

QoS-based Late-Binding of Service Invocations in Adaptive Business Processes

Pierre Châtel, Jacques Malenfant, Isis Truck

▶ **To cite this version:**

Pierre Châtel, Jacques Malenfant, Isis Truck. QoS-based Late-Binding of Service Invocations in Adaptive Business Processes. ICWS 2010 - 15th IEEE International Conference on Web Services, Jul 2010, Miami, FL, United States. pp.227-234, 10.1109/ICWS.2010.74 . hal-01243537

HAL Id: hal-01243537
https://hal.science/hal-01243537

Submitted on 16 Dec 2015

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

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



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

QoS-based Late-Binding of Service Invocations in Adaptive Business Processes

Pierre Châtel*, Jacques Malenfant† and Isis Truck‡
*Thales Communications France 1-5 avenue Carnot, Massy, 91883, France
Email: pierre.chatel@thalesgroup.com
†Université Pierre et Marie Curie-Paris 6 CNRS, UMR 7606 LIP6
104 av. du Président Kennedy, Paris, 75016, France
Email: Jacques.Malenfant@lip6.fr
‡LIASD – EA 4383, Université Paris 8
2 rue de la Liberté, Saint-Denis Cedex, 93526, France
Email: truck@ai.univ-paris8.fr

***Abstract*—Computing has reached the time of distributed applications everywhere. Service-oriented architectures are more and more used to organize such complex and highly dynamic applications into business processes calling services discovered in registries at load-time. In this context, Quality of Service (QoS) and agility in business processes become key issues. Instead of binding business processes to services at load-time, this paper proposes to monitor sets of candidate services for their current QoS and to choose among them at call-time. This new form of late-binding paves the way to more agile and robust applications called adaptive business processes. Besides the conceptual background and implementation of this late-binding in an industrial-strength web service platform, this paper presents the LCP-net formalism introduced to provide programmers with a mean to express qualitatively their preferences among the different QoS properties of services, hence tackling the multi-criteria decision making arising from the run-time choice among candidate services given several unrelated QoS properties.**

"Broadly speaking, the history of software development is the history of ever later binding times..." [1]

I. Introduction

Service-oriented architectures (SOA) deal with the growing need for open distributed applications capable of evolving and adapting continuously over their execution. To deal with the high dynamicity of the Web and the large variations in the QoS, the key proposal of this paper is to delay the choice of services to the run-time and to use up-to-date QoS information to perform this selection. To this end, usual service offers are enhanced with QoS commitments, but instead of dealing with this information statically, a loose coupling is implemented at run-time by the late-binding of abstract service requests (of Web processes) to concrete service offers. Central to this approach is the assumption that QoS commitments from the service providers express the QoS properties that users are entitled to monitor at run-time to judge the current performance of the service.

This paper focuses on the various steps involved in the implementation of this novel calling process. After filtering candidate services for each service call from registries upon their QoS commitments, a connection is established with their monitoring interfaces. During the whole execution of the business process, up-to-date QoS values are received asynchronously through a monitoring middleware. When the business process calls a service, it gathers all of the current QoS levels from all candidate services for that call and selects the one to be called upon these.

The dynamic selection of Web services then leverages user preferences to maximize the expected QoS from each call. In this multi-criteria decision making context, to make the best possible binding decisions between consumers and producers, preferences must be established among the various non-functional properties of required services. Given the subjective nature of these preferences, they are elicited by business process programmers before running them. This paper recalls the LCP-net formalism [2] to elicit and express user preferences in a business-oriented qualitative way.

The rest of the paper is organized as follows. Section II presents the conceptual background and related work. Section III describes the different concepts and programming abstractions adopted to implement the new kind of late-binding of service calls in business process languages. Section IV presents in more detail the LCP-net formalism, the cornerstone of the approach since it enables programmers to express their preferences in the service selection process, and its integration into the preceding programming abstractions. Section V describes the current implementation. A conclusion ends the paper, including a discussion of perspectives to this work.

II. Background and Related Work

The section briefly details the related work concerning QoS-awareness in SOA as well as preference modeling formalisms.

A. SOA and non-functional properties

SOA and Web services have recently gained broad industry acceptance. However, current standards do not meet the dynamicity of the Web, where services appear, disappear and exhibit large variations in their QoS even over a business process execution. To tackle these, QoS-awareness must be

Work partly funded by the French ANR project SemEUsE (07-TLOG-018).

built into the *run-time* SOA platform to dynamically select the best service available to fulfill each request.

Indeed, besides filtering services using non-functional constraints, most works on QoS-awareness have been done to compose web services so to fulfill QoS commitments of the business process [3], [4], [5], [6], [7]. However, this form of composition implies a selection of a service for each call, which is currently done at design- or load-time. In this paper, dynamically measured non-functional levels are used to further seek for the best offer just prior invoking the service; completing a static filtering of candidate services with the run-time selection of the one to be called among these.

B. Binding and binding times

First explored in the context of programming languages, binding times and late-binding relates to the conceptual framework which organizes the how and, more importantly, the when decisions are made to bind identifiers of entities (procedures, libraries, etc.) to their physical realizations to be used at run-time. Besides the well-known and explored late-binding of method calls in OOP, late-binding of remote calls in distributed programming is the ability to choose the remote application just prior to the call itself. Doing so provides not only for more agility, but also for more robustness in the context of failures. In this paper, we consider a new form of late-binding, where decisions are made upon the current QoS of providers. In SOA, this approach deals with the more competitive nature of the service paradigms where numerous equivalent services are competing in a highly dynamic environment.

