10

Learning Optimal Web Service Selections in Dynamic Environments when Many Quality-of-Service Criteria Matter

Stéphane Dehousse[1], Stéphane Faulkner[2], Caroline Herssens[3], Ivan J. Jureta[4] and Marcos Saerens[5]
[1,2,4]PReCISE Research Center (University of Namur), Namur,
[3]PReCISE Research Center(University of Louvain), Louvain-la-Neuve,
[5]ISYS Research Unit (University of Louvain), Louvain-la-Neuve,
Belgium

1. Introduction

The emergence of the World Wide Web has lead to growing needs for interacting components capable of achieving – together – complex requests on the Web. Service oriented Systems (SoS) are a response to this issue, given available standards for describing individual services and interaction between them, and the attention to interoperability combined with an uptake in industry. A *service* is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network (McIlraith & Martin, 2003; Papazoglou & Georgakopoulos, 2003). A *web service* is a service made available on the Internet via tailored technologies such as WSDL, SOAP or UDDI (Walsh, 2002). To fulfill elaborate requests that involve many execution steps, web services participate in *web services compositions*. To optimize such compositions, each step of the execution is achieved by the most competitive available web service. The most competitive web service is the one who performs the given task while fulfilling its functional requirements and providing the best observed values of quality of service (QoS).

QoS are the nonfunctional properties of a web service and refer to concerns such as availability, reliability, cost or security (Menascé, 2002). The selection of all web services that can participate in a composition (i.e., web services that will perform at least one step of the execution) is under the responsibility of the *service composer*. To achieve QoS-aware service selection, we rely on a Multi-Criteria Randomized Reinforcement Learning approach (MCRRL). MCRRL authorizes automated continuous optimization of service monitoring and leads the system to respond to the variation of the availability of web services without human involvement.

This paper focuses on the composition of services under the constraint of openness, resource distribution, and adaptability to changing web service availability w.r.t. multiple criteria. To enable such system characteristics, a fit between the system architecture and services composition behavior is needed, that is: (1) To support openness, few assumptions can be

Source: Machine Learning, Book edited by: Abdelhamid Mellouk and Abdennacer Chebira,
ISBN 978-3-902613-56-1, pp. 450, February 2009, I-Tech, Vienna, Austria

made about the behavior of the web services that may participate in compositions. It thus seems reasonable to expect services composition responsibility not to be placed on any web service: the architecture ought to integrate a special set of web services, the service composers, that coordinate services composition. (2) To allow the distribution of web services, no explicit constraints should be placed on the origin of entering services. (3) To enable adaptability, composer behavior should be specified along with the architecture. (4) Since there is no guarantee that web services will execute tasks at performance levels advertised by the providers, composition should be grounded in empirically observed service performance and such observations executed by the composers. (5) The variety of stakeholder expectations requires services composition to be driven by multiple criteria. (6) To ensure continuous adaptability at runtime, composition within the architecture should involve continual observation of service performance, the use of available information to account for service behavior and exploration of new options to avoid excessive reliance on historical information.

Contributions. We provide a complete composition process involving several steps: (1) The service user requests a service that involves several tasks that can be fulfilled by different web services. These tasks and all possible execution paths are described on on a statechart. (2) The service composer observes available web services and rejects those that can not achieve one of the existing task of the statechart. It then builds the resulting execution plan as a Directed Acyclic Hypergraph, on which it represents all services available for each task. (3) The service requester expresses its quality expectations with the help of our QoS model. (4) The composer rejects services that do no meet quality requirements and scores each candidate web service with our proposed QoS aggregation model to get a multi criteria measure of their performance. (5) This value is the one that the service composer maximizes in our RRL algorithm. The result of the computation gives us web services to select to get the most competitive composite web service.

Organization. We present our conceptual foundations for the remaining of the paper in Section 2. That section covers the case study used throughout this paper and our composition model with its statechart representation and its Directed Acyclic Hypergraph derivation. It also proposes our QoS model applied by the service user to specify its priorities and preferences about QoS. Section 3 presents how multiple quality criteria are aggregated into a single measure of performance. The method is illustrated with the previously introduced case study. Section 4 introduces our Reinforcement Learning solution to the composition problem by liken it to the task allocation problem. Section 5 presents experiments that we made on our Multi-Criteria Randomized Reinforcement Learning proposal. Section 6 outlines the related work. Finally Section 7 concludes this paper and exposes our future work.

2. Baseline

This section presents the different conceptual elements used through the paper. Our case study is introduced in Subsection 2.1. Our services composition model is proposed in Subsection 2.2 and involves two steps. The first is to define the possible composition process with a statechart as described in Subsection 2.2.1. We illustrate the statechart representation with the composition of web services introduced in the case study. The second is to represent candidate services for each elementary task of the whole composition. This representation is derived from statecharts with Directed Acyclic Hypergraph. The

procedure for doing so is explained and illustrated with our case study in Subsection 2.2.2. We present in Subsection 2.3 the QoS model dedicated to the service user to make its particular requirements about its QoS priorities and preferences.

2.1 Case study

To illustrate our selection of services entering in a composition, we propose a case study subsequently used throughout the paper. The European Space Agency's (ESA) program on Earth observation allows researchers to access and use infrastructure operated and data collected by the agency.[1] Our case study focuses on the information provided by the MERIS instrument on the Envisat ESA satellite. MERIS is a programmable, medium-spectral resolution imaging spectrometer operating in the solar reflective spectral range. MERIS is used in observing ocean color and biology, vegetation and atmosphere and in particular clouds and precipitation. In relation to MERIS, web services are made available by the ESA for access to the data the instrument sends and access and use of the associated computing resources.

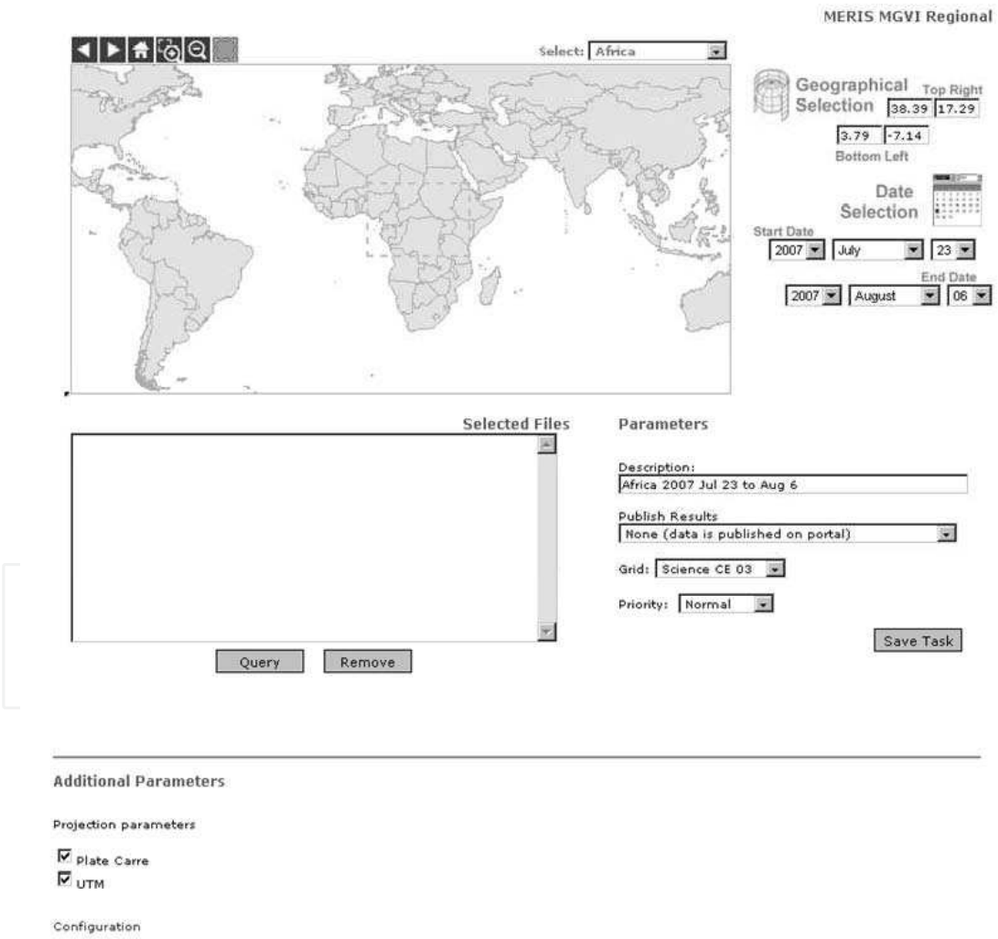


Fig. 1. Graphical user interface of the ENVISAT/MERIS MGVI web service

[1]http://gpod.eo.esa.int

Among available functionalities delivered by these web services, we focus our attention to services enabling to extract the vegetation index, or more precisely, the Fraction of Absorbed Photosynthetically Active Radiation (FAPAR) from MERIS data. The graphical interface used to determine the requested information for the vegetation index on a given region is illustrated in Figure 1. The graphical output produced for the world-wide map is given in Figure 2.

