{N \cdot \Sigma_{j=1}^{m} EAI_{a_j}}{m}$$

where m is the number of child nodes, $a_j|j = 1, ..., m$ denotes the j-th child activity, and N is the number of loop iterations. At design time, we are not able to calculate N exactly, however, it can be estimated with the aid of an operational profile.

Node *sequence*. The *sequence* construct is used to define activities that need to be performed in a sequential order. The EAI index of a node *sequence* is:

$$EAI \in \mathbb{Q}^+ \quad | \quad EAI_{sequence} = \frac{\Sigma_{j=1}^{m} EAI_{a_j}}{m}$$

where m is the number of child nodes, $a_j|j = 1, ..., m$ denotes the j-th child activity executed sequentially within the sequence node.
Referring to the example in Fig. 1:

$$EAI_{sequence} = \frac{EAI_{receive} \cdot EAI_{flow} \cdot EAI_{invoke\ s3} \cdot EAI_{if-else}}{4} = \frac{0+1.67+2+0.5}{4} = 1.04.$$

Note that the sequence node in Fig. 1 is also the root of T_p. Therefore, it results: $EAI_{root(T_p)} = EAI_{sequence} = 1.04$. The EAI of the root of T_{map} is calculated in the same way by considering for each service s_i all available service instances $|C_i|$. Referring to Fig. 1, $EAI_{root(T_{map})} = 1,33$, and therefore:

$$PAI_p = \frac{EAI_{root(T_p)}}{EAI_{root(T_{map})}} = \frac{1.04}{1.33} = 0.78$$

3 Relating adaptability with a quality of service attribute

Software quality attributes can rarely be achieved in isolation. Most often, the achievement of a quality attribute has an effect, positive or negative, on the achievement of others [1]. Process adaptability is not an exception, and it can influence quality attributes such as performance, reliability or maintainability. Therefore, an increment in the process adaptability can cause an improvement in some of them, but also a damage.

For example, given a performance response time requirement, it may happen that more adaptability produces better performance, since the expected fastest service instance can be chosen at each invocation moment. However, it can also happen that all service instances are always fast, and then the time necessary

to compute decision of which service instance to use creates a delay that results in a lower performance than a non-adaptive system.

The same happens for the probability of an execution of the business process to succeed. Although at first sight it may seem that having more alternative service instances to execute a concrete service will always improve its probability to succeed, we can also argue that the implementation of the required adaptation manager adds a complexity in the software, then it creates an additional point of failure and can damage the overall process quality.

From these examples, as in [11], we cannot assume that a certain quality attribute is always in the same type of relationship with the adaptability, hence it is needed a system analysis to identify their type of relation.

Quality computation: In this work we focus on the quality attribute of "probability of a process execution to succeed its execution", called *Qual* in the following. For computing *Qual* value in a given process, we use basic formula of availability evaluation.

We assume that leaf nodes in the process activity tree of type *assignment, receive, reply* never fail their execution, then their *Qual* is 1. We assume as known the *Qual* value of service instances Cij, called $Qual_{Cij}$.

Then, the *Qual* of a leaf node *invoke* is the probability of any of the service instances that receive such invocation to succeed an execution, whose formula is: $Qual_{invoke\ s_i} = 1 - \prod_{j=1}^{j=EAI_{invoke\ s_i}} (1 - Qual_{Cij})$

The quality of nodes *switch*, *pick*, is calculated as:

$$Qual_n = \sum_{j=1}^{m} p_j \cdot Qual_j \quad \forall n \in \{switch, pick\}$$

while, the quality of nodes *flow*, *sequence* and *while* is calculated as:

$$Qual_n = \prod_{j=1}^{m} Qual_j^{N_j} \quad \forall n \in \{flow, sequence, while\}$$

where m is the number of child nodes and $a_j | j = 1, ..., m$ denotes the j-th child activity within the scope of the node; p_j is the probability of executing child node j; and N_j is the number of times that child node j is executed in iteration (it is straightforward to see that this value is higher than 1 for nodes of type *while* and equal to 1 for the rest of types).

Therefore, system *Qual* is recursively computed in the process activity tree and it is equivalent to the quality value calculated for its root node $Qual_{root}$.