In the context of SOA, work has been done to add dynamicity in service selection. Mosincat and Binder [8] implement a binding manager that runs in background to rebind failed services according to blackbox service selectors specifically developed to work with the binding manager. Scene [6] proposes similar ideas, but relies on user-defined rules to trigger rebindings. The rules include user preferences but, as other works [9], [4], do not address conflicting preferences among QoS properties. The originality of this paper proposal lies in the extension of the BPEL language allowing programmers to provide for dynamicity at the business process language level, instead of lower level libraries/middleware, and in the LCP-net formalism providing programmers with an intuitive tool to express their preferences among services through their different QoS properties monitored at run-time.

C. Fuzzy Linguistic Approach and *CP-nets

Indeed, a major issue when dealing with QoS is the large number of different dimensions (e.g. latency, precision, etc.). Because offers are rarely the best for every QoS dimension, preferences are needed to rank them given their relative strength on the different dimensions. Preference elicitation and expression have received attention in past years; several formalisms have been proposed. For SOA, a good formalism must obey several requirements, among which usability by non-specialists business process programmers, is of primary importance. To this end, we have proposed a new formalism [2] based on the combination of CP-nets and the fuzzy linguistic approach [10] to qualitatively specify user preferences over the different non-functional properties of offers.

Introduced by Zadeh [10], the linguistic variable concept represents the qualitative aspect of a value, e.g. *low*, *tall*, etc. Zadeh associates fuzzy sets to the variables, i.e. a temperature value can be expressed by a fuzzy set whose membership function is a mapping from [10°C,30°C] to [0,1]. Several other representation models have been proposed, such as the semantic [11], the symbolic [12] or the 2-tuple [13] ones.

Among the different models used to express the user preferences, we have chosen what we call the *CP-nets ("wildcard" Conditional Preference Networks) family. *CP-nets are graphical formalisms with several benefits such as ease of use for the preference modeler, relatively low computation cost, and easily extendable to support additional properties. In a CP-net, the main elements are the *nodes* representing the problem variables, the *arcs* denoting preferences among these variables for given values, and the conditional preference tables (*CPTs*) attached to nodes [14]. CPTs indicate the preferences depending on the other variable values (the linked nodes). CP-nets allow for the preference modeling of statements such as "I prefer the V_1 value for property X over V_2 if properties $Y = V_Y$ and $Z = V_Z$". They assume the *ceteris paribus* (all else being equal) property. The highest node in the graph has the highest preference. UCP-nets (Utility CP-nets) [15] replace the binary relationships between node values by utility factors. Thus, in this formalism, CPTs contain precise numerical values instead of binary order relations. As for the TCP-nets (Tradeoffs-enhanced CP-nets) [16], they implement the conditional relative preference: "A better assignment for X is more important than a better assignment for Y given that $Z = V_z$". New kinds of tables and arcs are introduced thereby managing these tradeoffs.

The modeling of non-functional properties by *CP-nets has been proposed in [17] and [18] but in the context of a static composition of services and without considering qualitative preferences nor continuous value domains.

III. Late-binding of service calls

While as late as possible service-process binding decisions are sought, care must be taken not to overly compromise performance. In a current QoS oriented decision, service discovery and QoS values gathering could be done as late as the service call-time, but this would result in very high costs (delays). A traditional approach to mitigate the effects of late-binding is to do preprocessing in order to avoid inconvenient delays at call-time. Preprocessing can be done as early as process design-time, or later on, at process load-time. In both cases, this is *static* since it takes place before process execution; be fully aware though that there is no sharp separation between *static* and *dynamic*, but rather a smooth transition between both ends of the spectrum, as shown in Figure 1. This whole spectrum can be used to tune the overall performance of the late-binding by keeping static the most time-consuming operations (filtering and data adaptation

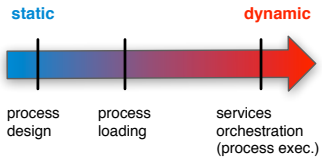


Fig. 1. From static to dynamic task execution.

generation) while making dynamic the necessary ones (QoS-based selection and binding of the service to be called).

While this new form of late-binding paves the way to more agile and more robust applications, we need to tackle issues raised by its implementation. In order to do so, we focus on the following sequential sub-steps (summarized in figure 2).

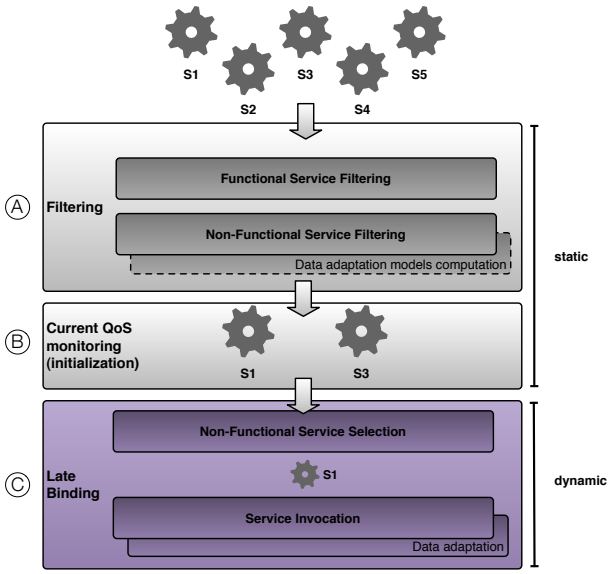


Fig. 2. Services funneled through late-binding conceptual layers.