Two main services allow the extraction of such data, one processing the information for the world-wide map and the other computing information for a given area of the world. The graphical user interface of the service providing regional data differs from the world wide one with a bounding box enabling to select an area on the map. World-wide data is much often requested than data for a given region of the world, so World-wide data can be more rapidly retrieved than the regional. Moreover, some services authorize the extraction of regional data from world-wide data.

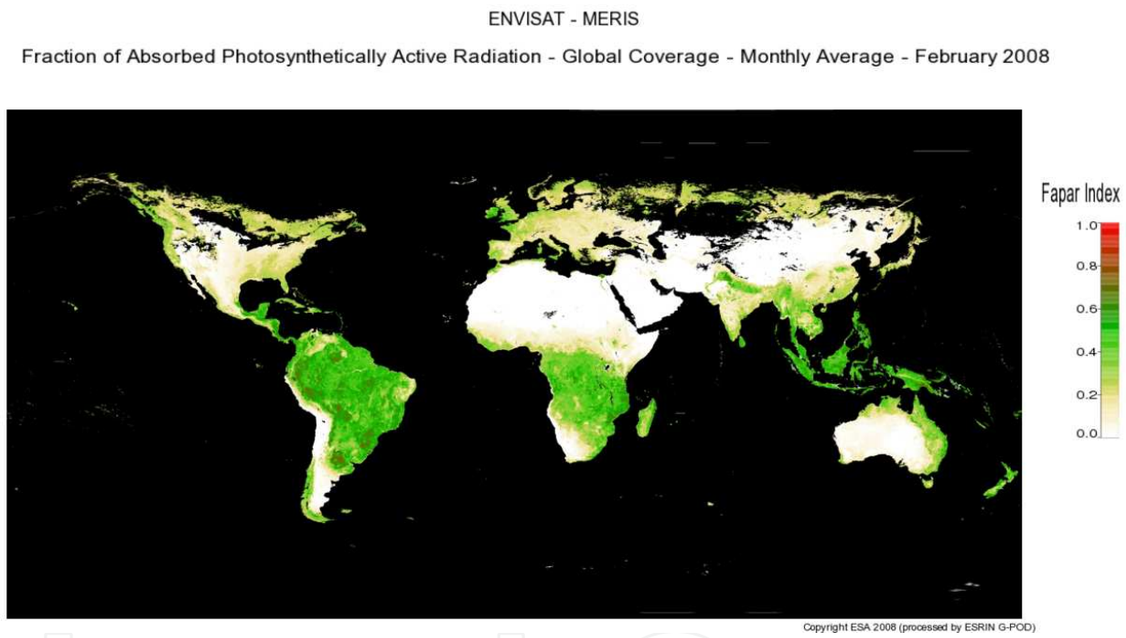


Fig. 2. Output provided by the world-wide vegetation service

2.2 Web services composition model

Service requests pointed out that various criteria can be used in specifying a service request; namely, QoS concepts cover deadline, reputation, monetary cost, and explicit requester preferences. Reputation and trust receive considerable attention in the literature (e.g., (Maximilien & Singh, 2005; Zacharia & Maes, 2000)). In AOSS, the ideas underlying Maximilien and Singh's approach (Maximilien & Singh, 2005) can be followed, with two caveats: they use "trust" to select services from a pool of competing services and exploit user-generated opinions to calculate reputation, whereas herein WS are selected automatically and reputation can be generated by comparing WS behavior observed by the composer and the advertised behavior of the WS. The following is one way to define reputation in AOSS.[2]

[2]Reputation is used here instead of trust since no user opinions are accounted for.

Definition 1. Reputation R_k^{a,w_i} of a WS w_i over the QoS aggregated score k is:

$$R_k^{a,w_i} = \frac{1}{n-1}\sum_{i=1}^{n}\left[\left(v_k^{Adv} - \hat{v}_k^i\right)^2 \delta^{-time(\hat{v}_k^i)}\right]$$

where $time()$ returns the time of observation (a natural, 1 for the most recent observed value, $time(\hat{v}_k^i) > 1$ for all other) and δ is the dampening factor for the given quality (can be used with $time()$ to give less weight to older observations). We assume that the advertised quality for w_i is $\langle (p_1, d_1, v_1^{Adv}, u_1), \ldots, (p_r, d_r, v_r^{Adv}, u_r)\rangle$, and that n observations $\hat{v}_k^i, 1 \le i \le n$ have been made over a quality parameter k.

It is apparent that many other criteria can be accounted for when selecting among alternative WS compositions. Decision making in presence of multiple criteria does not require full specification of all possible criteria for each WS---instead, it is up to the requester to choose what criteria to specify. The algorithm thus optimizes a single normalized variable (i.e., taking values in the interval $[0,1]$). An aggregation function for the criteria relevant to the service requester is applied, so that the result of the function is what the algorithm will optimize. The process providing the aggregation function is presented in Section 3.

2.2.1 Statechart representation

A services composition is a succession of elementary tasks, whose exution fulfills a complex request. We assume the request describes a process to execute. Individual web services are combined together according to their functional specifications. Compositions support alternative possibilities and concurrency of elementary tasks. Similarly to Zeng and colleagues, our service process is defined as a statechart (Zeng et al., 2003). Statecharts offer well defined syntax and semantics so that rigorous analysis can be performed with formal tools to check specification concordance between services. Another advantage is that they incorporate flow constructs established in process modeling languages (i.e, sequence, concurrency, conditional branching, structured loops, and inter-thread synchronization). Consequently, standardized process modeling languages, such as, e.g., BPMN (OMG, 2006a), can be used to specify the process model when selecting services that will enter in the composition. Statecharts offer the possibility to model alternatives and a composite task can be achieved by different paths in the statechart. Such paths are named *execution paths* ant their definition in relation to statecharts is given in Definition 2.2.1. The statechart is a useful representation of a process that a WS composition needs to execute, most selection algorithms cannot process a statechart in its usual form. Instead, a statechart is mapped onto a Directed Acyclic Hypergraph (DAH), using Definition 2.2.1 and the technique for constructing DAH, described below.

(Adapted from (Zeng et al., 2003)) An execution path of a statechart is a sequence of states $[t_1, t_2, \ldots, t_n]$, such that t_1 is the initial state, t_n the final state, and for every state $t_i (1 < i < n)$, the following holds:

- t_i is a direct successor of one of the states in $[t_1, \ldots, t_{i-1}]$.
- t_i is not a direct successor of any of the states in $[t_{i+1}, \ldots, t_n]$.

- There is no state t_j in $[t_1,\ldots,t_{i-1}]$ such that t_j and t_i belong to two alternative branches of the statechart.

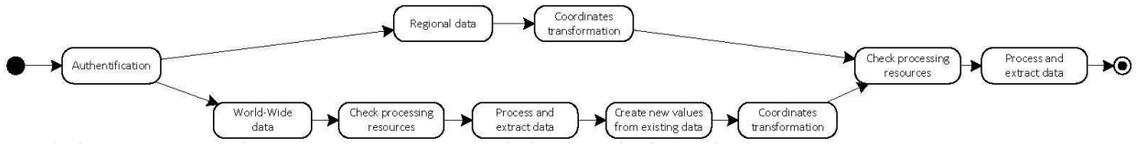


Fig. 3. Statechart representation of the composite service

We concentrate our efforts here on the description of elementary tasks of a composite service processing the FAPAR for a given region of the world. Two main paths of tasks allow to achieve this composite service. Besides elementary tasks stepping in both paths, the first path uses services allowing to process data for a given region of the world while the second path process the world-wide data and restrains the information to the given area. The composite service and its elementary tasks are illustrated with the corresponding statechart in Figure 2.2.1.

2.2.2 Directed acyclic hypergraph instantiation

It is apparent that an acyclic statechart has a finite number of execution paths. If the statechart is not acyclic, it must be "unfolded" (Zeng et al., 2003): logs of past executions need to be examined in order to determine the average number of times that each cycle is taken. The states between the start and end of a cycle are then duplicated as many times as the cycle is taken on average. Assuming for simplicity here that the statechart is acyclic, an execution path can be represented as a Directed Acyclic Hypergraph.

Given a set of distinct execution paths $\{[t_{1,k},\ldots,t_{n,k}]\}$ (k is the index for execution paths), the Directed Acyclic Hypergraph (DAH) is obtained as follows:

- DAH has an edge for every pair $(task,WS)$ which indicates the allocation of WS to the given task. DAH thus has as many edges as there are possible allocations of WS to tasks.
- DAH has a node for every state of the task allocation problem. Such a state exists between any two sequentially ordered tasks of the task allocation problem (i.e., a node connecting two sets of edges in the DAH, whereby the two tasks associated to the two sets of edges are to be executed in a sequence).

Note that: (i) the DAH shows all alternative allocations and all alternative execution paths for a given statechart; (ii) conditional branchings in a statechart are represented with multiple execution paths.