4 A case study: the University Student Enrollment

This section presents a more realistic example to illustrate the proposed metrics. The example is a web service application used by students to register for an

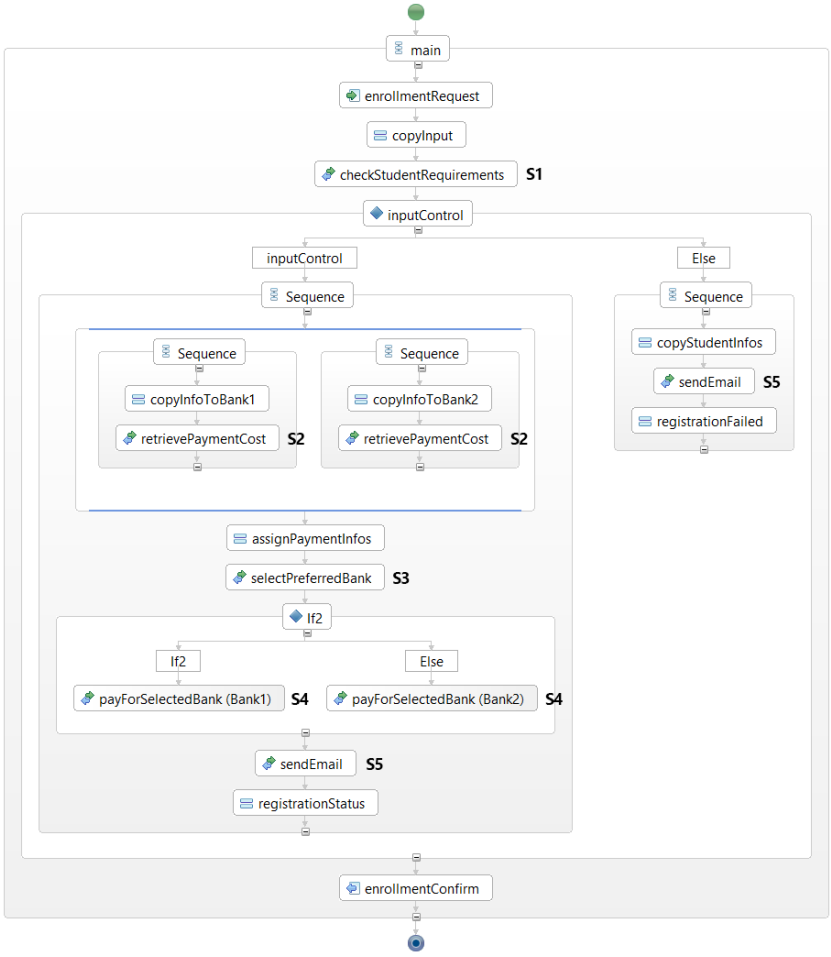

Fig. 2. Student enrollment BPEL process

academic year in the University. Figure 2 sketches the BPEL process for student enrollment as modeled through the Eclipse BPEL Designer plug-in[6].

At first, a student registers and introduces his/her proposal (a list of courses to take) in the web system. Then, the proposal is sent to a web service (an application logic layer) to check if it fulfills the University rules. In case of reject, the registration process interacts with a mail web service to send an email to the student with information about the registration failure. If, instead, the proposal fulfills the University rules, the process interacts with two bank web services (known at design time) to proceed with the payment. To this purpose, the process

[6] Eclipse BPEL Designer Plug-in: `http://www.eclipse.org/bpel/`

asynchronously calls back the student with the bank transaction costs, thus allowing the student to choose the bank service with the lowest costs. Once the bank has been selected by the student, and the bank payment has been predisposed, the process invokes a web mail service (not known at design time) to send an email to the student with the information of the successful registration. For both proposals approve or reject cases, a presentation layer with a web-GUI and mechanisms to interact with the student sends a message back to student with information about the registration outcome.

For the student enrollment BPEL process, Table 1 relates the abstract services (ID and description) invoked by the process with the corresponding concrete service instances (ID and description) available as internal or external service components. Figure 3 shows a possible process activity tree for the student enrollment process. This process configuration has been obtained by selecting some concrete service instances. The activity tree of the corresponding most adaptable process for the student enrollment example is similar and takes into account all the service instances reported in Table 1. Note that labels n_k in Fig. 3 enumerate the internal nodes.