A. Service filtering

Sets of candidate services are obtained by filtering them from the registry upon their functional and non-functional properties. Rather than waiting until call-time, this can be done for each call site either at process design- or load-time. Considering computing and network-related bottlenecks, such as the unavoidable delays to call the registry or the execution times of the service matchmaking algorithms, static filtering can minimize the impact of service discovery on the late-binding process as a whole.

Data-adaptation is also impacted. Indeed, Moreau et al. [19] have shown that design-time filtering provides for enough time to compute elaborate data-adaptation schemes between services requirements and selected offers sets. On the other hand, the load-time approach forces to use simpler schemes, but is particularly well suited when numerous functionally equivalent services are competing in a highly dynamic environment. In this paper, as a compromise, the setting up of the sets is postponed to process load-time where the pool of available services is more up-to-date. The counterpart is that data-adaptation adopts a more basic approach [20] to meet the performance constraints. Indeed, unless requiring a very long execution time, the business process should not wait much for data-adaptation schemes to be computed.

Returning to the service filtering process itself, strict non-functional requirements are injected to elect the final set of candidate services to monitor, rejecting at the same time all the non-conforming services upon the QoS offers they previously registered. Another result of this filtering is to consider that the information necessary for QoS monitoring setup is provided as an output of the selection process. More precisely, the current implementation uses negotiated SLA between processes and services to carry this information to the business process.

B. Current QoS monitoring

Central to our approach is the assumption that non-functional commitments from service offers express the QoS properties that users are entitled to monitor at run-time, through a specific service monitoring interface, in order to judge the current performance of the service. After service filtering, monitoring probes from these interfaces are identified based on the information put in the SLA negotiated between the business process and the candidate services. The goal is to subscribe to the monitoring interfaces of every candidate service for each call in the business process. Then, during the whole execution, up-to-date QoS values will be received, allowing for fast on-the-fly decision making.

As said earlier, QoS values could be tardily retrieved at service call-time, but then the service call would be delayed by the latency induced by the network for this remote query. Indeed, a push-mode is preferable for QoS value transmission. With carefully chosen publish frequencies, the business process can have immediately accessible “fresh” QoS values for all candidate services to enable a fast decision at service call-time. Optimizing this monitoring and providing for the required quality of information (precision, freshness, coherency, ...) is dealt with by a dedicated framework [21]

C. Non-functional service selection and invocation

At service call-time, five steps are required: QoS levels gathering for candidate services, service selection, data adaptation for the parameters, service invocation, and finally data adaptation of the result. QoS value gathering is done by calling the montoring middleware, which provides them without delay, thanks to the push-mode and asynchronous collection of these data discussed above. Data adaptation amounts to applying preprocessed transformations computed at service filtering time, while the service call itself use standard business process interpreters mechanisms.

Service selection requires to compare candidates services upon several different QoS dimensions which comparisons can contradict each other. As a multi-criteria decision problem, *it requires preferences*, tailored before process execution to its context, and applied to currently measured QoS values. These non-functional preferences, fundamental to our approach and which formalism is exposed in section IV, strive to maximize the expected QoS from each call by enabling the consumer to

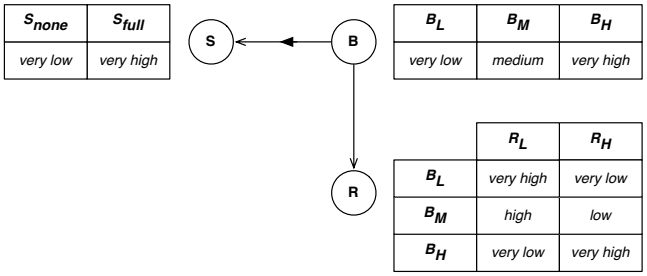


Fig. 3. The imaging service QoS preferences with LCP-nets.

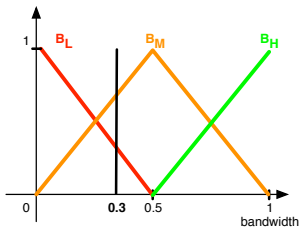


Fig. 4. Bandwidth QoS domain fuzzy partitioning.

better specify its needs in an machine-interpretable way. They are also very useful to deal with contradictory requirements since they are established *between* the various QoS properties: a total order over services can now be obtained at run-time, and the final binding decisions made.

IV. LCP-nets and user preferences

Selection upon non-functional properties is a multicriteria decision making problem which is better dealt with in a qualitative framework, as the LCP-nets presented in this section.

A. Preference elicitation with LCP-nets

LCP-nets [2] uses qualitative assessments to allow for the preference modeling of statements such as "I prefer the *more or less* V_1 value for X over *exactly* V_2 if Y is *approximately* V_Y and Z is *a bit more than* V_Z". LCP-nets allow users to express preferences, relative importance and tradeoffs among non functional properties quite easily with a graphical tool using linguistic terms.