Available web services for fulfilling individual tasks of our composite service proposed in Figure 2.2.1 need to be represented in a DAH to apply our selection approach. Each state of the statechart will become a node in the DAH with an additional starting node depicting the initial state. The resulting DAH is available in Figure 3 with each edge standing for a service able to fulfill the task specified in the outgoing node of the edge. Several services provided by the ESA are able to fulfill each individual tasks of the composite service providing the FAPAR index. The DAH representation gather web services which can be used at different steps of the execution.

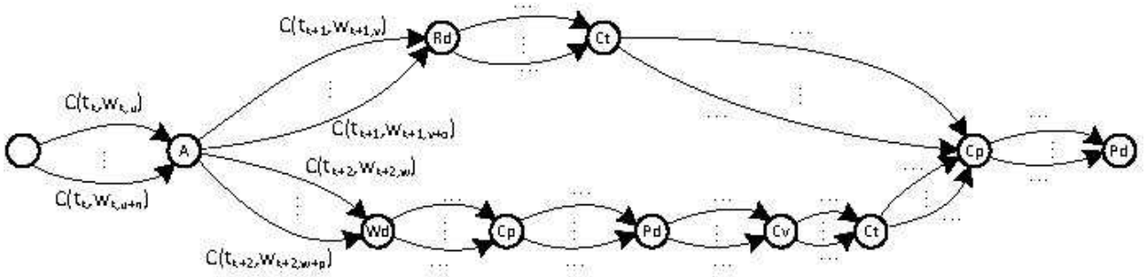


Fig. 4. DAH representation of the composite service

2.3 Specification of user priorities and preferences

We suggest a QoS model that enables the user to express accurately its needs about quality properties of its required service. To account for various aspects of user expectations, this model must include advanced concepts such as priorities over quality characteristics or preferences on offered values. To enable specifying these concepts, the model contains modeling constructs dedicated to various facets of user expectations.

Among multiple available QoS models (D'Ambrogio, 2006; Keller & Ludwig, 2003; Zhou et al., 2004), we base our model on the UML QoS Profile. The original UML QoS Framework metamodel, introduced by the Object Management Group (OMG, 2006b), includes modeling constructs for the description of QoS considerations. It has some advantages over other models: it is based on the Unified Modeling Language (UML); it is a standard provided by the Object Management Group (OMG); it is a metamodel that can be instantiated in respect to users needs; and it covers numerous modeling constructs and allows to add some extensions. This model with our extensions are shown in Figure 2.3.

In that metamodel, a *QoS Characteristic* is a description for some quality consideration, such as e.g., latency, availability, reliability or capability. Extensions and specializations of such elements are available with the sub-parent self-relation. A characteristic has the ability to be derived into various other characteristics as suggested by the templates-derivations self-relation. A *QoS Dimension* specifies a measure that quantifies a QoS Characteristic. The *unit* attribute specifies the unit for the value dimension. *QoS Values* are instantiations of QoS Dimensions that define specific values for dimensions depending on the value definitions given in QoS DimensionSlots. A *QoS DimensionSlot* represents the value of QoSValue. It can be either a primitive QoS Dimension or a referenced value of another QoSValue. While constraints usually combine functional and non-functional considerations about the system, *QoS Context* is used to describe the context in which quality expression are involved. A context includes several QoS Characteristics and model elements. The aim of *QoS Constraints* is to restrict values of QoS Characteristics. Constraints describe limitations on characteristics of modeling elements identified by application requirements and architectural decisions.

In comparison with the original OMG metamodel, we make some additional assumptions:

- In the OMG standard, *QoS Characteristics* are quantified by means of one or several *QoS Dimensions*. We assume that the value of a QoS Dimension can similarly be calculated with quantitative measures of other QoS Dimensions. This assumption is expressed in the metamodel in Figure 2.3 through the *Compose-Composed by* relationship of the *QoS Dimension* metaclass.

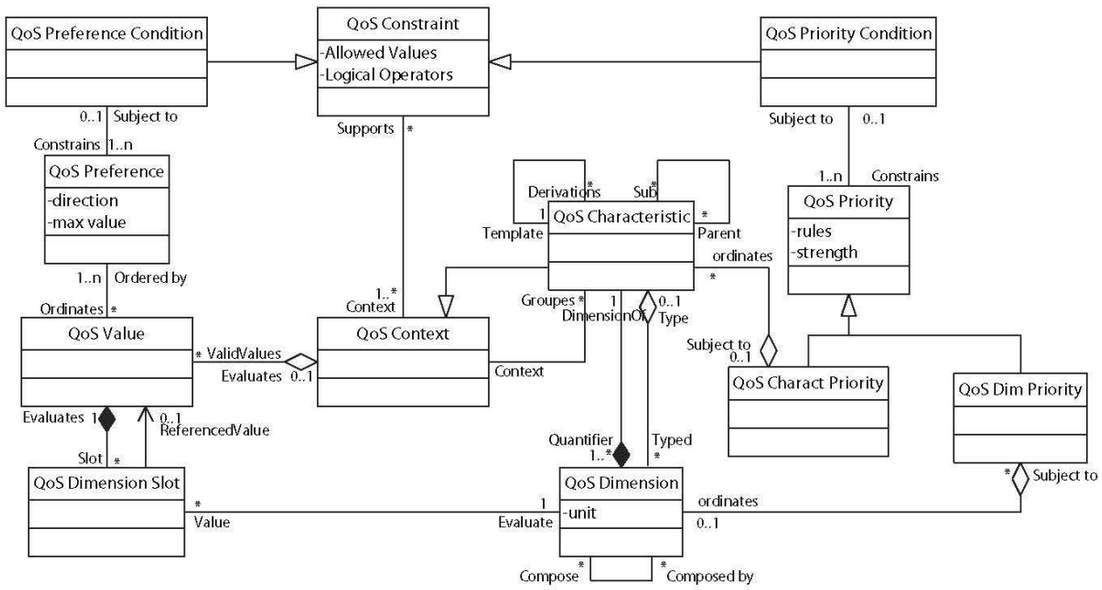


Fig. 5. UML metaclasses to user modeling

- We allow the user to express its priorities over QoS Characteristics and over QoS Dimensions by means of, respectively, *QoS Charact Priority* and *QoS Dim Priority* metaclasses whose are specializations of the *QoS Priority* metaclass. Its attribute *rules* concerns QoS Characteristics or QoS Dimensions involved in the priority and the direction of the priority while the attribute *strength* indicates the relative importance of the priority. *QoS Priority Condition* indicates conditions that need to hold in order for the priority to become applicable.
- To enable the user to express its preferences over values of QoS Characteristics and QoS Dimensions, we add a specific metaclass: *QoS Preference*. Preferences over values are defined with some attributes: *direction* states if the value has to be minimized or maximized; *max value* indicates the maximal value expected by the user and defines its preference.

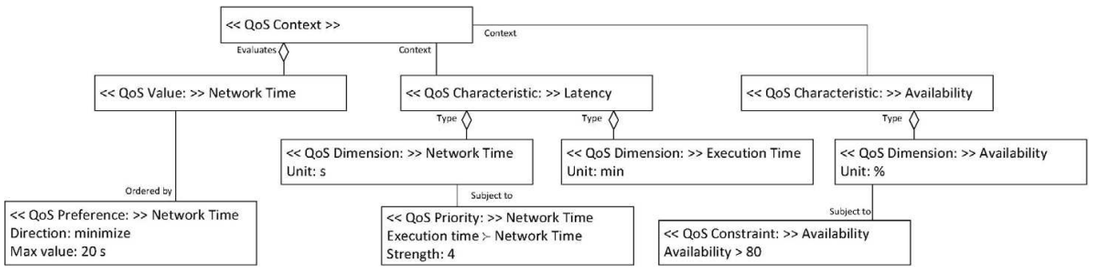


Fig. 6. User specifications

To illustrate our scoring model, we suppose a service requester who wishes to use our composite service processing FAPAR for a given area of the world while optimizing the following QoS Characteristics: *availability, cost, latency, reliability, reputation* and *security*. Some of these quality considerations are not directly quantifiable, and are measured with help of multiple QoS Dimensions (e.g.: latency is quantified by network time and execution time), others are measured with a single QoS Dimension (e.g.: the availability is a measure

provided in %). All these information are specified by the service requester with the help of our proposed QoS model. Parts of the complete specification of the user are illustrated in Figure 2.3.