Service	**Concrete service**
s_1 Check student requirements	C11 Application Logic 1 C12 Application Logic 2
s_2 Retrieve payment cost	C21 Mobile cost provider C22 Email cost provider
s_3 Select preferred bank	C31 Bank selection (callback handler)
s_4 Pay for selected bank	C41 PayPal payment provider C42 NFC payment provider C43 Mobile payment provider C44 Credit card payment provider
s_5 Send email	C51 Local email provider C52 Email provider 1 C53 Email provider 2

Table 1. Abstract and concrete services for the student enrollment process

Table 2 shows the value of the metrics (the EAI values for the internal nodes and the final PAI) for the process configuration in Fig. 3. The given process configuration exhibits a good adaptability since it differs from the most adaptable one by 7%.

Adaptability and Quality measures at work. At present, the student enrollment process invokes service instance $C11$ for $s1$. With the information returned it has been observed that only 2.5% of students do not satisfy the University requirements and an email rejection is sent by $C51$. The interaction with $s2$ of the two different banking web services is done by invoking service instance $C21$. The interaction with the bank payment process is done by the invocation

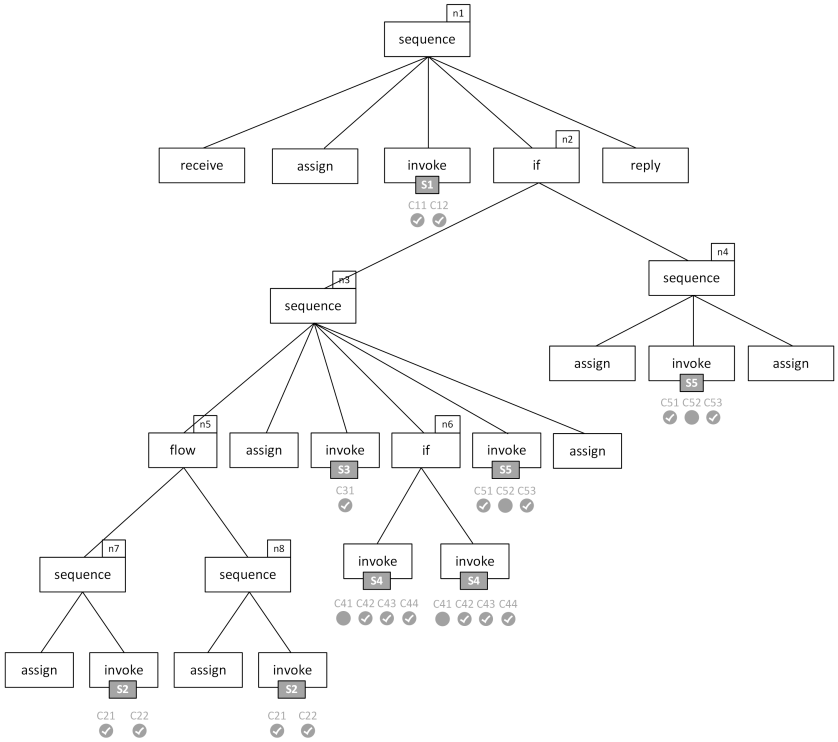

Fig. 3. A process activity tree for the student enrollment process

Activity tree T_p	**Activity tree** T_{map}
$EAI_{n_7} = \frac{0+2}{2} = 1$	$EAI_{n_7} = \frac{0+2}{2} = 1$
$EAI_{n_8} = \frac{0+2}{2} = 1$	$EAI_{n_8} = \frac{0+2}{2} = 1$
$EAI_{n_5} = \frac{1+1}{2} = 1$	$EAI_{n_5} = \frac{1+1}{2} = 1$
$EAI_{n_6} = 0.5 \cdot 3 + 0.5 \cdot 3 = 3$	$EAI_{n_6} = 0.5 \cdot 4 + 0.5 \cdot 4 = 4$
$EAI_{n_3} = \frac{1+0+1+3+2+0}{6} = 1.17$	$EAI_{n_3} = \frac{1+0+1+4+3+0}{6} = 1.5$
$EAI_{n_4} = \frac{0+2+0}{3} = 0.67$	$EAI_{n_4} = \frac{0+3+0}{3} = 1$
$EAI_{n_2} = 0.5 \cdot 1.17 + 0.5 \cdot 0.67 = 0.92$	$EAI_{n_2} = 0.5 \cdot 1.5 + 0.5 \cdot 1 = 1.25$
$EAI_{n_1} = \frac{0+0+2+0.92+0}{5} = 0.58$	$EAI_{n_1} = \frac{0+0+2+1.25+0}{5} = 0.65$
$PAI_p = \frac{EAI_{root(T_p)}}{EAI_{root(T_{map})}} = \frac{0.58}{0.65} = 0.89$	