Figure 3 shows an LCP-net for an imaging service (e.g. a security camera) where QoS properties are: security (S), bandwidth (B) and image resolution (R). As the overall goal is to get images as fast as possible, the user always prefers bandwidth over security, and if the bandwidth is low, prefers low-resolution images to get them as fast as possible. As this example shows, the elicitation of the preferred assignments for a specific QoS domain (or interdependent ones) is performed using CPTs and assigning weights to the nodes, depending on their position in the graph. Specifically, for each node, a CPT expresses the *qualitative utility* of QoS values for this dimension (e.g. *very high*, *low*, etc.), given the QoS values of graph-related dimensions which influence this node utilities.

The semantic of the linguistic terms used to constrict QoS values in a particular CPT is given by a prior fuzzy partitioning of the pertaining domains normalized over [0,1], as is also the one of the linguistic terms used to express utilities in the tables. Figure 4 shows the discretization for the bandwidth domain.

B. Service selection led by LCP-nets and current QoS levels

With preferences elicited and expressed into an LCP-net, several steps are required to evaluate the preference network at run-time. Considering a service with current QoS values over S, B and R, the focus is on the computation of its global utility by injecting these values into the LCP-net. With multiple services involved, the ultimate goal is indeed to select the service offering the best utility.

As a preprocessing step of LCP-net evaluation, CPTs are translated and mapped to fuzzy rules (e.g. in Figure 3: "if $R = R_L$ and $B = B_L$ then preference is *very high*"), then aggregated into corresponding Fuzzy Inference Systems (FIS) for each table. At service call-time, QoS values are injected in the evaluation process after normalization and fuzzification (e.g. a 0.3 singleton fuzzy subset for a crisp 30Kb/s bandwidth value, as shown in figure 4). For each preference table, the corresponding FIS allows for the computation of the local node utility. Utilities are obtained as crisp or fuzzy values.

Nodes have weights capturing their importance (position) in the graph, which are distributed (summing to 1) according to a decreasing function (see [2] for details). In the example, they are: *(B, 0.529)*, *(S, 0.235)* and *(R, 0.235)*. The global utility value is then the weighted aggregation of the local utility values. After applying this process to every considered service, the one offering the largest global utility is selected and passed-on to the late-binding framework for invocation.

C. A new feature: incremental definition of LCP-nets

Pragmatically, programmers often have similar preferences applying to several service call sites, with eventually slight changes to cope with the specifics of each call. Instead of repeating most of the preference network each time, a nice feature is to be able to factor common preferences into a shared base LCP-net, and to locally add the specific preferences. To this end, we introduce *LCP-net fragments*, constructs, built over an existing LCP-net, describing unambiguously *additions*, *deletions* (with *anti-fragments*) or *modifications* (with *modifier fragments*) to any base LCP-net element. The general form of a fragment follows the same pattern and graphical nature of a classic LCP-net construct, each base LCP-net construct (nodes, arcs and tables) having a corresponding derived fragment type.

Possible integrations of this new feature of the LCP-net formalism can be envisioned for BPEL, where LCP-nets and fragments could be attached to specific activities and their lexical scope exploited to link fragments to the base LCP-net. It should therefore optimize preference usage by reducing repetitions, the direct consequence being savings on development costs by fostering reusability across the board, as highlighted by the SOA philosophy.

To illustrate the incremental definition process, we compose the previous LCP-net $ImagingPref$ with a fragment $ImagingPref_{frag}$ to obtain a new LCP-net $NewImagingPref$.

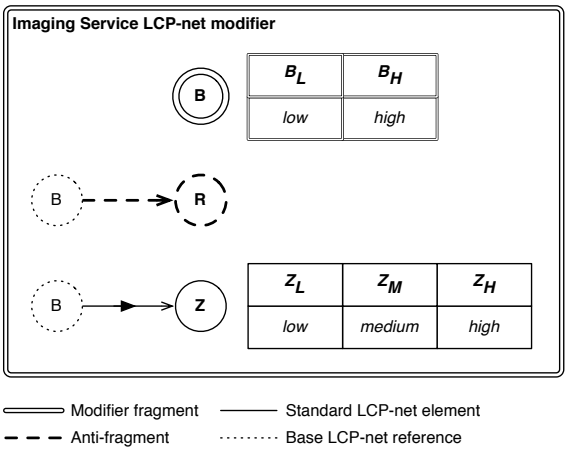


Fig. 5. Imaging service preference fragment.

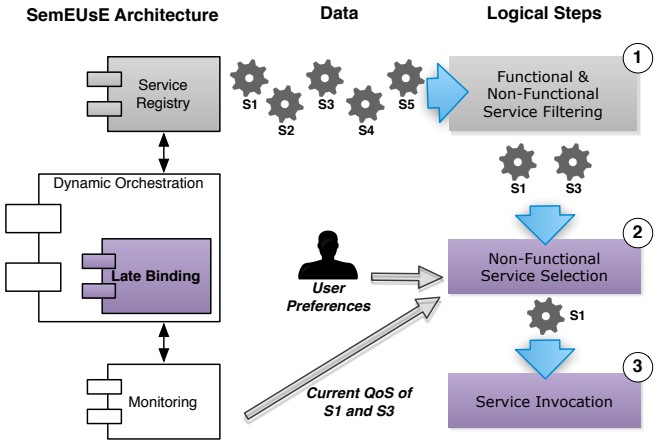


Fig. 7. SemEUsE architecture overview.

This fragment removes the preference on resolution (R) and replaces it by ones on a newly introduced zoom (Z) QoS property. It also changes the type of arc between B and Z, and the utility values from the bandwidth (B) CPT table. Finally, the *NewImagingPref* preference resulting from its application is given in its graphical form in Figure 6.