3. QoS scoring of services

In order to select web services that will fulfill the different elementary tasks of the composition, the service composer must decide between them. Because web services represented in the DAH meet functional requirements, their discrimination will be made on their quality properties. To account for multiple quality properties in the reinforcement learning composition process, QoS need to be adequately aggregated. We explain in this section how the composer give an aggregated QoS score to each available service of the composition with help of Multi-Criteria Decision Making (MCDM) techniques. The QoS score is calculated by considering quality requirements expressed by the service user. To express such requirements, that must be interpretable by the service composer, the user needs an appropriate quality model. We present our QoS model and illustrate its utilization with the earth observation composite service of the ESA introduced in Subsection 2.2.
The service composer uses information specified with the QoS model in combination with Multi-Criteria Decision Making (MCDM) techniques to establish an aggregated measure of quality properties on all available services. This measure must be calculated for each service candidate of the composition. However multiple execution paths are available in the DAH representation of the composite service and, these paths can be subject to major variations in quality performance. In our ESA case study, we observed that services used to generate world-wide data are slower than services providing regional data but are also more reliable. Anyway, scores of services need to be comparable to service candidates on all paths of the composition. To achieve this global measurement, the scoring will be established by pairwise comparisons on all services suitable for any tasks of the composition.
The scoring process involves the following steps: (1) apply hard constraints on services, to restrict the set of services upon whose MCDM calculation will be made. (2) establish the hierarchy of quality properties with information related to characteristics and dimensions decomposition, each property being considered as a criterion of the MCDM model. Moreover, two distinct hierarchies are build, the first dedicated to benefits, i.e.: criteria to maximize, the second dedicated to costs, i.e.: criteria to minimize. (3) fix the priorities of quality properties by applying the Analytic Hierarchy Process (AHP) on both hierarchies. (4) give a score to each service alternative for both benefits and costs hierarchies. This step is done with the Simple Additive Weighting (SAW) process, which gives us the opportunity to score alternatives with few information given on criteria. (5) for each alternative, the ratio benefits/costs is computed by service composer and a score is linked to each available service.

3.1 Fixing hard constraints

Hard constraints on quality properties (i.e.: QoS Characteristics or QoS Dimensions) are defined by the user to restrict the set of accepted services. These are specified with the *QoS Constraint* metaclass and fix thresholds to values of a QoS Dimension. While the service composer assigns best available services to the service requester, services that do not fulfill thresholds values for the different QoS Dimensions taken into account are considered

irrelevant. Constraints allow us to decrease the number of alternative services to consider when applying MCDM - all services that do not satisfy the constraints are not considered for comparison.

The complete specification made by the service requester with the QoS model is transmitted to the service composer that will process all steps of the selection. The composer starts by rejecting services that do not fulfill hard constraints. For example, in specification given in Figure 2.3, the composer restrains available services to those that have an *Availability* higher than 80%.

3.2 Characteristics and dimensions hierarchies

Decomposition of QoS Characteristics into QoS Dimensions and QoS Dimensions into others QoS Dimensions may be used by the service composer to build a complete hierarchy of QoS properties. This information is expressed with help of the relations *Type - Typed* between the QoS Characteristic and the QoS Dimension metaclasses and *Compose - Composed by* defined over the QoS Dimension metaclass. The hierarchy established by the service composer allows to bind weights to QoS properties at different levels. This way, their relative importance is aggregated in accordance with the QoS properties that these quantify. To account for measurement of QoS Characteristics by QoS Dimensions and quantification of QoS Dimensions, we classify them into two separate hierarchies. The first is dedicated to benefits, all quality properties that have to be maximized: availability, reliability, reputation, etc. The second is designed for costs, involving quality properties to minimize: execution time, failures, cost, etc. Modality (maximize or minimize) of QoS properties is defined with the attribute direction of the QoS Value class. These two hierarchies are linked to the same global optimization goal. This top-down organization clearly indicates the contributions of lower levels of quality properties to upper ones. The final hierarchy obtained takes the form of a tree.

The second step of the service composer is to establish benefits and costs hierarchies with the information provided by the service requester. The hierarchy corresponding to expectations formulated by the requester for the ESA composite service is illustrated in Figure 3.2.

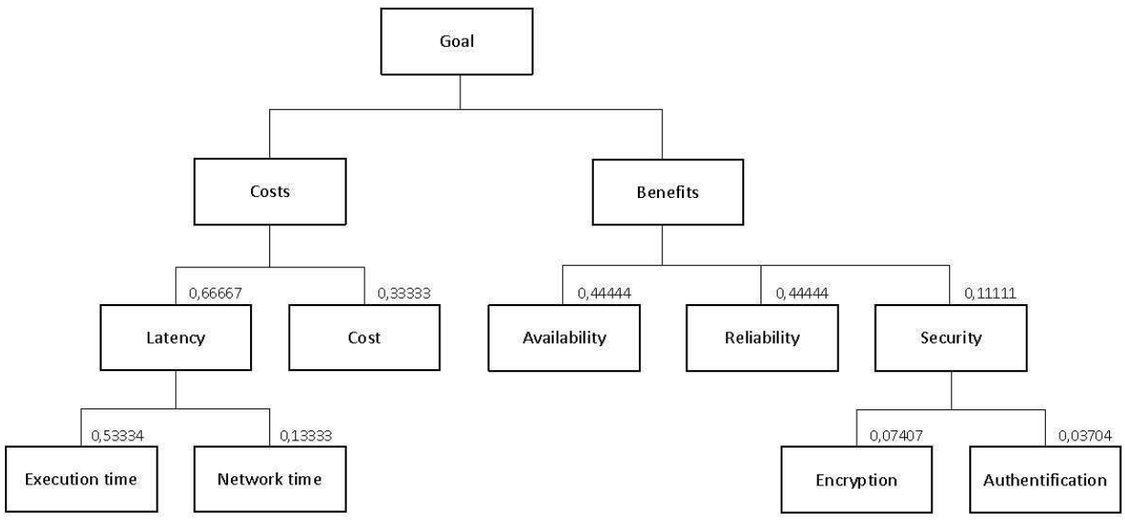


Fig. 7. Benefits and costs hierarchies

3.3 Priorities over criteria

Priorities information is used to bind weights to QoS Characteristics and QoS Dimensions, reflecting their respective relative importance. These weights are defined using QoS Priorities specifications given by the service user and are linked to the corresponding QoS properties. Once the hierarchy is established, the relative importance of each QoS property has to be fixed with a weight reflecting its contribution to the main optimization goal. These weights must be fixed independently for benefits criteria and for costs criteria to consider separately positive and negative QoS properties. To fix weights on such hierarchies, we use the Analytic Hierarchy Process (AHP) (Saaty, 1980). The Analytic Hierarchy Process fixes weights to criteria with help of comparison matrices provided for each level of criteria. For a same level, each criterion is compared with other criteria of its level on a scale fixed between 1/9 and 9. Each matrix is build with QoS Priority specifications: rules express direction of pairwise comparisons of criteria and strength fixes the value chosen by the user on the scale for the comparison. Next, weights of QoS properties are obtained with the computation of the right eigenvector of the matrix. The eigenvector is computed by raising the pairwise matrix to powers that are successively squared each time. The rows sums are then calculated and normalized. The computation is stopped when the difference between these sums in two consecutive calculations is smaller than a prescribed value. The service composer adopts a top-down approach, the weights of each level being multiplied by the weight of the quality property of its upper level to determine its relative importance on the whole hierarchy. This process is performed on both sides of the tree, for positive and negative quality properties.

The third step of the composer is to fix weights for each level of criteria with the AHP method. With the information provided by QoS Priority instance in Figure 2.3, the service composer is able to build a comparison matrix for dimensions quantifying the *Latency*. In the case or our composite service computing the FAPAR index for a given area of the world, the service requester favors the *Execution time* rather than the *Network time*. In fact, *Execution time* is the main bottleneck of the service execution due to huge quantity of data processed. This matrix is $\begin{pmatrix} 1 & 4 \\ 1/4 & 1 \end{pmatrix}$. The composer computes its eigenvector to obtain weights for this level, in the example: 0.2 for *Network Time* and 0.8 for *Execution Time*. These weights are multiplied by weights of upper levels to determine weights of the whole hierarchy that are illustrated in Figure 3.2.

3.4 QoS scoring with user preferences

Preferences information specified by the user on QoS Values is used by the service composer to compute the score of the service. We use this information to determine what values are preferred for a given QoS Characteristic or QoS Dimension. The priorities of quality properties have been fixed with weights reflecting their relative importance. Preferences on values allow us to discriminate services on a given criterion. To quantify these preferences, we rely on a specific class of MCDM methods: scoring methods (Figueira et al., 2005) and more specifically the Simple Additive Weighting (SAW) method (Hwang & Yoon, 1981). This method is based on the weighted average. An evaluation score is calculated for each alternative by multiplying the scaled value given to the alternative of that attribute with the weights given by the AHP method. Next, these products are summed for all criteria involved in the decision making process. Each service alternative is evaluated on both hierarchies, i.e.: benefits and costs, with the following formula:

$$s_u = \sum w_i \times x_{ui}^* \quad (1)$$

Where w_i is the weight of the QoS property *i* get with the AHP method and x_{ui}^* is the scaled score of the service alternative *u* on the QoS property *i*.
The scores for the QoS properties are measured with different scales, i.e.: percentage, second, level, etc. Such measurement scales must be standardized to a common dimensionless unit before applying the SAW method. The scaling of a service alternative for a given QoS property is evaluated with the following formula:

$$x_{ui}^* = \frac{x_{ui}}{x_i^{max}} \quad (2)$$

where x_{ui}^* is the scaled score of the service alternative *u* on the QoS property i. x_{ui} is the score of the service alternative *u* on the QoS property *i* expressed with its original unit. x_i^{max} is the maximal possible score on the QoS property *i*. This maximal score is expressed by the user with help of the *max value* attribute of the QoS Preference class illustrated in Figure 2.3. When the unit of the QoS Property is a percentage, the maximal value is systematically equal to 100. If the unit is a time period as second, the user defines himself the maximal value. So, the scaled scores will reflect the preferences of the user with means of the relative importance of the maximal value by contrast to observed values.
Once weights reflecting the relative importance of each QoS property have been fixed, the fourth step of the service composer is to define the score of each alternative for both benefits and costs hierarchies with user preferences. It uses the SAW method and begins by scaling the score of all alternatives on all QoS properties involved in the selection process. For example, in Figure 2.3, the max value proposed by the service user for the Network time is 20 sec. With a service alternative offering a Network Time of 13 sec, the scaled score of this service for the Network Time QoS property is 65%. This score is then multiplied by 0,13333, the weight of the Network Time. This process is summed for all QoS properties considered and repeated for all existing service alternatives on both hierarchies.