Table 2. The metrics values for the given process configuration

Instance	prob. fail	Instance	prob. fail	Instance	prob. fail	Instance	prob. fail
$C11$	0.975	$C21$	0.95	$C31$	0.999	$C41$	0.9
$C42$	0.8	$C43$	0.7	$C44$	0.99	$C51$	0.98
$C12$	0.99	$C22$	0.98	$C52$	0.98	$C53$	0.99

Table 3. Quality of service instance

Service instances	Qual	EAI	PAI	Adjusted Qual $Qual \cdot f(EAI, 0.01)$
Initial process	0.8637	1.0303	0.7066	0.8635
ALL ten service instances	0.9967	1.4582	1	0.9921
all instances but $C12$	0.9721	1.2582	0.8628	0.9695
all instances but $C52$	0.9965	1.4443	0.9904	0.9921
all instances but $C53$	0.9963	1.4443	0.9904	0.9919
all instances but $C52$ and $C53$	0.9768	1.4303	0.9808	0.9726
all instances but $C22$	0.9037	1.2442	0.8532	0.9015

Table 4. Processes adaptability and expected quality values

of one of the service instances $C41$, $C42$, $C43$ or $C44$. The final interaction with email service is carried out again by $C51$. The observed quality of these service instances in terms of the probability of executing a request to their offered service without errors is shown in the higher part of Table 3. We assume that the rest of nodes in the process activity tree that are not of *invoke* type can never fail. Therefore, the calculated quality of this process in terms of the probability of executing a complete request for enrollment without errors ($Qual$) is computed according to formulas presented in Section 3 and is 0.86377. When a request for enrollment fails its execution, the student has to go to the secretariat to personally request his/her enrollment.

The University wants to improve the application, since it has 30,000 students but the secretary service can manually manage 200 of them without saturating. Then, it is required a $Qual \geq 29700/30000 = 0.99$. The IT service has considered the local deployment of a new service instance (called $C12$) that has been recently developed and offers an alternative for the execution of the application logic. For $s2$, it would be possible to use a service instance $C22$. Regarding the email service, the University considers the utilization of two other relays (called $C52$ and $C53$) to use in the moment when the local relay is not working properly (e.g., unreachable or saturated being rejecting connections). The quality of these existing service instances is shown in the lower part of Table 3.

Therefore, the current business process can use service instances $C11$, $C21$, $C31$, $C41$, $C42$, $C43$, $C44$ and $C51$; while the most adaptable process would be able to use also services $C12$, $C22$, $C52$ and $C53$. The implementation of

an adaptive service-oriented application through composition of heterogeneous services – even if they provide the same functionality – will require some programming effort from the IT department to make interoperable their interfaces; i.e., the service invocation is not completely seamless in this case. For this reason, the IT department would like to know the business process that provides enough *Qual* and uses the lowest level of adaptation. For calculating *Qual* we use the formulae in Section 3 while the proposed *EAI* metrics are used for calculating the quantity of adaptability of each candidate business process. Using *EAI* metric and an estimation of the bug inclusion rate when implementing autonomic manager of adaptive processes, we can estimate the expected quality of the resulting adaptive process. The more adaptive a node in the BPEL process is, the higher its likelihood to include a bug created during its implementation. The estimated bug inclusion rate is 0.01, meaning that for each service instance that adds adaptability, the probability of failure during instance decision process increases by 1%. We do not go in detail calculation of the success rate of the autonomic manager execution and we call it $f(EAI, 0.01)$, just assuming that it is a non increasing function for EAI. By calculating the *PAI* of each candidate process, we can give an evaluation of the mean adaptability of each element in the process with respect to the most adaptable one and, in consequence, an insight of the relative implementation effort that the IT department saves by not deciding directly in favor of the most adaptable business process.

Table 4 shows the results of the metrics *EAI*, *PAI*, *Qual* and *Adjusted Qual* for some of the evaluated processes. Among the processes that satisfy the execution success requirement, the one composed by all service instances but $C53$ is the one that showed highest *Ajusted Qual* lowest EAI value.

5 Implementation and tool support

To automatically calculate the proposed metrics, we adopted SOLAR [14] (SOftware quaLities and Adaptability Relationships), a tool developed in [11] for adaptability quantification of a software architecture. We extended SOLAR to include the new metrics defined at the business process level and validate it on the case study presented previously. The implementation units of this SOLAR extension, called B-SOLAR (Business SOLAR) are shown in Fig. 4.

First (*phase 1* in Fig. 4), the user must provide manually an input file `settings.xml` containing the names of two XML input files used by SOLAR. One input XML file is the conventional input read by SOLAR through the software module `Components and services Parser` (see *phase 2* in Fig. 4). This input file contains the description of the software architecture comprising components and connectors. This same input is also used by B-SOLAR (see *phase 3* in Fig. 4) to determine the relation between abstract services and concrete service instances. This information is stored in an hash table for a faster access during the metric computation.

The second input file is read by B-SOLAR trough the `Business Process Parser` (see *phase 4* in Fig. 4). This file is the BPEL XML file containing (among

other things) the description of the considered business process. From this file a process activity tree is created (phase 5 in Fig. 4). Through a combinatorial algorithm, the various combinations that can be taken of the service concrete instances of a particular service type are generated (*phase 6* in Fig. 4). Each combination generated is then associated with the process activity tree and the metrics PAI and EAI are then computed according to the approach presented in Sect. 2 (*phase 7* in Fig. 4). The results of such evaluation are finally saved (*phase 8* in Fig. 4) in an output XML file through the module `Output writer`.

In order to enable the B-SOLAR tool in a cloud context, we added to B-SOLAR a further parser (*phase 2'* in Fig. 4) that is able to accept as input a TOSCA-based[7] description of a cloud-based software application (the SaaS layer only) and transforms it into the internal C&C view description used by SOLAR for representing software architectures. For the dynamic aspects, namely the TOSCA plans defined as process models, TOSCA specification relies on existing languages like BPMN or BPEL. We assume, therefore, TOSCA plans are provided using BPEL as supported by B-SOLAR.

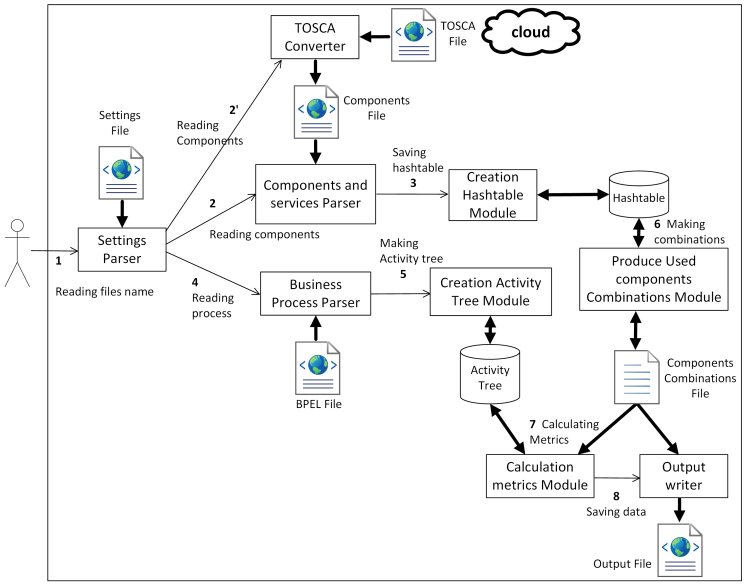

Fig. 4. B-SOLAR tool architecture

[7] The OASIS Topology and Orchestration Specification for Cloud Applications. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html`

6 Related work

In the following we briefly review papers dealing with metrics and approaches for adaptability and evolution of business processes. An extensive related work about metrics for system adaptability applicable at architectural level can be found in [11].

In software engineering, many approaches have been proposed to support the adaptability and evolution of business processes (such as [12], [7], to name a few). In these approaches process adaptability is conceived differently from the vision we take in this paper. Indeed, we quantify the variability degree of BPEL processes to adapt their bindings to concrete service instances during their life cycles. Instead, in the works mentioned above, process adaptability is intended in the broadest sense of self-adaptive systems [4,6,13]. They assume that business process specification languages are extended with special constructs and self-adaptation plug-ins (e.g., for monitoring or diagnosis so that the plug-in can decide if the adaptation is needed though a feedback adaptation control loop) that allow the business process specification to change at runtime without the need to redeploy it and lose the ongoing transactions.

In [8], the authors propose an approach for quantifying the degree of structural adaptability of BPMN business processes using software metrics. They aim at quantifying how easily a process can be adapted to a different form by replacing combinations of BPMN constructs with other ones having the same runtime semantics. Such a degree can be helpful for quality assessment during development or decision support during migration. In such a work, yet process adaptability is conceived differently from our vision. The author considers adaptability as sub-characteristic of the portability quality, i.e. the degree to which a process can be adapted in order to be executed in a different execution platform.

7 Conclusions and future work

BPEL processes are workflow-oriented service compositions for creating service-oriented applications. Rapidly changing environmental and market conditions require flexible BPEL processes that adapt their bindings to concrete services during their life cycles.

In this paper, we have extended the set of metrics presented in [11] that quantify the software adaptability at architectural level with metrics that quantify the software adaptability at the business process level. These metrics allow us to quantitatively evaluate and compare different service-oriented applications in terms of architectural and behavioral adaptability and quality requirements. The approach can help software architects to find architectures and business processes satisfying all system quality requirements. The software architect may applies the approach when changes in the execution context force to change the service instances of the business process for satisfying quality requirements. To automate the analysis we have extended the SOLAR tool.

At present we are working on testing the B-SOLAR tool by designing several experiments from with real-size service-oriented applications. In particular, we

are looking for a test-bed in a cloud computing scenario. We are also working to overcome some limitations. One limitation is that it is not possible to list and program all variability of service instances at design time. The generation of all possible process activity trees (or process configurations) for an input business process should, instead, carried out at run-time through a web service discovery mechanism.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Addison-Wesley, 2005.
2. R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Trans. Software Eng.*, 37(3):387–409, 2011.
3. V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the the 7th joint meeting ESEC/FSE*, pages 131–140, New York, NY, USA, 2009. ACM.
4. B. H. Cheng and al. Software engineering for self-adaptive systems: A research roadmap. 5525:1–26, 2009.
5. R. Cognini, F. Corradini, S. Gnesi, A. Polini, and B. Re. Research challenges in business process adaptability. In *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pages 1049–1054, 2014.
6. R. de Lemos and al. Software Engineering for Self-Adaptive Systems: A second Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 7475 of *LNCS*, pages 1–32. Springer, 2013.
7. G. Hermosillo, L. Seinturier, and L. Duchien. Using complex event processing for dynamic business process adaptation. In *IEEE Int.Conf. on Services Computing, SCC 2010*, pages 466–473, 2010.
8. J. Lenhard. Towards quantifying the adaptability of executable BPMN processes. In *Proceedings of the 6th Central-European Workshop on Services and their Composition, ZEUS 2014.*, pages 34–41, 2014.
9. R. Mirandola, P. Potena, and P. Scandurra. Adaptation space exploration for service-oriented applications. *Science of Computer Programming*, 80:356–384, 2014.
10. J. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, Mar 1993.
11. D. Perez-Palacin, R. Mirandola, and J. Merseguer. On the relationships between QoS and software adaptability at the architectural level. *Journal of Systems and Software*, 87:1–17, 2014.
12. L. Sabatucci, C. Lodato, S. Lopes, and M. Cossentino. Towards self-adaptation and evolution in business process. In *Proceedings of the Workshop AI Meets Business Processes 2013*, pages 1–10, 2013.
13. M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.
14. SOLAR. `http://webdiis.unizar.es/GISED/?q=tool/solar`. Universidad de Zaragoza, 2011.