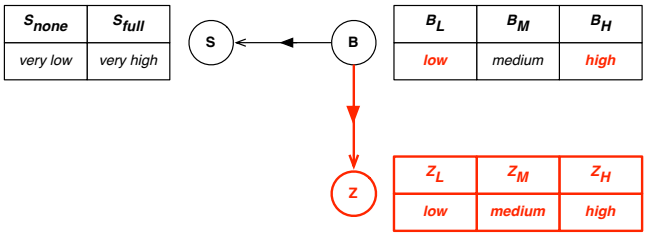


Fig. 6. Modified imaging service QoS preferences..

V. Implementation

QoS-based late-binding has been implemented in the three-year SemEUsE project funded by the French National Agency for Research. Started in 2008, it brings together several academic and industrial partners (Thales, Orange Labs, EBM Websourcing, LIP6, INSA, INRIA, INT) with the goal of fostering a "Semantic Middleware" that allows for the implementation of "pervasive, flexible and reliable applications".

The parallel between SemEUsE architecture and abstract concepts such as *service filtering*, *selection* and *invocation* is illustrated in Figure 7 through a simple abstract example. Multiple services from a registry, ranging from *S1* to *S5*, are funneled through each implemented logical step until one service (*S1*) selected upon its current QoS values and statically defined LCP-net preferences, is invoked. In SemEUsE, functional and non-functional service filtering is being taking care of by the service registry, while non-functional service selection and invocation is implemented by the late-Binding component alongside dynamic orchestration.

A. LCP-net preferences definition

The LCP-net model, including all its core constructs, has been defined using the Eclipse Modeling Framework (EMF), an Eclipse-based modeling and code generation facility for building tools based on a structured data model. Our particular model is an *ecore* file serialized under the XMI OMG standard, which should ensure its sustainability and provides the foundation for interoperability with other modeling tools.

This model specification produces a set of corresponding Java classes and a set of adapter classes that enable viewing and command-based editing of the model. As such, LCP-net can be constructed programmatically by an external software component. However, in our context, LCP-nets are manually defined before run-time. As a result, building over this set of Java classes, a basic graphical editor has been implemented to ease both preference elicitation and definition. Its GUI is reproduced in a compact way by figure 8: the *monitoring_preferences.lcpnet* file is editable as a tree, each constitutive element being selectable to edit its properties (e.g. name, linked elements, etc.) in a dedicated *Properties* panel. The next logical step will be to enable a "real" graphical

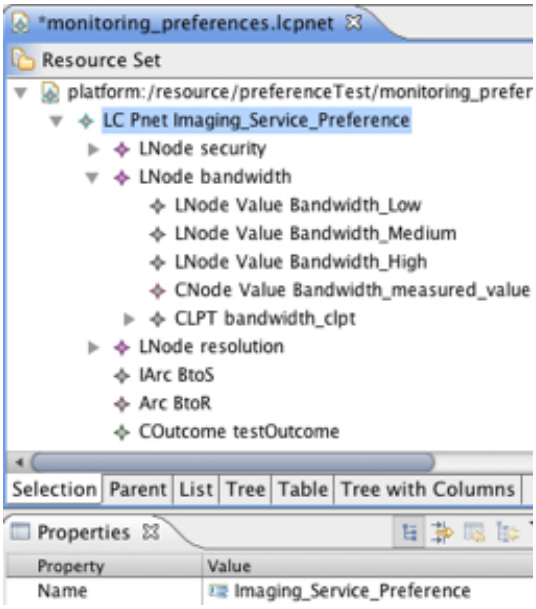


Fig. 8. LCP-net editor GUI.

representation for each core LCP-net construct in the editor, using the Graphical Modeling Framework (GMF).

Listing 1. Imaging service LCP-Net – XML excerpt

```
<LCPnet:LCPnet name="Imaging_Service" (...)>
 <nodes xsi:type="LCPnet:LNode" name="bandwidth"
     outArcs="(...)">
   <domain xsi:type="LCPnet:LNodeValue" name="
       Bandwidth_Low" linguisticValue="(...)"/>
   <domain xsi:type="LCPnet:LNodeValue" name="
       Bandwidth_Medium" linguisticValue="(...)"/>
   <domain xsi:type="LCPnet:LNodeValue" name="
       Bandwidth_High" linguisticValue="(...)"/>
   <domain xsi:type="LCPnet:CNodeValue" name="
       Bandwidth_measured_value" crispValue="1.0"/>
   <linguisticTable> ... </linguisticTable>
 </nodes>
 <nodes xsi:type="LCPnet:LNode" name="resolution"
     inArcs="(...)">
   <domain xsi:type="LCPnet:LNodeValue" name="
       Resolution_Low" linguisticValue="(...)"/>
   <domain xsi:type="LCPnet:LNodeValue" name="
       Resolution_High" linguisticValue="(...)"/>
   <domain xsi:type="LCPnet:CNodeValue" name="
       Resolution_measured_value" crispValue
       ="0.51"/>
   <linguisticTable> ... </linguisticTable>
 </nodes>
 <arcs xsi:type="LCPnet:IArc" name="BtoS" startNode
     ="//@nodes.1" endNode="//@nodes.0"/>
 <arcs name="BtoR" startNode="//@nodes.1" endNode
     ="//@nodes.2"/>
 <valueDomains name="Bandwidth">
   <subsets name="Bandwidth_Low">
     <fuzzySubset y="1.0"/>
     <fuzzySubset x="0.5"/>
   </subsets>
   (...)
 </valueDomains>
 (...)
</LCPnet:LCPnet>
```

In any event, there is no need from a preference modeler perspective to directly manipulate an XML document: serialization is handled by the editor and EMF support framework. Should this be required anyway later on, it is currently possible to directly write or modify a LCP-net instance at the XML level, by obeying the same fairly simple syntax used for automatic serialization: listing 1 provides an excerpt of the XML document for the imaging service LCP-net.