3.5 Benefits/costs analysis

Scores of services alternatives get with the SAW method on both hierarchies define the relative performance of services on positive properties (benefits) and negative properties (costs). Benefits should be maximized while costs have to be minimized, to aggregate both considerations into a single measure of performance, the AHP MCDM method proposes to execute the benefits/costs ratio (Figueira et al., 2005). The benefits/costs ratio is evaluated with the following formula:

$$r_u = \frac{s_u^{benefits}}{s_u^{costs}} \quad (3)$$

where r_u is the final score of the service alternative *u*. $s_u^{benefits}$ is the score of the service *u* on the benefits hierarchy and s_u^{costs} is the score of the service a on the costs hierarchy.

The last step of the composer is then to compute the benefits/costs ratio of each alternative as suggested by some AHP variations. E.g.: if a service alternative has a score of 0,8126 for its benefits hierarchy and a score of 0,7270 for its costs hierarchy, the final score of its service is $\frac{0,8126}{0,7270} = 1,1177$. The respective score of each service is then linked to reflect its relative performance.

4. Web services composition with randomized RL algorithm

An important issue is the selection of WS that are to participate in performing the process described in the composition model. This problem is referred to as the task allocation problem in the remainder.

Reinforcement Learning (RL) (see, e.g., (Sutton & Barto, 1998) for an introduction) is a particularly attractive approach to allocating tasks to WS. RL is a collection of methods for approximating optimal solutions to stochastic sequential decision problems (Sutton & Barto, 1998). An RL system does not require a teacher to specify correct actions. Instead, the learning agent tries different actions and observes the consequences to determine which are best. More specifically, in the RL framework, a learning agent interacts with an environment over some discrete time scale $t = 0,1,2,3,\ldots$. At each time step t, the environment is in some state, k_t. The agent chooses an action, u_t, which causes the environment to transition to state k_{t+1} and to emit a feedback, r_{t+1}, called ``reward''. A reward may be positive or negative, but must be bounded and it informs the agent on the performance of the selected actions. The next state and reward depend only on the preceding state and action, but they may depend on it in a stochastic fashion. The objective of reinforcement learning is to use observed rewards to learn an optimal (or nearly optimal) mapping from states to actions, which is called an optimal policy, Π. An optimal policy is a policy that maximizes the expected total reward (see, § 4.2, Eq. 5). More precisely, the objective is to choose action u_t, for all $t \leq 0$, so as to maximize the expected return. Using the terminology of this paper, RL can be said to refer to trial-and-error methods in which the composer learns to make good allocations of WS to tasks through a sequence of " interactions" . In task allocation, an interaction consists of the following:

1. The composer identifies the task to which a WS is to be allocated.
2. The composer chooses the WS to allocate to the task.
3. The composer receives a reward after the WS executes the task. Based on the reward, the composer learns whether the allocation of the given WS to the task is appropriate or not.
4. The composer moves to the next task to execute (i.e., the next interaction takes place).

One advantage of RL over, e.g., queuing-theoretic algorithms (e.g., (Urgaonkar et al., 2005)), is that the procedure for allocating WS to tasks is continually rebuilt at runtime: i.e., the composition procedure changes as the observed outcomes of prior composition choices become available. The WS composer tries various allocations of WS to tasks, and learns from the consequences of each allocation. Another advantage is that RL does not require an explicit and detailed model of either the computing system whose operation it manages, nor of the external process that generates the composition model. Finally, being grounded in Markov Decision Processes, the RL is a sequential decision theory that properly treats the

possibility that a decision may have delayed consequences, so that the RL can outperform alternative approaches that treat such cases only approximately, ignore them entirely, or cast decisions as a series of unrelated optimizations.

One challenge in RL is the tradeoff between *exploration* and *exploitation*. Exploration aims to try new ways of solving the problem, while exploitation aims to capitalize on already well-established solutions. Exploration is especially relevant when the environment is changing: good solutions can deteriorate and better solutions can appear over time. In WS composition, exploitation consists of learning optimal allocations of WS to tasks, and systematically reusing learned allocations. Without exploration, the WS composer will not consider allocations different than those which proved optimal in the past. This is not desirable, since in absence of exploration, the WS composer is unaware of changes in the availability of WS and appearance of new WS, so that the performance at which the composition is fulfilled inevitably deteriorates over time in an open and distributed service-oriented system.

Two forms of exploration can be applied: *preliminary* and *continual online* exploration. The aim with *preliminary* exploration is to discover the state to reach, and to determine a first optimal way to reach it. As the composition model specifies the state to reach in WS composition, *continual online* exploration is of particular interest: therein, the set of WS that can be allocated to tasks is continually revised, so that future allocations can be performed by taking into account the availability of new WS, or the change in availability of WS used in prior compositions. Preliminary exploration is *directed* if domain-specific knowledge is used to guide exploration (e.g., (Thrun, 1992b; Thrun, 1992a; Thrun et al., 2005; Verbeeck, 2004)). In *undirected* preliminary exploration, the allocation of new WS to tasks is randomized by associating a probability distribution to the set of competing WS available for allocation to a given task.

To avoid domain-specificity in this paper, the RL algorithm in MCRRL relies on *undirected continual exploration*. Both exploitation and undirected continual exploration are used in WS composition: exploitation uses available data to ground the allocation decision in performance observed during the execution of prior compositions, whereas exploration introduces new allocation options that cannot be identified from past performance data. This responds to the first requirement on WS composition procedures (item 1, § 1), namely that optimal WS compositions will be built and revised at runtime, while accounting for change in the availability of WS and the appearance of new WS. As shown in the remainder (see, § 4.1), the WS composition problem can be formulated as a global optimization problem which follows either a *deterministic shortest-path* (in case the effects of WS executions are deterministic) or a *stochastic shortest-path* formulation. Requirement 4 (§ 1) is thus also addressed through the use of RL to guide WS composition. Since the RL approach can be based on observed performance of WS in compositions, and the algorithm in MCRRL accepts multiple criteria and/or constraints (see, § 3 and § 4.1), requirements 2 and 3 (§ 1) are fulfilled as well.

4.1 Task-allocation problem

If RL is applied to task allocation, the exploration/ exploitation issue can be addressed by periodically readjusting the policy for choosing task allocations and re-exploring up-to-now suboptimal execution paths (Mitchell, 1997; Sutton & Barto, 1998). Such a strategy is, however, suboptimal because it does not account for exploration. The Randomized

Reinforcement Learning (RRL) algorithm introduced in (Saerens et al., 2004) is adapted herein to task allocation in WS composition, allowing the assignment of tasks to WS while: (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the performance of new agents so as to continually adjust task allocation, and (iv) exploring new options in task allocation. The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a task to a task specialist. This permits the continual measurement and control of exploration.

The task-allocation problem that the RRL resolves amounts to the composer determining the WS to execute the tasks in a given process model. By conceptualizing the process of the composition model as a DAH (see, § 2.2.2), the task-allocation problem amounts to a deterministic shortest-path problem in a *directed weighted hypergraph*. In the hypergraph, each node is a *step* in WS composition problem and an edge corresponds to the *allocation of a task* t_k *to a WS* $w_{k,u}^{WS}$, where u ranges over WS that can execute t_k according to the criteria set with the QoS model. Each individual allocation of a task to a WS incurs a cost $c(t_k, w_{k,u}^{WS})$, whereby this " cost" is a function of the aggregated criteria (as discussed earlier § 3) formulated so that the minimization of cost corresponds to the optimization of the aggregated criteria (i.e., minimization or maximization of aggregation value). For illustration, consider the DAH representation of our composite ESA service in Figure 3.

The task allocation problem is a global optimization problem: learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints (specified with the QoS model). At the initial node in the graph (in Fig.3, blank node), no tasks are allocated, whereas when reaching the destination node (last 'Pd' node in the same figure), all tasks are allocated.