Listing 2. Imaging service LCP-net fragment – XML excerpt

```
<fragments:LCPnetFragment (...)
   fragmentName="imaging_service_fragment">
 <baseLCPnet href="imaging_service.lcpnet#/"/>
 <nodes xsi:type="fragments:LNodeFragment"
        name="" fragmentName="R_antifragment"
        antiFragment="true">
   <baseLNode href="monitoring_preferences.lcpnet#//
       @nodes.2"/>
 </nodes>
   <nodes xsi:type="LCPnet:LNode" name="Z" inArcs
       ="//@arcs.1" valueDomain="//@valueDomains.0">
   <domain xsi:type="LCPnet:LNodeValue"
       name="Zoom_Low" linguisticValue=(...)/>
   <domain xsi:type="LCPnet:LNodeValue"
       name="Zoom_Medium" linguisticValue=(...)/>
   <domain xsi:type="LCPnet:LNodeValue"
       name="Zoom_High" linguisticValue=(...)/>
   <linguisticTable name="zoom_cpt">
     (...)
   </linguisticTable>
 </nodes>
 <arcs xsi:type="LCPnet:IArc" name="BtoZ" startNode
     ="//@nodes.2" endNode="//@nodes.1"/>
 (...)
</fragments:LCPnetFragment>
```

The new fragment construct also benefits from an EMF specification and implementation. The same tools used for LCP-net elicitation are used for fragment definition. As such, an excerpt from the XML definition of the previously introduced $ImagingPref_{frag}$, as in section IV-C, is given in listing 2, putting an emphasis on:

- the initial import of the $imaging_service.lcpnet$ base LCP-net (that would be deduced automatically from the lexical scope in our envisioned BPEL integration);
- the deletion of the node R and its attached CPT, through a node *anti-fragment*.
- the introduction of a new Z node and $BtoZ$ arc by the definition of their fragments;
- the use of XPath expressions to refer to any elements from $imaging_service.lcpnet$.

Our LCP-nets framework is now available as a free download under the GPLv3 license at Google Code `http://code.google.com/p/lcp-nets/`.

B. Service filtering: from abstract to extended BPEL business process

In SemEUsE, a UDDI [22] like service registry has been implemented. It supports both functional and non-functional service requests at the syntactic and semantic levels. Filtering is driven by a matchmaking algorithm using the relationships between the various ontological concepts used to annotate the more common syntactic information (method or operation names, data types and QoS properties). OWL-DL [23] has been chosen to define the related ontologies since it gives some guarantees on the outcome of the reasoning process.

Functional service requests (and offers) are provided to the registry as SAWSDL [24] specifications, while non-functional ones are given by WS-agreements [25]. Final negotiated WS-agreements computed by the registry for each filtered service contain all the necessary information for monitoring probe deployment, and defines, inside an inline QML-based specification [26], all the QoS constraints fulfilled by the service.

The registry is queried, and services filtered, when abstract business processes are translated into *extended BPEL processes*. In SemEUsE, this translation is made at *abstract processes* load-time. In these abstract processes, service binding parts have been left out, since available services are not known at design-time. They are filled in with information provided by the registry, and then a runnable extended BPEL process is obtained and can be evaluated.

Listing 3. Resulting extended BPEL process excerpt

```
<process name="semeuse" (...) >
```

```
<!-- (...) Import client WSDL, define partner links
    , define variables -->
<sequence name="main">
  <extensionActivity>
    <semeuse:lateBindingConfigure invocationID="
        invocation1" preference="prefs.lcpnet">
      <semeuse:candidateServices>
        <semeuse:service EPR="service1" portType="
            service1PT" operation="getVideo"
            contract="wsag1.xml"/>
        <semeuse:service EPR="service3" portType="
            service3PT" operation="getVideo"
            contract="wsag3.xml"/>
      </semeuse:candidateServices>
    </semeuse:lateBindingConfigure>
  </extensionActivity>
  <extensionActivity>
    <semeuse:monitoring state="start" />
  </extensionActivity>
  <!-- (...) Receive input from requester. -->
  <extensionActivity>
    <semeuse:lateBindingInvoke invocationID="
        invocation1" inputVariable="invoke_in"
                        outputVariable="invoke_out"
         preference="prefs.lcpnet"/>
  </extensionActivity>
  <!-- (...) Generate reply to request -->
  <extensionActivity>
    <semeuse:monitoring state="stop" />
  </extensionActivity>
</sequence>
</process>
```

In this extended BPEL process, three new late-binding centric extended activities can be found: *lateBindingConfigure*, responsible for setting up monitoring; *monitoring*, for launching and controlling it; and finally *lateBindingInvoke*, for initiating service selection and invocation. The BPEL process excerpt given in listing 3 integrates these activities in the previous imaging service scenario. Since its is obtained after selection, only candidate services are mentioned (*service1* and *service3*, as in figure 7).

In SemEUsE, Orchestra has been chosen as the business process orchestration engine and Petals as its service bus. The engine, which provides an accomplished implementation of the BPEL extension mechanism, allows us to write three dedicated classes to process these required activities when encountered during process execution. These classes act as proxies to the LCP-net decision engine and monitoring framework: they retrieve the necessary configuration arguments, pass them on, and trigger the corresponding actions (monitoring configuration, launch, and service invocation) in these modules.

The Java classes are extensions of the abstract class `org.ow2.orchestra.definition.activity.AutomaticActivity` which itself inherits from `AbstractActivity`. This method allows for the implementation of advanced business logic alongside the usual BPEL workflow, with a certain level of visibility and interaction over the execution context made possible through a `org.ow2.orchestra.runtime.BpelExecution` variable.

C. Current QoS monitoring with the M4ABP middleware

Probes and monitoring interfaces are provided by an external middleware designed with adaptive business processes and decision making in mind: M4ABP (Monitoring for Adaptive Business Process) [21]. Configuration of M4ABP is made with the information contained in *lateBindingConfigure* activities, as can be seen in Listing 3, where needed technical (such as the End Point Reference, Port Type or operation name) and non-functional information (LCP-net preferences and WS-agreements URLs) is provided. At this point, the LCP-net preference file is only used to determine which QoS properties should be monitored; most of the related technical information needed for probe deployment is contained in the negociated WS-agreements (*wsag1.xml* and *wsag3.xml*). An excerpt of the second one is given in Listing 4.

Listing 4. WS-agreement excerpt for *service3*

```
<agreement:Agreement (...) >
 <name>wsag3</name><agreementId>id</agreementId>
 <context/>
 <terms><all>
  <serviceDescriptionTerm name="getVideo"
      serviceName="getVideo" />
  <serviceProperties name="getVideo" serviceName="
      getVideo">
   <variableSet>
    <variable name="resolution">
     <location xsi:type="semeuse:WSBinding"
      mode="pull" type="ws"
      epr="http://localhost:8080/services/Camera3"
      operation="getResolution" />
    </variable>
    <variable name="bandwidth">
     <location xsi:type="semeuse:WSBinding"
      mode="pull" type="ws"
      epr="http://localhost:8080/services/Camera3"
      operation="getBandwidth" />
    </variable>
    (...)
   </variableSet>
  </serviceProperties>
 </all></terms>
</agreement:Agreement>
```

Following directly, monitoring of services is launched by the *monitoring* extended activity with the *start* argument. Since both the *lateBindingConfigure* and *monitoring* activities are placed at the very beginning of the BPEL process, current QoS monitoring effectively starts before the evaluation of the BPEL process "business" body, at load-time.

D. Late-binding in Orchestra

The *lateBindingInvoke* activity implements the run-time service selection over candidate service sets, using the current QoS values being provided by M4ABP and the LCP-net specified preferences. Indeed, the binding decisions between processes and services are made upon service utilities through the LCP-net evaluation process explained in section IV.

The LCP-net engine, which is the component responsible for outputting the global utility for each considered service, has also logically been implemented in Java, to maximize its integration with the whole framework. At its core, the jFuzzyLogic library processes QoS levels inputs through computed Fuzzy Inference Systems in order to obtain local utilities for each monitored QoS dimensions. Indeed, it implements the

standardized Fuzzy control language (FCL) specification (EC 61131-7) under which our FISs are serialized.

After selection, the remote invocation of the bound service is passed on to the underlying Orchestra and Petals support infrastructure, leaving only data adaptation concerns on services inputs and outputs to be solved. Since service filtering has been made at process load-time, the necessary data adaptation schemes have also been computed at the same time, leaving only their application at invocation-time. As indicated earlier, generating data adaptation at load-time forces to use simpler matching schemes than the ones used offline, which are too computationally involving. The objection that fewer services can then be matched is largely offset by the vast offering of functionally-equivalent services promoted by Semantic SOAs giving more choices for matching.

VI. Conclusion

In this paper, a new programming abstraction, QoS-based late-binding of service invocations, has been proposed to provide for more agility to business process execution. Based on current QoS values of a set of services and user preferences expressed in a qualitative way, the service selection elects and calls the current best offer among equivalent concrete candidate services capable of answering a call to an abstract service functionality. A new formalism, called LCP-nets [2], is used to allow non-specialist programmers to qualitatively express their preferences among values of the different QoS properties in this multi-criteria decision making process.

This new programming abstraction has been implemented as BPEL extended activities for the Orchestra engine. These extended activities use an implementation of LCP-nets in EMF providing for both the creation of LCP-nets by programmers and their use at run-time in the decision making. QoS data are gathered by a monitoring middleware [21], which organizes and optimizes the regular flow of data from services to the business process and provides for their buffering for immediate access and gathering when the late-binding decisions must be made upon them. This data mediation layer also provides some means to ensure the temporal coherence of the monitoring data bundled for each late-binding decision, a key dimension in the quality of information for such middleware.

Perspectives and future work are numerous. More experiments and performance measurements are needed to fine tune the current implementation. Currently, the incremental definition of LCP-nets is limited to two levels, and it is implemented in a statically scoped way in BPEL. Allowing more levels and providing them as first-class entities would provide for even more flexibility to programmers in order to share common preferences among decision sites in complex business processes. The monitoring shall be extended to cope with more quality of information properties for monitoring data, such as its freshness, its precision, etc.