The remainder of this Section is organized as follows: § 4.2 introduces the notations, the standard deterministic shortest-path problem, and the management of continual exploration. § 4.3 introduces the unified framework integrating exploitation and exploration presented in (Achbany et al., 2005). Finally, § 4.3 describes our procedure for solving the deterministic shortest-path problem with continual exploration.

4.2 RL formulation of the problem

At a state k_i of the task allocation problem, choosing an allocation of $t_{k_i,l}$ (where l ranges over tasks available in state k_i) to $w_{k_i,u}^{WS}$ (i.e., moving from k_i to another state) from a set of potential allocations $U(k_i)$ incurs a cost $c(t_{k_i,l}, w_{k_i,u}^{WS})$. Cost is an inverse function of the aggregated criteria the user wishes to optimize (see, § 3), say r. The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic (Christofides, 1975). Task allocation proceeds by comparing WS over estimated $\hat{r}$ values and the hard constraints to satisfy (see, s 3.1). The allocation $(t_{k_i,l}, w_{k_i,u}^{WS})$ is chosen according to a *Task Allocation policy (TA)* Π that maps every state k_i to the set $U(k_i)$ of admissible allocations with a certain probability distribution $\pi_{k_i}(u)$, i.e., $U(k_i): \Pi \equiv \{\pi_{k_i}(u), i = 0,1,2,\ldots,n\}$. It is assumed that: (i) once the action (i.e., allocation of a given task to a WS) has been chosen, the sate next to k_i, denoted k_i', is known deterministically, $k_i' = f_{k_i}(u)$ where f is a one-to-one mapping from states and actions to a resulting state; (ii) different actions lead to different states; and (iii) as in (Bertsekas, 2000), there is a special cost-free *destination* state;

once the composer has reached that state, the task allocation process is complete. Although the current discussion focuses on the deterministic case, extension to the stochastic case is discussed elsewhere (Achbany et al., 2005) due to format constraints.

As remind, one of the key features of reinforcement learning is that it explicitly addresses the exploration/exploitation issue as well as the online estimation of the probability distributions in an integrated way. Then, the exploration/ exploitation tradeoff is stated as a global optimization problem: find the exploration strategy that minimizes the expected cumulated cost, while maintaining fixed degrees of exploration at same nodes. In other words, exploitation is maximized for constant exploration. To control exploration, entropy is defined at each state.

The degree of exploration E_{k_i} at state k_i is quantified as:

$$E_{k_i} = -\sum_{u \in U(k_i)} \pi_{k_i}(u) \log \pi_{k_i}(u) \tag{4}$$

which is the entropy of the probability distribution of the task allocations in state k_i (Cover & Thomas, 1991; Kapur & Kesavan, 1992). E_{k_i} characterizes the uncertainty about the allocation of a task to a WS at k_i. It is equal to zero when there is no uncertainty at all ($\pi_{k_i}(u)$ reduces to a Kronecker delta); it is equal to $\log(n_{k_i})$, where n_{k_i} is the number of admissible allocations at node k_i, in the case of maximum uncertainty, $\pi_{k_i}(u) = 1/n_{k_i}$ (a uniform distribution).

The exploration rate $E^r_{k_i} \in 0,1]$ is the ratio between the actual value of E_{k_i} and its maximum value: $E^r_{k_i} = E_{k_i} / \log(n_{k_i})$.

Fixing the entropy at a state sets the exploration level for the state; increasing the entropy increases exploration, up to the maximal value in which case there is no more exploitation--- the next action is chosen completely at random (using a uniform distribution) and without taking the costs into account. Exploration levels of composers can thus be controlled through exploration rates. Service provision then amounts to minimizing *total expected cost* $V_\pi(k_0)$ accumulated over all paths from the initial k_0 to the final state:

$$V_\pi(k_0) = E_\pi\left[\sum_{i=0}^{\infty} c(k_i, u_i)\right] \tag{5}$$

The expectation E_π is taken on the policy Π that is, on all the random choices of action u_i in state k_i.

4.3 Computation of the Optimal Policy

The composer begins with task allocation from the initial state and chooses from state k_i the allocation of a WS u to a task $t_{k_i,l}$ with a probability distribution $\pi_{k_i}(u)$, which aims to exploration. The composer then performs the allocation of the task $t_{k_i,l}$ to a WS u and the associated aggregated quality score, the cost $c(t_{k_i,l}, w_u^{WS})$ is incurred and is denoted, for simplicity $c(k_i, u)$ (note that this score may also vary over time in a dynamic environment); the composer then moves to the new state, $k_{i'}$. This allows the composer to update the

estimates of the aggregated quality score of the policy, and of the average aggregated quality value until destination; these estimates will be denoted by $c(k_i, i)$, $\pi_{k_i}(i)$ and $V(k_i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no edge going backward (i.e., there exists no edge linking a state $k_{i'}$ to a state k_i where $k_{i'}$ is a successor state of k_i ($k_{i'} > k_i$), is as follows (a detailed treatment can be found in (Achbany et al., 2005)):

1. *Initialization phase:* Set $V(k_d) = 0$, which is the expected cost at the destination state.
2. *Computation of the TA policy and the expected cost under exploration constraints:* For $k_i = (k_d - 1)$ to the initial state k_0, compute:

$$\begin{cases} \pi_{k_i}(u) = \dfrac{\exp\left[-\theta_{k_i}\left(c(k_i,u) + V(k'_{i,u})\right)\right]}{\sum_{u' \in U(k_i)} \exp\left[-\theta_{k_i}\left(c(k_i,u') + V(k'_{i,u'})\right)\right]}, \\ V(k_i) = \sum_{u \in U(k_i)} \pi_{k_i}(u)\left[c(k_i,u) + V(k'_{i,u})\right] \text{ for } k_i \neq k_d \end{cases} \quad (6)$$

where $k'_{i,u} = f_k(u)$, $k'_{i,u'} = f_k(u')$ and θ_{k_i} is set in order to respect the prescribed degree of entropy at each state (see Eq.4 which can be solved by a simple bisection search).
Various approaches can be applied to update the estimated criterion $\hat{r}_u$; e.g., exponential smoothing leads to:

$$r_u \leftarrow \alpha \bar{r}_u + (1-\alpha) r_u \quad (7)$$

where $\bar{r}_u$ is the observed value of the criterion for w_u^{WS} and $\alpha \in]0,1[$ is the smoothing parameter. Alternatively, various stochastic approximation updating rules could also be used. The composer updates its estimates of the criterion each time a WS performs a task and the associated cost is updated accordingly.

5. Simulation results

Experimental setup. Task allocation for the service provision problem diplayed in Fig.3 was performed. A total of three distinct WS were made available for each distinct task. Each $w_{k,u}$ is characterized by its actual r_u which is an indicator of the WS's performance over the optimization criterion (see, § 4.2). In this simulation, it will simply be the probability of successfully performing the task (1 -- probability of failure). In total, 42 WS are available to the Composer for task allocation. For all WS u, r_u takes its value $\in 0,1]$; for 70% of the WS, the actual r_u is *hidden* (assuming it is unknown to the Composer) and its initial expected value, r_u, is set, by default, to 0.3 (high probability of failure since the behavior of the WS has never been observed up to now), while actual r_u value is available to the Composer for the remaining 30% (assuming these WS are well known to the Composer). Actual r_u is randomly assigned from the interval $[0.5,1.0]$ following a uniform probability distribution. It has been further assumed that $c(t_i, w_u) = -ln(r_u)$, meaning that it is the product of the r_u

along a path that is optimized (this is a standard measure of the reliability of a system). After all tasks are allocated, the selected WS execute their allocated tasks according to their actual r_u value (with failure $1-r_u$). The estimated WS criterion $\hat{r}_u$ is then updated by exponential smoothing, according to Eq.7. In Eq.7, $\overline{r}_u$ equals 1 if w_u is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are of course updated in terms of the r_u and each time a complete allocation occurs, the probability distributions of choosing a WS are updated according to Eq.6. 10,000 complete allocations were simulated for exploration rate 20%.

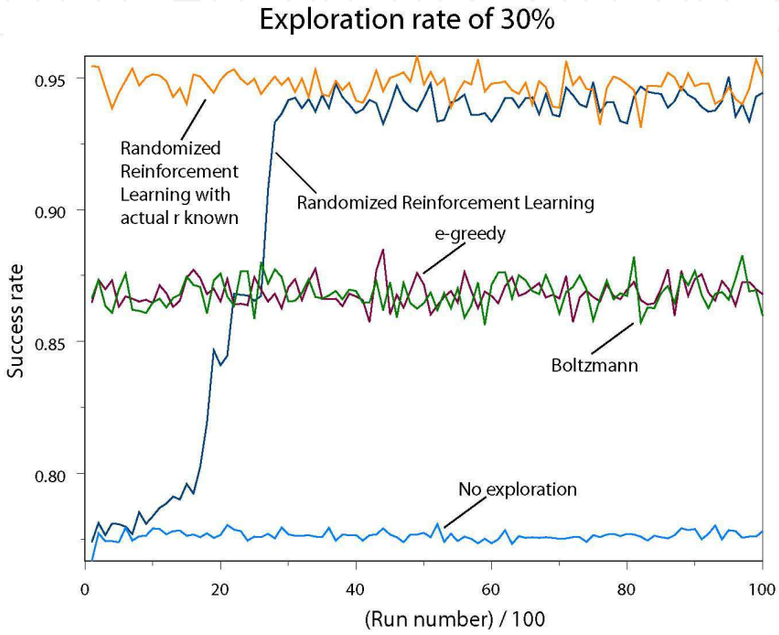


Fig. 8. Success rate in terms of run number, for an exploration rate of 20%, and for the five methods (no exploration, actual r known, ε *-greedy*, naive Boltzmann, RRL).