References

[1] M. Halpern, "Binding," in *Encyclopedia of Computer Science*, 3rd ed., A. Ralston and E. Reilly, Eds. Chapman & Hall, 1993, p. 125.

[2] P. Châtel, I. Truck, and J. Malenfant, "A linguistic approach for non-functional preferences in a semantic SOA environment," in *Proc. of the 8th Int. FLINS Conf.*, 2008.

[3] L. Zeng *et al.*, "QoS-aware middleware for web services composition," *IEEE Trans. Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[4] G. Canfora *et al.*, "Service composition (re) binding driven by application-specific qos," *LNCS*, vol. 4294, p. 141, 2006.

[5] G. Chafle *et al.*, "Adaptation in web service composition and execution," in *Int. Conf. on Web Services*, 2006.

[6] M. Colombo, E. D. Nitto, and M. Mauri, "SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules," in *ICSOC'06*, ser. LNCS, no. 4294. Springer-Verlag, 2006, pp. 191–202.

[7] D. Chiu, S. Deshpande, G. Agrawal, and R. Li, "A Dynamic Approach toward QoS-Aware Service Workflow Composition," in *Int. Conf. on Web Services*. IEEE Computer Society, 2009, pp. 655–662.

[8] A. Mosincat and W. Binder, "Transparent Runtime Adaptability for BPEL Processes," in *Service-Oriented Computing - ICSOC'08*, ser. LNCS, no. 5364. Springer-Verlag, 2008, pp. 241–255.

[9] J. Siljee, I. Bosloper, J. Nijhuis, and D. Hammer, "DySOA: Making Service Systems Self-adaptive," in *ICSOC'05*, ser. LNCS, no. 3826. Springer-Verlag, 2005, pp. 255–268.

[10] L. Zadeh, "The Concept of a Linguistic Variable and Its Applications to Approximate Reasoning," *Inf. Sci., Part I, II, III*, vol. 8,8,9, pp. 199–249, 301–357, 43–80, 1975.

[11] R. Degani and G. Bortolan, "The Problem of Linguistic Approximation in Clinical Decision Making," *International Journal of Approximate Reasoning*, vol. 2, pp. 143–162, 1988.

[12] I. Truck and H. Akdag, "A Tool for Aggregation with Words," *Inf. Sci., Special Issue: "Linguistic Decision Making: Tools and Applications"*, vol. 179, no. 14, pp. 2317–2324, 2009.

[13] F. Herrera and L. Martínez, "A 2-tuple fuzzy linguistic representation model for computing with words," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 6, pp. 746–752, 2000.

[14] C. Boutilier *et al.*, "CP-nets: A tool for representing and reasoning with conditional *Ceteris Paribus* Preference Statements," *J. of Artificial Intelligence Research*, vol. 21, pp. 135–191, 2004.

[15] C. Boutilier, F. Bacchus, and R. I. Brafman, "UCP-Networks: A directed graphical representation of conditional utilities," in *Proc. of the 17th Conf. on Uncertainty in Artificial Intelligence*, 2001, pp. 56–64.

[16] R. I. Brafman and C. Domshlak, "Introducing variable importance tradeoffs into CP-nets," in *Proc. of the 18th Conf. on Uncertainty in Artificial Intelligence*, 2002, pp. 69–76.

[17] C. Schröpfer *et al.*, "Introducing preferences over NFPs into service selection in SOA," in *Proc. Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC'07)*, 2007.

[18] G. R. Santhanam, S. Basu, and V. Honavar, "TCP - Compose* - A TCP-Net Based Algorithm for Efficient Composition of Web Services Using Qualitative Preferences," in *ICSOC'08*, ser. LNCS, no. 5364. Springer-Verlag, 2008, pp. 453–467.

[19] A. Moreau, J. Malenfant, and M. Dao, "Data Flow Repair in Web Service Orchestration at Runtime," in *4th Int. Conf. on Internet and Web Applications and Services, ICIW 2009*. IEEE Computer Society Press, 2009, pp. 43–48.

[20] P. Châtel, "Toward a Semantic Web service discovery and dynamic orchestration based on the formal specification of functional domain knowledge," in *Proc. of the 20th Int. Conf. on Software & Systems Engineering and their Applications*, 2007.

[21] B. Le Duc *et al.*, "Non-functional Data Collection for Adaptive Business Process and Decision Making," in *Proc. of MW4SOC'09 Workshop*, 2009, pp. 7–12.

[22] F. Curbera *et al.*, "Unraveling the Web Services Web: an introduction to SOAP, WSDL, and UDDI," *IEEE Internet computing*, pp. 86–93, 2002.

[23] D. McGuinness *et al.*, "OWL web ontology language overview," *W3C recommendation*, vol. 10, pp. 2004–03, 2004.

[24] J. Kopeckỳ, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: Semantic annotations for WSDL and XML schema," *IEEE Internet Computing*, vol. 11, no. 6, pp. 60–67, 2007.

[25] A. Andrieux *et al.*, "Web services agreement specification (WS-Agreement)," in *Global Grid Forum*, 2004.

[26] S. Frølund and J. Koistinen, "QML: A Language for Quality of Service Specification," Software Technology Laboratory, Hewlett-Packard, Tech. Rep. HPL-98-10, 1998.