Results. The RRL is compared to two other standard exploration methods, ε *-greedy* and *naive Boltzmann* (see (Achbany et al., 2005) for details), while tuning their parameters to ensure the same exploration level as for RRL. The *success rate* is defined as the proportion of services that are successfully completed (i.e., all tasks composing the service are allocated and executed successfully) and is displayed in Fig. 4 in terms of the run number (one run corresponding to one complete assignment of tasks, criterion estimation and probability distribution update). Fig. 4 shows the RRL behaves as expected. Its performance converges almost to the success rate of the RRL in which all actual r are known from the outset (i.e., need not be estimated)---and indicate that exploration clearly helps by outperforming the allocation system without exploration (which has a constant 75% success rate). Fig.5 compares the three exploration methods by plotting the average absolute difference between actual r_u and estimated r_u criterion values for a 30% exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new agents---i.e., exploration is useful for directing Composer behavior in dynamic, changing, and open architectures, i.e., in the SCA.

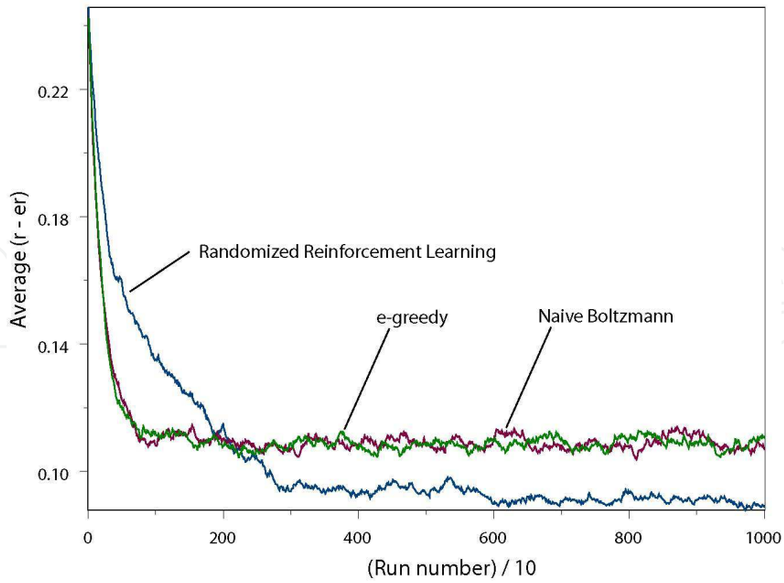


Fig. 9. Average absolute difference between actual (r) and estimated (r) criterion values in terms of run number, for three exploration methods (ε -greedy, naive Boltzmann, RRL).

6. Related work

Various possibilities for representation of web services composition have already been addressed. Jaeger et al. use composition patterns (sequence, loop, xor, and, or , etc.) to represent structural elements of the composition. Hamadi (Hamadi & Benatallah, 2003) and Benetallah and Fu et al. (Fu et al., 2006) approaches refer both to Petri nets for modeling web services control flow. Rather than using such patterns and their associated aggregation rules, we choose, as Zeng et al. (Zeng et al., 2003b; Zeng et al., 2004), Benatallah and Dumas (Benatallah et al., 2002) and Zhang et al. (Zhang et al., 2007), to control services composition with the help of statecharts.

While statecharts and their associated formal semantic are used to represent the different tasks entering in the composition, Directed Acyclic Graph (DAG) (Gu & Nahrstedt, 2002) are used to represent alternative web services allowing to fulfill these tasks. Zeng (Zeng et al., 2004) proposes to model alternatives with multiple execution paths derived from statecharts possibilities. We choose to represent our composition possibilities with a Directed Acyclic Hypergraph (DAH) where nodes represent functional steps of execution and edges are web services alternatives to fulfill a given task.

Our selection of services that will enter in the services composition is based on their quality properties, i.e., their QoS. To lead the selection from the requester view, we provide him a QoS model enabling to specify its expectations about quality behavior. This behavior is expressed by relationships between characteristics and dimensions, priorities between quality properties and preferences over values. The preference over values has already been addressed in other approaches under the form of a direction attribute indicating if a property has to be maximized or minimized (Jaeger et al., 2004; Liu et al.,

2004; Naumann et al., 1999; Zeng et al., 2004). Our preference structure offers more information, we allow the user to specify conditions and indifference thresholds. The priority relationship is defined in some proposals with means of a weight attribute associated to quality properties (Jaeger et al., 2004; Zeng et al., 2004). Our model authorizes weights binded to quality properties at different levels and we define a method to fix adequately these weights.

Most QoS composition approaches aim at summing QoS values of services entering in the composition rather than computing their individual performance (Cardoso et al., 2004; Cheng et al., 2006; Jaeger et al., 2005; Yu & Lin, 2004; Yu & Lin, 2005; Zeng et al., 2003b; Zhang et al., 2007). In our MCRRL proposal, we focus on the individual evaluation of each web service candidate to the whole composition. Rather than using Reinforcement Learning computation, Zeng and colleagues (Zeng et al., 2003b) proceed to finding optimal WS compositions through linear programming techniques. In contrast to RL, their approach considers each WS composition as a new problem to solve, so that there is no learning. Canfora and colleagues (Canfora et al., 2004) use genetic algorithms, avoiding thus the need for a linear objective function and/or linear constraints in the search for the optimal WS composition (required for the linear programming approach (Zeng et al., 2003b)). MCRRL improves responsiveness of the system to varying availability and appearance of new WS because of exploration. MCRRL allows the execution of potentially complex processes, permits concurrency, while assuming that the set of available WS is changing. One distinctive characteristic the composer's behavior suggested in the present paper is that the MCRRL accounts for a vector of criteria when allocating tasks, including QoS, service provision deadline, provision cost, explicit user preferences, and agent reputation. Feedback mechanisms are also used by Maximilien and Singh (Maximilien & Singh, 2005) that propose service selection driven by trust values assigned to individual services. Trust is extracted from user-generated reports of past service performance (as usual in reputation systems) over qualities defined by a system-specific QoS ontology. Level of trust depends on the degree to which reputation and quality levels advertised by the provider match. Similar approaches have been proposed, yet fail to address service selection in open, distributed MAS architecture, furthermore without dynamic allocation so that autonomic requirements are not fulfilled. By basing selection on trust only and generating levels of trust from advertised and user-observed behavior, Maximilien and Singh's approach involves learning driven by exploitation of historical information, without exploration.

7. Conclusions and future work

This paper advocates that WS compositions optimal w.r.t. a set of criteria need to be learned at runtime and revised as new WS appear and availability of old WS changes, whereby the learning should be based on observed WS performance, and not the performance values advertised by the service providers. To enable such learning, a selection procedure is needed which both *exploits* the data on observed WS performance in the past, and *explores* new composition options to avoid excessive reliance on past data.

As a response, this paper proposes the Multi-Criteria Randomized Reinforcement Learning (MCRRL) approach to WS composition. MCRRL combines a generic service request and the

Randomized Reinforcement Learning (RRL), a reinforcement learning algorithm. The SR model describes the process to execute by the WS composition and the criteria and constraints to meet when executing it. The RRL selects the WS for performing tasks specified in the service request. The algorithm decides on the WS to select among competing WS based on multiple criteria, while both exploiting available WS performance data and exploring new composition options.

MCRRL responds to four common requirements when defining a task allocation procedure for WS composition. First, the RRL uses both exploitation and undirected continual exploration in WS composition: exploitation uses available data to ground the allocation decision in performance observed during the execution of prior compositions, whereas exploration introduces new allocation options that cannot be identified from past performance data. Optimal WS compositions are thus identified revised at runtime. Second, the generic SR model combined with the optimization approach in the RRL allow many criteria for comparing alternative task allocations. Third, the comparison over various criteria relies on observed performance over the given criteria, instead of vales advertised by service providers. Finally, the algorithm can be extended to allow underterministic outcomes of WS executions (as explained elsewhere (Achbany et al., 2005)).

Since undirected exploration may be costly in actual applications, future work will investigate the performance of MCRRL within realistic applications, so that the approach can be optimized for practical settings.

8. Acknowledgments

We are grateful to Emmanuel Mathot of the European Space Agency, who provided precise information about the GPOD project and assisted our efforts in describing quality information of services related to the GPOD project.

9. References

Achbany, Y.; Fouss, F.; Yen, L.; Pirotte, A. & Saerens, M. (2005) Tuning Continual Exploration in Reinforcement Learning. *Technical report, http://www.isys.ucl.ac.be/staff/francois/Articles/ Achbany2005a.pdf*.

Benatallah, B.; Sheng, Q. Z.; Ngu, A. H. & Dumas M. (2002) Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. *Proceedings of the International Conference on Data Engineering*, pages 0297, Los Alamitos, CA, USA.

Bertsekas, D. P. (2000) *Dynamic programming and optimal control*. Athena sientific.

Canfora, G.; Di Penta, M.; Esposito, R. & Villani, M.-L. (2004) A Lightweight Approach for QoS-Aware Service Composition. *Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04)*, pages 36-47.

Cardoso, J.; Sheth, A. P.; Miller, J. A.; Arnold, J. & Kochut, K. (2004) Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3):281-308.

Chen , Y.P.; Zeng-Zhi, L.; Qin-Xue, J. & Chuang, W. (2006) Study on QoS Driven Web Services Composition. *Frontiers of WWW Research and Development - APWeb 2006*, :702--707.

Christofides, N. (1975) *Graph theory: An algorithmic approach*. Academic Press.

Cover, T. M. & Thomas, J. A. (1991) *Elements of information theory*. John Wiley and Sons.

D'Ambrogio, A. (2006) A Model-driven WSDL Extension for Describing the QoS ofWeb Services. *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 789--796, Washington, DC, USA, 2006, IEEE Computer Society.

Figueira, J.; Greco, S. & Ehrgott, M. (2005) *Multiple Criteria Decision Analysis: State of the Art Surveys*. Springer Verlag, Boston, Dordrecht, London.

Fu, Y.; Zhijiang, D. & Xudong, H. (2006) Modeling, validating and automating composition of web services. *ICWE '06: Proceedings of the 6th international conference on Web engineering*, pages 217--224, New York, NY, USA, 2006. ACM.

Gu, X. & Nahrstedt, K. (2002) A scalable QoS-aware service aggregation model for peer-to-peer computing grids. *Proceedings of the IEEE HPDC-11*, 2002.

Hamadi, R. & Benatallah, B. (2003) A Petri net-based model for web service composition. *ADC '03: Proceedings of the 14th Australasian database conference*, pages 191--200, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

Hwang, C.L. & Yoon, K. (1981) *Multiple attribute decision making : Methods and applications*. Springer-Verlag, 1981.

Jaeger, M. C.; Rojec-Goldmann, R. & Muhl, G. (2004) QoS Aggregation for Web Service Composition using Workflow Patterns. *edoc*, 00:149-159.

Jaeger, M. C.; Rojec-Goldmann, R. & Muhl, G. (2005) QoS Aggregation in Web Service Compositions. *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 181--185, Washington, DC, USA, 2005. IEEE Computer Society.

Kapur, J. N. & Kesavan, H. K. (1992) *Entropy optimization principles with applications*. Academic Press, 1992.

Keller, A. & Ludwig, H. (2003) The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 11(1):57--81.

Liu, Y.; Ngu, A. H. & Zeng, L. Z. (2004) QoS Computation and Policing in Dynamic Web Service Selection. pages 66--73, New York, NY, USA, ACM Press.

Maximilien, E. M. & Singh, M. P. (2005) Multiagent System for Dynamic Web Services Selection. *Proceedings of the Int. Conf. Auton. Agents and Multi-Agent Syst.*, 2005.

McIlraith, S. A. & Martin, D. L. (2003) Bringing Semantics to Web Services. *IEEE Intelligent Systems*, 18(1):90--93.

Menascé, D. A. (2002) QoS Issues in Web Services. *IEEE Internet Computing*, 6(6):72--75, 2002.

Mitchell, T. M. (1997) *Machine learning*. McGraw-Hill Compagnies, 1997.

Naumann, F.; Leser, U. & Freytag, J. C. (1999) Quality-driven Integration of Heterogenous Information Systems. *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 447-458, Edinburgh, UK, 1999.

OMG / Business Process Management Initiative. Business Process Modeling Notation Specification. Final Adopted Specification dtc/06-02-01, 2006.Technical report, Object Management Group / Business Process Management Initiative., 2006.

OMG. UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms. Technical report, Object Management Group, 2006.

OMG. UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms. Technical report, Object Management Group, 2006.

Papazoglou, M. P. & Georgakopoulos, D. (2003) Service-oriented computing. *Commun. ACM*, 46(10).

Saaty, T.L. (1980) *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, New york.

Saerens, M.; Souchon, N.; Renders, J. M. & Decaestecker, C. (2004) Decision-making under uncertainty about the class priors. *Submitted for publication*.

Sutton, R.S. & Barto, A.G. (1998) *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Thrun, S. (1992) The Role of Exploration in Learning Control. In D.A. White and D.A. Sofge, editors, *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, Florence, Kentucky 41022.

Thrun, S.; Burgard, W. & Fox, D. (2005) *Probabilistic Robotics*. MIT Press.

Thrun, S. (1992) Efficient Exploration In Reinforcement Learning. Technical report, Pittsburgh, PA, USA.

Urgaonkar, B.; Pacifici, G.; Shenoy, P.; Spreitzer, M. & Tantawi, A. (2005) An analytical model for multi-tier internet services and its applications. *SIGMETRICS Perform. Eval. Rev.*, 33(1):291--302.

Verbeeck, K. (2004) Coordinated Exploration in Multi-Agent Reinforcement Learning. PhD thesis, Vrije Universiteit Brussel, Belgium.

Walsh, A. E. (2002) *Uddi, Soap, and Wsdl: The Web Services Specification Reference Book*. Prentice Hall Professional Technical Reference, 2002.

Tao, Y. & Kwei-Jay, L. (2005) Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. *Service-Oriented Computing - ICSOC 2005*, :130--143.

Tao, Y. & Kwei-Jay, L. (2004) Service Selection Algorithms for Web Services with End-to-End QoS Constraints. *CEC '04: Proceedings of the IEEE International Conference on E-Commerce Technology (CEC'04)*, pages 129--136, Washington, DC, USA, 2004. IEEE Computer Society.

Zacharia, G. & Maes, P. (2000) Trust Management Through Reputation Mechanisms. *Applied Artificial Intelligence*, 14:881-907.

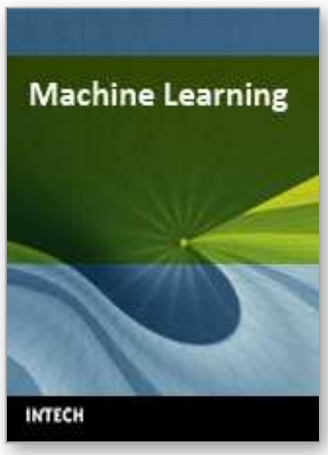
Zeng, L.; Benatallah, B.; Dumas, M.; Kalagnanam, J. & Sheng, Q. Z. (2003) Quality Driven Web Services Composition. *Proc. Int. Conf. on World Wide Web (WWW2003)*, 2003.

Zeng, L.; Benatallah, B.; Ngu, A. H.; Dumas, M.; Kalagnanam, J. & Sheng, H. (2004) QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.*, 30(5):311--327.

Zeng, L; Jeng, J.-J.; Kumaran, S. & Kalagnanam, J. (2003) Reliable Execution Planning and Exception Handling for Business Process. *Technologies for E-Services*, pages 119-130.

Zhang, C.; Chang, R. N.; Perng, C.-S.; So, E.; Tang, C. & Tao, T. (2007) QoS-Aware Optimization of Composite-Service Fulfillment Policy. *Services Computing, 2007. SCC 2007. IEEE International Conference on*, :11--19.

Zhou, C.; Chia, L.-T. & Lee, B.-S. (2004) DAML-QoS Ontology for Web Services. *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pages 472, Washington, DC, USA, 2004. IEEE Computer Society.



Machine Learning

Edited by Abdelhamid Mellouk and Abdennacer Chebira

ISBN 978-953-7619-56-1
Hard cover, 450 pages
Publisher InTech
Published online 01, January, 2009
Published in print edition January, 2009

Machine Learning can be defined in various ways related to a scientific domain concerned with the design and development of theoretical and implementation tools that allow building systems with some Human Like intelligent behavior. Machine learning addresses more specifically the ability to improve automatically through experience.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Stéphane Dehousse, Stéphane Faulkner, Caroline Herssens, Ivan J. Jureta and Marcos Saerens (2009). Learning Optimal Web Service Selections in Dynamic Environments when Many Quality-of-Service Criteria Matter, Machine Learning, Abdelhamid Mellouk and Abdennacer Chebira (Ed.), ISBN: 978-953-7619-56-1, InTech, Available from: http://www.intechopen.com/books/machine_learning/learning_optimal_web_service_selections_in_dynamic_environments_when_many_quality-of-service_criteri

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen