Towards a Smart Webservice Marketplace

Ralph Vigne
CERN
CH-1211 Genve 23
Switzerland
Email: ralph.vigne@cern.ch

Werner Mach
Faculty of Computer Science
University of Vienna
Vienna, Austria, A-1090 Währingerstr. 29
Email: werner.mach@univie.ac.at

Erich Schikuta
Faculty of Computer Science
University of Vienna
Vienna, Austria, A-1090 Währingerstr. 29
Email: erich.schikuta@univie.ac.at

***Abstract*—Electronic contracts are crucial for future e-Business models due to the increasing importance of webservices and the cloud as a reliable commodity enabling service-based value chains. Negotiation is the prerequisite for establishing a contract between two or more partners. These contracts are usually based on Service Level Agreements (SLAs). In this paper we present the framework of a smart webservice marketplace, which allows for automatic, autonomous, and adaptive negotiation and re-negotiation of webservices based on economic principles. Our approach enables market based service trading following a bazaar style and extends the classical supermarket approach typical for service negotiation today. We extend the WS-Agreement standard by feasible workflows to support auctioning for negotiation and re-negotiation. A specific highlight of our framework is the mapping of business strategies defined by economic goals of the respective organization into an ICT enabled framework. It facilitates autonomic agents acting as organizational representatives stipulating SLAs without human interaction. This allows for business transactions transparently to the environment but adhering to business objectives of the originating organization.**

I. Introduction

Service oriented utility computing paves the way towards realization of service markets, which promise metered services through negotiable Service Level Agreements (SLA). A market does not necessarily imply a simple provider-consumer relationship, rather it is the culmination point of a complex chain of stake-holders with integration of value along each link in a service value chain. In such chains, services corresponding to different partners are aggregated in a producer-consumer manner resulting in added value.

We believe that in current SLA research negotiation between provider and consumer is only insufficiently considered. In most cases *one-phase negotiations* are being used to keep the effort for the negotiation as small as possible. One-phase negotiation means that service providers offer their services in form of agreement templates which the consumer can either accept or reject. Although this templates may contain a number of alternative service descriptions with different service qualities, the consumer can only choose one template which fulfills its requirements best. After choosing the offered template consumer and provider create an agreement. This approach is like buying in a supermarket: the provider offers a set of products and the consumer chooses one or more of it.

Open Grid Forum has defined an extension to the WS-Agreement Specification [1] named the **WS-Agreement Negotiation** version 1.0 [2] supporting multi-round negotiations [3]. Although the WS-Agreement Negotiation protocol does not explicit support auctioning and biding, it still provides an applicable protocol for communication.

Today, the range of information technology supported negotiations are wide spread, starting from simple communication structures, supporting only electronic messaging, up to fully evolved negotiation support systems [4]. Current research [5], [6] shows a special interest in self-interacting software agents enabling fully automated negotiations.

Negotiations can be divided in two categories: (1) single-attribute negotiating and (2) multi-attribute negotiating. In case of a single-attribute negotiating only one attribute, e.g. the price, is a negotiable part of a contract. While this may be sufficient for people's every day business, todays electronic commerce has become more complex. Usually, negotiations in the context of electronic commerce cover multi-attribute contracts. This means that for example not only the price of a resource is addressed by the negotiation but also its availability and probability of failure are negotiable. While single-attribute negotiations are already extensively investigated, multi-attribute negotiations are still at an early stage [7].

In our approach we aim to provide a single- and multi-attribute negotiation system where the participating agents do only know about their own preference/utility function but have no initial knowledge about the preferences/utility functions of the other participants. Based on this characteristics we present a webservice marketplace, where negotiation is performed in a fully automatic way in a Bazaar style form opposed to the conventional supermarket form.

In the next section we identify the fundamental questions for delivering the envisioned marketplace. In section III we provide detailed answers and describe the necessary components of our framework. This is followed by a justifying example applying our methodology. The paper is closed by conclusions and description of future work.

II. Towards a Service Bazaar

Existing frameworks for self-governing ICT infrastructures use a knowledge base for their decisions during operation [8], following the concept of autonomic managers which comprises four states: Monitor, Analyze, Plan and Execute (MAPE). Self-governing principles augment the autonomic systems. In autonomic systems the rules and policies are defined by humans whereas self-governing systems may produce, improve, and evolve the rules without intervention from outside. Due to dynamics of infrastructure changes (e.g. frequent service failures) the rules for QoS re-negotiation have to evolve reactively (e.g. new negotiation strategy have to be used). This has to happen without human interaction and has to be based on predefined guidelines.

Our proposed webservice marketplace establishes a self-governing infrastructure, which allows for automatic, adaptive and autonomous negotiation processes. To realize this vision we have to provide answers to the following questions:

- How can services be selected and service chains instantiated from a set of existing and registered services?
  We need a **Service Template Registry**, a contract aware marketplace, which abstracts from the heterogeneous offers of different services providers and allows for the selection/filtering and instantiation of services (see section III-A).
- How can a negotiation process between service providers and requesters be realized, which allows for multi-round negotiations?
  We need a self governing, generic **Negotiation Framework**, which allow for multi-round negotiation for consumer-provider contracting by auctioning processes (see section III-B).
- On what basis have decisions on service negotiations be made?
  We need a **Utility Knowledge Base**, which provides on the one hand all information necessary to calculate an over-all utility function based on economic principles, and on the other hand specialized information about each service invocation within distinct domains (see section III-C).
- How can a specific business strategy be implemented?
  We need autonomous **Business-aware Process Agents**, which are capable to cope with insufficient and changing information and make autonomously business decisions using AI technology, e.g. Blackboards, based on specific business strategies during the negotiation process (see section III-D).

Thus, in the following section we will provide detailed answers to the stated questions above.

III. Components of a Smart Webservice Marketplace

A. Service Template Registry

The service template registry allows consumer agents to *search* for services of a certain domain and provider to *advertise* their services to participate. It therefore acts as a marketplace supporting agents in doing business together (*business enabler*). To enable this functionality it must present knowledge about **what** services are offered and **how** they can be discovered, bought and used.

Various implementations of web service registries/repositories have been addressed for some time in research publications (e.g. [9], [10], [11], [12]). Although they differ in various aspects, they share the same goal, which is to **provide an open and flexible marketplace for providers and consumers supporting automated real-time service discovery and invocation on a pay-per-use basis.** One of the main differences between them is what information is presented. This ranges from semantic annotations (e.g. OWL-S) in a service repository (e.g. UDDI) to mash-up and service composition techniques (e.g. BPEL).

In [13] we presented a feasible ***Service Template Registry*** by introducing so called *domain-level operations* for unified interface and protocol description and *instance-level operations* for vendor specific interfaces and protocols. This multi-layered architecture aims to preserve the flexibility for all participants to keep their own interface implementations. In its latest version [14] WSAG support for unified interfaces was added. Protocols are provided in form of microflows representing in which order services must be called to achieve the desired functionality. To support the concept of unified interfaces, these microflows may further include information about transformations based on the unified interfaces to meet specific interfaces (e.g. parameter split/join/conversion, ...). Further important characteristics of this marketplace are:

- **Use Cases Based:** Participants can easily discover exposed functionality not considering specific technical details. A use case is defined by the interaction protocol to invoke a certain service functionality.
- **Unified Invocation/Semantics:** Services within the same market domain are interchangeable. This is a key prerequisite to allow providers to compete and consumers to reason and therefore creating a market.
- **Passive and Scalable:** Making the marketplace passive allows for easy distribution over several nodes within a cloud environment which further compliments scalability demands.

We envision our Service Template Registry to provide all information needed to allow autonomous agents, consumer and provider to interact with each other in potentially complex business transactions. Therefore two different kinds of knowledge must be presented. One represents general knowledge about the market itself while the second one focuses on each provider individually.

1) Market Knowledge: First, knowledge about the market itself is needed. What the commonly agreed protocols and interfaces are, is focused by this group. For example, when an actor wants to participate in a certain domain of the market, it first needs to know how business in general is done there and what information is exchanged e.g.

- **By which protocol participants do compete?** E.g. supermarket approach, dutch auction, sealed-bid, two sided auction, ...
- **In which units trade is done?** E.g. currency, time units, data units, ...
- **How are common/unified interfaces described and what are their semantics?** E.g. messages from other participants like WSAG Offers/Templates [15] or service usage requests, bidding requests from an auctioneer and information necessary to place bids, ...

We consider this kind of interaction as being **many-to-many** as it represents basic rules each participant has to agree on. Providing generic information without vendor specific details is also the purpose of *domain-level operations* described in [13] which will therefore be used to express such knowledge.

To make our framework work, each offered operation is represented by a tuple of three related domain-level operations[1]:

- **Service Negotiation:** Represents the unified interface and protocol to negotiate with all participants. If an agent follows this protocol and provides all information defined in the unified interface (i.e. WSAG Template) it can relay on that each available provider will participate.
- **Service Usage/Invocation:** Represents the actual service invocation without any vendor specific details. It should be noted that we see **payment** as part of the service invocation because it is not obvious when it must be done and therefore must be defined explicit. If an agent implements this interface and protocol it can relay on that it can use the requested service independently of the chosen provider.
- **Service Observation:** Represents how the execution of an invoked service can be observed and what data can be expected to be received. This will be used to ensure that agreed SLAs are respected during the invocation. If an agent implements this interface and protocol it can relay on that it is able to observe the execution of the invoked service independent of the chosen service provider.

2) Service Knowledge: The **second** group of provided knowledge focuses on direct interaction with each provider individually. As this information is only relevant in **one-to-one** interactions it is sufficient if only the two interacting parties agree on it. For example, if two participants want to do business with each other (i.e. a consumer wants to invoke a specific service method), additional specific knowledge about the business partner is needed e.g.

- **Which protocol must be followed for which interaction?** E.g. invoke a specific service method, bidding request, execution observation, ...
- **How to interact on a technical level?** E.g. service endpoints, interface transformations (based on the unified interfaces) and descriptions (e.g. WSDL [16]), ...

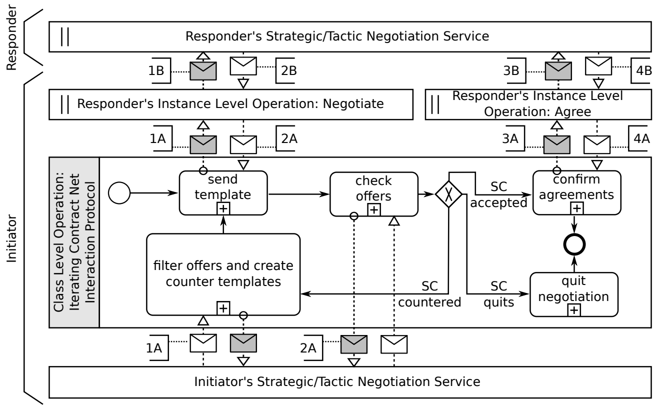
Providing this kind of information about each participant individually is exactly the purpose of *instance-level operations* as introduced in [13].

As imposed by the marketplace, each domain-level operation consists of at least one related instance-level operation. In our case this means that at least the same three methods defined as domain-level operations must be present on instance level, only this time their focus is on the protocols and interfaces obliged by each provider individually. As this is only relevant if two specific participants want to interact with each other (one-to-one interaction), it is on their own concern if they agree on each others interfaces (e.g. does the service support SSL encryption?, ...) and protocols (e.g. the supermarket approach is the only supported negotiation protocol, ...) or not.

B. Negotiation Framework

The negotiation and re-negotiation Engine is a central component of our framework. This engine is responsible for the whole life-cycle of a Service Level Agreement. We designed the negotiation engine as an autonomic manager [8].

1) N:M Negotiation Pattern: In various publications, this pattern is also referred to as *(Multilateral) Competing Offers Protocol* [17], [18]. In this pattern two different roles are identified: (1) the **Initiator** and (2) multiple **Responders**. Although the example (illustrated in Figure 1) is illustrated from the perspective of a single active initiator interacting with multiple responders, a single responder can still simultaneously participate in multiple negotiation processes at the same time. Therefore the pattern covers M:N negotiation as well as 1:N negotiation.


Responder
Responder's Strategic/Tactic Negotiation Service
1B
2B
3B
4B
Responder's Instance Level Operation: Negotiate
Responder's Instance Level Operation: Agree
1A
2A
3A
4A
Initiator
Class Level Operation: Iterating Contract Net Interaction Protocol
send template
check offers
SC accepted
confirm agreements
filter offers and create counter templates
SC countered
SC quits
quit negotiation
1A
2A
Initiator's Strategic/Tactic Negotiation Service

1A	WSAG Negotiation (Counter) Template	1B	(Specific) Negotiation (Counter) Template
2A	WSAG Negotiation (Counter) Offer	2B	(Specific) Negotiation (Counter) Offer
3A	WSAG Negotiated Offer	3B	(Specific) Negotiated Offer
4A	WSAG Agreement	4B	(Specific) Agreement

Fig. 1. BPMN: N:M Negotiation

It is on behalf of the initiator to provide the functionality and infrastructure to simultaneously negotiate with all included responders, registered in the *Service Template Registry*, on-demand (activity *send template*). As indicated by the *Multiple Instances Symbol* (||), defined in BPMN [19], each responder provides one *instance-level operation: Negotiate* describing how to interact with its *Strategic/Tactic Negotiation Service* when negotiating and one *instance-level operation: Agree* describing how to request a final agreement when an *acceptable WSAG Agreement* was negotiated. Both *Strategic/Tactic Negotiation Services* implement the provided *Business Rules* of their respective owner. To make this protocol work, responders should (not must![2]) further be able to create *counter offers* (2B) based on *templates* (1B). Every time the initiator **advertises**

[1] For a detailed discussion about further possibilities of domain-level methods e.g. reusing methods for complex service composition see [13]

[2] For example, responders who do not support re-negotiation can still participate by treating each request like it was the first, but if every responder does like this: Why re-negotiate in the first place?

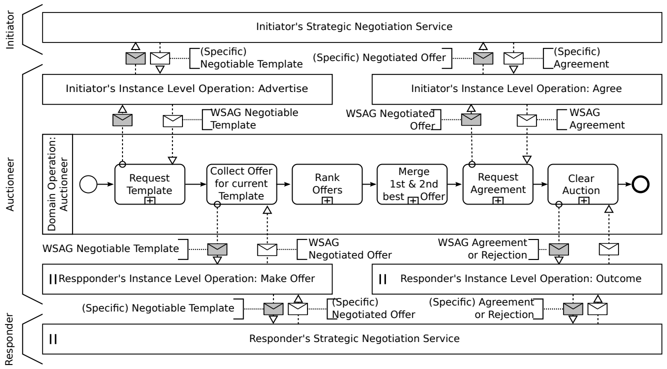

Initiator
Initiator's Strategic Negotiation Service
(Specific) Negotiable Template
(Specific) Negotiated Offer
(Specific) Agreement
Auctioneer
Initiator's Instance Level Operation: Advertise
Initiator's Instance Level Operation: Agree
WSAG Negotiable Template
WSAG Negotiated Offer
WSAG Agreement
Domain Operation: Auctioneer
Request Template
Collect Offer for current Template
Rank Offers
Merge 1st & 2nd best Offer
Request Agreement
Clear Auction
WSAG Negotiable Template
WSAG Negotiated Offer
WSAG Agreement or Rejection
Respponder's Instance Level Operation: Make Offer
Responder's Instance Level Operation: Outcome
(Specific) Negotiable Template
(Specific) Negotiated Offer
(Specific) Agreement or Rejection
Responder
Responder's Strategic Negotiation Service

Fig. 2. BPMN: Vickrey Auction

a new *template* (1A), each responder is entitled to respond either that it rejects further negotiations based on this *template* or a *binding negotiated offer* (3A). This is performed in the respective *Responder's Strategic/tactic Negotiation Services*. After the initiator has checked all offers, it decides if it is willing to accept one of them.

If it decides to **reject all of the received offers**, it filters the offers for responders to be left out of the next negotiation iteration and creates new *counter templates* (1A) for the remaining ones. Similar to responders, the rules for accepting or rejecting an offer, and how to create *counter templates* (1A) based on the received *counter offers* (2A) are applied by the *Initiator's Strategic/Tactic Negotiation Service* and defined in the provided *Business Rules*.

If the initiator **accepts an offer**, it sends it back to the awarded responder in expectation to receive an *agreement* (4A) as response, which is binding for both parties. Based on the content of this *agreement*, the *service usage* and *service observation* processes (see Section III-A1) can be initiated.

2) Auctioning Pattern: In this pattern not only an initiator and multiple responders are involved, but also one or more auctioneers are included in order to establish an agreement. The auctioneer is considered as an detached authority which is in charge to execute the defined protocol. Every responder is able to check the protocol in advance, as it is provided inside the *Service Template Registry*. Because the protocol also includes information about how the offers will be ranked (i.e. which parts of the offer will be weighted with what factor), the procurement during an auction is transparent for responders (even if it is a *sealed bid* auction). It should be further noted that the ranking algorithm is intended to be the same for each instance of this auction protocol and therefor expresses domain conventions instead of personal preferences. While this additional transparency allows responders to customize their offers in order to the defined ranking algorithm, the initiator looses the flexibility to express its specific preferences. If an initiator wants to apply its own ranking algorithm, the M:N negotiation pattern described in III-B1 provides this possibility.

Figure 2 depicts the BPMN diagram of a *Vickrey Auction* [20]. In this type of auction the responder with the best bid wins the auction, but it pays only the price of the second best bid. The initiator must accept the final offer (*WSAG Negotiated Offer*) presented by the auctioneer.

In the auctioning based negotiation process four roles are involved. The auctioneer is responsible for the whole process including, defining, and controlling the auctioning rules, collecting all offers and bids, executing clear actions, etc. as defined in the according *class-level operation*.

In Figure 2 we assume that the initiator has registered itself for an auction of this domain as well as all interested responders. The registering can be implemented in various ways and is therefore beyond the scope of this example.

In our example the negotiation process begins as soon as an auctioneer requests the initiator's *WSAG Negotiable Template*. This *WSAG Negotiable Template* is now advertised by the auctioneer to all responders who claimed interest in the described kind of service before. Next, each responder checks if the current offer meets its expectations and applies its defined *Business Rules* (defined inside its private *Responder's Strategic Negotiation Service*) to derive the price it is willing to pay for the offered service[3]. A *WSAG Negotiated Offer*, defining the offered price, is created and advertised to the auctioneer. After the auctioneer has collected all offers from the responders (or a timeout was reached), the auctioneer compares and ranks the distinct offers by its price. Because the auctioneer executes a Vickrey auction, now in the offer with the highest price, the price offered by the second highest bid, is inserted. The resulting offer is now sent back to the initiator in expectation of an *WSAG Agreement* in return. As soon as the agreement is responded, the auctioneer clears the auction by sending the agreement to the winning responder and a reject message to all the others.

C. Utility Knowledge Base

We group our knowledge base into two major groups: (1) the Process Region providing all information necessary to execute the over-all user process, and (2) multiple Domain Regions providing specialized information about each service invocation within distinct domains.

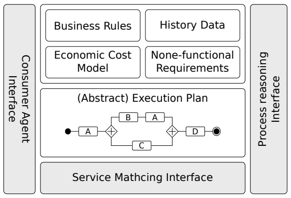

Consumer Agent Interface
Business Rules
History Data
Economic Cost Model
None-functional Requirements
(Abstract) Execution Plan
A
B
A
C
D
Service Mathcing Interface
Process reasoning Interface

Fig. 3. The Process Region

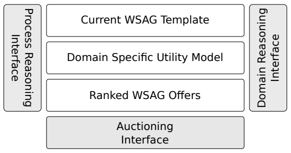

Process Reasoning Interface
Current WSAG Template
Domain Specific Utility Model
Ranked WSAG Offers
Auctioning Interface
Domain Reasoning Interface

Fig. 4. The Domain Region

1) The Process Region: The first region we will discuss is the Process Region (see Fig. 3). Its main purpose is to provide the utility function of the process. [8] identified the following three subregions as being crucial for such systems:

The **Business Rules Repository** is where the participant (consumer or provider) stores its own business rules (representing the specific business strategy). To allow agents to apply this business rules while negotiating, it is crucial that the parameters defined inside these business rules match with

[3] For the sake of simplicity, we assume that only the price-attribute (i.e. single-attribute negotiation) in the WSAG template is marked as being negotiable within a given range.

parameters defined in the according interfaces of the *Service Template Registry*.

The **Economic Cost Model** includes costs of each production factor (e.g. computational power, disk space, network bandwidth, ...). The cost model distinguishes between fixed costs and variable costs. This is necessary to reason about complex situations e.g.: A provider has free capacity of a resource and a customer already running a service is paying the fixed cost for it. Now the provider must decide at which price the free resource can be offered.

The **History Data** represents the experience an actor has collected so far while participating in the market. For example, a consumer agent may store data collected during negotiation and observation for each provider individually. This data will further be used whenever the agent needs to choose between different offers from vendors.

In order to cover as much aspects of service negotiation as possible we hereby extend our original framework [8] by more subregions.

The **None-functional Requirements**, although similar to the business rules but not quite the same, represent additional constraints for the negotiation process. This group contains constraints related to none-functional aspects of an negotiation like legal rules (e.g. only storage hosted inside the US is allowed) or ethical rules (e.g. like only companies which are member of the Fair Trade Association are considered).

As we will discus in Section III-D2 we also extend the framework to make it process aware.

The **Execution Plan** represents the over-all business process of the user. It is later used to encapsulate distinct service invocations in the process and initiate the according negotiations. Relaying on de-facto standards like e.g. BPEL/WS allows for high re-usability of agents interacting with this subregion and also for wide support on the consumer and provider side as most of them may have either implemented them already or can relay on profound tool sets supporting them.

It should be noted that all four aspects of the knowledge-base are considered to be private for each agent[4]. In our framework they must therefore be provided each time a new instance of a *Consumer Agent* is created.

In our framework this region offers a **Consumer Agent Interface** to access all four described subregions. The **Service Matching Interface** is only offered by the *Execution Plan* subregion and is intended to allow the identification of *Service Template Registry* domains (see Section III-A) of the various service calls. Last is the **Process Reasoning Interface** which is intended to allow all necessary operations to reason about service selection and also to observe the state of the various service invocations and the over-all process. It therefore includes access to all four subregions. A detailed discussion about this interfaces and their intended use is given in Section III-D2.

2) The Domain Region(s): Second type of regions are **Domain Regions** (see Fig. 4). Each of these regions represents one distinct domain of the *Service Template Registry* referred in the *Execution Plan*. Their intention is to provide a space for detailed knowledge about a single domain provided by domain experts and therefore acting as knowledge source for the *Process Reasoner*. Each of this regions consists of three subregions:

Current WSAG Template is the area where the Process Reasoner states what service (specific attributes like storage space, computing power, ...) it is looking for. In case of **multilateral iterated protocols** it is further used to represent the baseline for the next negotiation iteration.

Domain Specific Utility Model is a specialized form of the information provided in the *Business Rules*, *Economic Cost Model*, and *History Data* from the process region. The modell will be applied to rank the WSAG offers in order of their *Utility Value* for the user and to create *Counter WSAG Templates* during the negotiation process.

Ranked WSAG Offers contains all currently available WSAG offers of all participating providers. The offers are ranked according to the utility function expressed in the *Domain Specific Utility Model*.

Again three interfaces are offered by this type of regions. First is the **Process Reasoning Interface** as described above. It is used to set up and update the information state to always be in sync with the domain expectations of the *Process Reasoner* and to check out the currently available offers for a domain. Second is the **Auctioning Interface** which is used and managed by the *Auctioneer Agent* to enable *Provider Agents* to place their offers. Finally there is the **Domain Reasoning Interface**. As the name suggests, the interface is used by the *Domain Reasoner* to apply the *Domain Specific Utility Model* and rank the offers collected by the *Auctioneer Agent*. Again, a detailed discussion about the related agents is provided in Section III-D2

D. Business-aware Process Agents

In distributed, service-oriented systems, it is a crucial task to efficiently select and compose services required to respond to a given request.

When constructing a concrete service value chain, each service of the abstract chain has to be instantiated. This has to be done in a way that is optimizing (seeking for a minimum or a maximum) a custom **utility function** – the exact objective functions depend on the specific problem. In the focus of our economic domain this utility function will be defined by the business strategy of the stakeholder and represents in turn a specific economic value/goal optimization. Mathematically, this can be mapped to a multi-dimension multi-choice knapsack problem. Several heuristics have been proposed to solve these QoS-aware service selection problems which are known as NP-hard.

1) Blackboard-based Approach: Being aware of this computational complexity we decided to utilize a heuristic approach for optimize our business service value chains. A blackboard [21] – initially developed in the area of artificial intelligence – implements an A^*-algorithm to heuristically solve

[4] Of course it would be beneficial if historical data is shared between different agents of the same type (complete market transparency), but as this will have a major impact on the overall system, we will postpone this discussion to upcoming publications.

NP-hard problems. It is especially suited for complex problems with incomplete knowledge and uncertainties regarding the attributes and the behavior of the involved components. So we identified a blackboard based approach as extremely useful for the given service chain optimization problem:

- It is very unlikely that each registered provider of a domain is available all the time. We therefore need an approach which is capable to provide also a **solution based on insufficient knowledge**. Blackboards are especially designed to cover this requirement.
- With an increasing number of requests, the intended system needs to scale to provide acceptable response times. Blackboards can be easily distributed due to its agent based structure. This attribute of Blackboards compliments **scalability and throughput concerns** of our system.
- In the blackboards paradigm, the knowledge base is divided into multiple inter-connected regions where each is representing a distinct aspect of the global problem. Each region is maintained by an expert for this aspect, cooperating with all other experts to find a solution for the global problem. This **problem decomposition** fits very well to our problem statement when decomposing process into multiple services and negotiating them individually using **autonomous experts** under consideration of various constraints defined for the process.

In general, a blackboard consists of the three components. A **global blackboard** represents a shared information space containing input data and partial solutions. In our implementation, the information shared consists of services and their characteristics stored in the Service Template Registry (see section III-A). As shown above, the (utility) knowledge base (see section III-C) is composed of several independent regions, each resembles a single blackboard competence. The global blackboard acts as a "mediator", allowing the different regions to communicate. In our implementation, the regions are services which are realized by specific actors (detailed in the following) performing the negotiation process (see section III-B).

The blackboard mechanism is listed in Algorithm 1. The goal function for evaluating possible service offer combinations for a request is called the *happiness function*. A decision tree is generated based on estimation of the happiness function for the visited paths. As shown in Algorithm 1, the expansion (choice) of promising service offers for a step in the request set, is handled by an `OpenList` and `BlockedList`. The `OpenList` contains a list of all possible service combinations to choose from. Each of these steps is rated by applying the happiness function that sums up the happiness of past decisions and the happiness of the next step. Considering this, the service which maximizes the happiness function is chosen for the next step in the optimization approach. The `BlockedList` contains services that do not fulfill the given requirements and therefore must be excluded from the set of possible solutions.

We used this heuristic approach for several multi-dimension multi-choice knapsack problems in the area of computational science in the past [22], [23], [24] and it proved extremely feasible.

```
$OpenList$ = expand($s_1$); $BlockedList$ = [];
while $OpenList \neq []$ do
    $Act$ = best($OpenList$); $OpenList$ \= $Act$;
    if $Act == Goal$ then
        return $Act$;
    end
    foreach $d_x$ in expand($Act$) do
        $d_x.costs = Act.costs + h(d_x.costs)$;
        if $d_x \notin OpenList \wedge d_x \notin BlockedList$ then
            ¡ $OpenList$ += $d_x$;
        else
            if $d_x.costs$ ¡ $OpenList[d_x.id].costs$ then
                $OpenList[d_x.id]$ = $d_x$;
            end
        end
    end
end
```

Algorithm 1: Blackboard Algorithm

2) *Process Aware Agents:* A specific contribution of this paper is the mapping of the Blackboard approach presented above to a pure service oriented process aware scheme. In our case this means that the initiator of a negotiation is allowed to provide complex control flow structures (e.g. BPMN processes) to its agent. In Figure 5 we provide an overview about the necessary agents and how they must interact with each other to enable this behaviour.

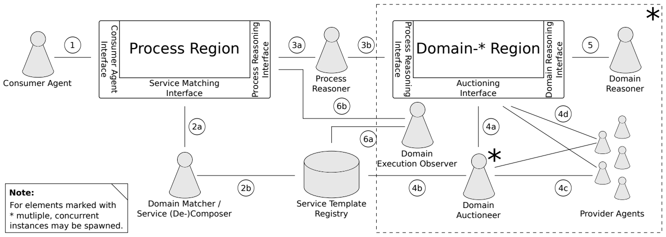

Consumer Agent
1
Consumer Agent Interface
Process Region
Service Matching Interface
Process Reasoning Interface
3a
Process Reasoner
3b
Process Reasoning Interface
Domain-* Region
Auctioning Interface
Domain Reasoning Interface
5
Domain Reasoner
*
6b
2a
6a
Domain Execution Observer
4a
4d
*
Note:
For elements marked with * mutliple, concurrent instances may be spawned.
Domain Matcher / Service (De-)Composer
2b
Service Template Registry
4b
Domain Auctioneer
4c
Provider Agents

Fig. 5. Agents Involved During a Negotiation

The proposed architecture aims to be able to optimize the execution plan of a process in terms of service selection, constantly track its execution, and trigger immediate if incidents occur. Using the $A*$ algorithm allows for nearly optimal resources efficiency, as the further the execution of the process is going, the less uncertainties must be considered (i.e. resources must be spared in order) during the service selection.

In Figure 5 a **Process Region** and a **Domain Region** are depicted, representing a simple, shared and structured storage space for all information related to the process execution and service invocation.

a) Consumer Agent: In our framework the Consumer Agent is in charge of **setting up the process region** on behalf of a service consumer. It therefore needs to fill all four subregions (see Sec. III-C1 for details) with data expressing the consumers preferences. After the setup is complete, the *Domain Matcher / Service (De-)Composer* (Sec. III-D2b) can start with its work. During the execution of the process, the *Consumer Agent* can check the *Execution Plan* for (a) the **selected providers** of each invocation and (b) if the **agreed conditions from the WSAG are respected** (see Sec. III-D2f for details).

b) Domain Matcher / Service (De-)Composer: The purpose of this agent is twofold. First it matches the **service invocations against the *Service Template Registry* domains**.

By parsing the business process, each activity causing a service invocation is assigned to one domain defined inside the *Service Template Registry* (see Sec. III-A) and annotated with the according information while all internal activities of the process are removed to keep it as simple as possible.

Second task of this agent is to check if other options can be explored by **service (de-)composition**. This is basically an attempt to extend the solution space from a single service invocation to (complex) combinations of various service invocations with similar functionality. An extended list of possibilities allows the *Process Reasoner* to explore more options and therefore makes the optimization result more trustworthy.

c) Process/Meta-Reasoner: This agent is the most complex one in our framework. Its purpose is to coordinate the goals/restrictions of the over-all process (*Process Region*) and the various service invocations (*Domain Regions*). To do so, it first parses the *(Abstract) Execution Plan* (updated and annotated by the *Domain Matcher / Service (De-) Composer*) and deduces the domain specific *WSAG templates* for each service invocation by identifying relevant subsets of all data defined in the *Business Rule* subregion, *Economic Cost Model* subregion, *None-functional* subregion, and the *History Data* subregion.

At his point, also the **Meta-Reasoning** is applied. By doing so the Process Reasoner tries to predict potential differences between the agreed SLA and the expected result during invocation in a certain domain. This knowledge is now used to adapt all succeeding WSAG templates to be already aware of potential threats. After all WSAG templates are educed, it spawns a new *Domain Region* for each distinct one and set it up by filling all three subregions (see Section III-C2).

After it has set up all required domain regions (and spawned the according agents) it waits for the first offers of each domain. With the first set of offers it can start to reason about the distinct WSAG offers and either (1) accept them, (2) wait for additional ones, or (3) updated the according WSAG template to meet its expectations. For a start we propose to use an A* algorithm for the process optimization as it has special advantages when finding paths through **multi-dimensional DAGs**.

By applying all defined constraints/models provided by the *Consumer Agent*, the *Process Reasoner* ensures that only valid execution plans are created. As soon as the first valid execution plan is found, the *(Abstract) Execution Plan* is updated and the *Consumer Agent* gets notified. From this time on the optimization is a constant process which updates the execution plan every time the *Ranked WSAG offers* subregion is updated.

d) Domain Auctioneer: As discussed in Sec. III-A the purpose of this agent is to enforce the commonly agreed protocols of the domain (*Class Level Negotiation Protocol*). Every time the WSAG template is updated it starts a new bidding round. For each (first) round all *Provider Agents*, registered in the *Service Template Registry*, are invoked according to their specific protocol (*Instance Level Negotiation Protocol*) to request a WSAG offer and updates the according entries on the blackboard. As this is a constant process, the blackboard always provides the latest valid WSAG offers of each participating provider. To avoid resource dissipation the order in which the *Provider Agents* are asked for offers is deduced from the *History Data* favouring the ones with good results in the recent past. For iterating negotiation protocols, the agent must maintain a list with *Provider Agents* that already left the bidding round to avoid unnecessary resource usage and decide if they should be re-invited after a significant change in the WSAG template happened. If a change is considered to be *significant* is deduced from the *Economic Cost Model*.

e) Domain Reasoner: This agent applies the *Domain Specific Utility Model* to rank the particular WSAG offers collected by the *Auctioneer Agent*. Doing so allows the *Process Reasoner* to efficiently request the top *N* WSAG offers when optimizing the execution plan. It is further in charge of maintaining the list of WSAG offers by deleting expired once and notifying the *Auctioneer Agent* about it. In case of **iterating protocols** this agent constantly updates the WSAG template with the current best, valid offer and therefore triggers constantly new bidding rounds till the best one is identified (i.e. a Pareto Optimum has been achieved).

f) Domain Execution Observer: This agent utilizes the knowledge provided by the *Service Template Registry* to observe the actual service invocation and notifying the *Process Reasoner* about the current state of the execution and about SLA violations. It further adds all observations to the *History Data* subsection and therefore constantly extends the knowledge accessible to the *Process Reasoner* and the *Auctioneer Agent*.

IV. Resource Selection Example

In this section we apply the so far stated concept to a real-world use case in the field of physics data analysis. Due to the limitation of the paper size we have to stick to the sketch of a quite simple example. The Worldwide LHC Computing Grid (WLCG) [25] is used for all computing intensive data analysis tasks related to the various experiments hosted at CERN. We assume that the deadline of a big physics conference is nearing and the need for certain analysis arises. We further assume that all available resources of the WLCG are already occupied by this task. In order to stick to the deadline, additional resources must be acquired (e.g. by *cloud bursting*). The task of deciding which resource provider to use will be used to illustrate our approach to choose the best matching offers to a given request

A prerequisite to rank competing offers and therefore reason about the best one are (1) constraints (e.g. money, disk space, . . .), (2) happiness functions H expressing the added value of the distinct resources (e.g. band width, disk space, . . .) to the overall happiness. To keep our example focused and avoid unnecessary complexity, we assume to have three WSAG attributes to consider: (1) available band width (BW), which we weight with 0.7 in our happiness function. We further state that we need at least a 10Mbit connection to consider the offer and that we do not gain additional happiness over 20Mbit, as we are limited by our infrastructure. (2) we consider disk space (DS) as crucial in the received offers. We know that we need at least 100 gigabyte and that we can achieve a speed-up in execution time up to 200 gigabyte of disk space while running our data analysis. Therefore offers with less than 100 GB are excluded and offer with more than 200 GB do not increase the happiness value. Finally we weight disk space with 0.3 for the overall happiness.

As an example how such constraints can be expressed the resulting happiness function for disk space ($H_{DS}{}^{5}$) is provided:

$$H_{DS}(ds) = \begin{cases} (1_a) & 0: \text{ if } ds < 100GB \\ (1_b) & H_{DS}(ds): \text{ if } 100GB < ds < 200GB \\ (1_c) & H_{DS}(200): \text{ if } 200GB < ds \end{cases}$$

The values must further be normalized to avoid a single input dominating others due to the unit it is traded in e.g. 10 MBit would always be worth only one tenth of the happiness value of 100 GB although their weighting in the happiness function may be different.

$$\begin{array}{lrcl} (2_a) & H_{DS}(ds) & = & 0.7 * \frac{ds - ds_{min}}{ds_{max} - ds_{min}} \\ (2_b) & H_{BW}(bw) & = & 0.3 * \frac{bw - bw_{min}}{bw_{max} - bw_{min}} \end{array}$$

Finally the overall value of the happiness function can be derived as

$$(3) \quad H_{all}(ds, bw) = H_{DS}(ds) * H_{BW}(bw)$$

Due to a very strict budget in research expenses the happiness function is obviously influenced by the money spent for service invocation. So, the price (P) mentioned inside the responded WSAG offers (O) adds as third attribute to the equations. The goal function of the optimization process in this scenario could look like this:

$$(4) \quad min(P((H_{all}(O_n)) \ \forall \ H_{all}(O_n) > 0$$

Consumers can very easy and fast reason about what offer provides their minimal requirements (or as close as it gets) for the lowest price.

Equation 4 is the goal function for our Blackboard algorithm. For our algorithm 1 the start node is the set of defined service requests. The nodes in the OpenList are expanded by mapping matching offers to requests. The happiness function H_{all} evaluates all expanded nodes on basis of usefulness and is used for ordering possible solutions in the OpenList.

This happiness function shown in the examples is defined by the *business rules repository*, which is in the center of our framework as it defines the agent's behaviour when reasoning about which offer to choose.

This example is easily extendable to more complex scenarios. The Blackboard regions allow also the adaptation of requests if offers are not available, e.g. there is no provider offering 100 GB or more. So, if in the business rules repository a compromise is allowed, the request constraint can be weakened by adapting the expanded solution nodes in the OpenList. This approach is similar to handling dynamics in Blackboard methods, which we elaborated in [26].

V. Conclusion and Future Work

E-Business infrastructures need to act in a more agile fashion due to the shift of economy from CAPEX to OPEX. Accordingly ICT environments have to adapt automatically to changing needs.

We presented a novel framework for a smart webservice marketplace which allows for Bazaar-style negotiation processes building service-based value chains based on economic principles.

The implementation of the framework is ongoing. In the near future we will conduct an empirical evaluation of our approach by simulation. A main focus of our future research will be an analysis of the behavior of different auctioning models. Specific focus will be laid on defining business strategies in the knowledge base of the autonomic system by following a Canvas-based modelling approach [27].

References

[1] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open Grid Forum*, 2007.

[2] OpenGridForum, "Ws-agreement negotiation," http://www.ogf.org/documents/GFD.193.pdf, 2012.

[3] A. Rumpl, O. Wldrich, and W. Ziegler, "Extending ws-agreement with multi-round negotiation capability," in *Grids and Service-Oriented Architectures for Service Level Agreements*, P. Wieder, R. Yahyapour, and W. Ziegler, Eds. Springer US, 2010, pp. 89–103. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-7320-7_9

[4] G. E. Kersten and S. J. Noronha, "WWW-based negotiation support: design, implementation, and use," *Decision Support Systems*, vol. 25, pp. 135–154, 1999.

[5] *Agent-based Negotiation for Resource Allocation in Grid*, vol. In Proceeding of the 3rd Workshop on Computational Grids and Applications (WCGA'2005), Rio de Janeiro, Brasil, LNCC, 2005.

[6] H. Saddiki and H. Harroud, "An agent-based negotiation model for user-adaptive service provision," in *Multimedia Computing and Systems (ICMCS), 2012 International Conference on*, may 2012, pp. 973 –978.

[7] G. Lai and K. Sycara, "A generic framework for automated multi-attribute negotiation," *Group Decision and Negotiation*, vol. 18, pp. 169–187, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10726-008-9119-9

[8] W. Mach and E. Schikuta, "A generic negotiation and re-negotiation framework for consumer-provider contracting of web services," in *iiWAS*, 2012, pp. 348–351.

[9] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering it Services as Computing Utilities," in *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, 2008.

[10] M. Sabou and J. Pan, "Towards Semantically Enhanced Web Service Repositories," *Web Semantics: Science, Services and Agents on the . . .*, vol. 5, no. 2, pp. 142–150, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826807000066

[11] D. Martin, M. Burstein, D. Mcdermott, S. Mcilraith, M. Paolucci, K. Sycara, D. L. Mcguinness, E. Sirin, and N. Srinivasan, "Bringing Semantics to Web Services with OWL-S," *World Wide Web*, vol. 10, no. 3, pp. 243–277, 2007.

[12] P. A. Bernstein and U. Dayal, "An overview of repository technology," in *Proceedings of the International Conference on Very Large Data Bases*. Institute of Electrical & Electronics Engeniers (IEEE), 1994, pp. 705–705.

[13] R. Vigne, J. Mangler, E. Schikuta, and S. Rinderle-Ma, "A structured marketplace for arbitrary services," *Future Generation Computing Systems*, vol. 1, no. 28, pp. 48–58, Jan. 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X11001063

[5] Please note that the constraint (1_a) and (1_b) are usually part of the WSAG template (to avoid filter invalid offers) but due to comprehensiveness reasons we included them in the happiness function for this example.

[14] R. Vigne, J. Mangler, E. Schikuta, and S. Rinderle-ma, “WS-Agreement based Service Negotiation in a Heterogeneous Service Environment,” in *to appear in: Service-Oriented Computing and Applications (SOCA'12), The 2012 5th IEEE Internation Conference on*, 2012.

[15] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, “Web services agreement specification (WS-Agreement),” in *Global Grid Forum*, 2004.

[16] E. Christensen, F. Curbera, G. Meredith, and W. Sanjiva, “Web Service Definition Language (WSDL),” http://www.w3.org/TR/wsdll, 2001. [Online]. Available: http://www.w3.org/TR/wsdl

[17] J. Yan, R. Kowalczyk, J. Lin, M. B. Chhetri, S. K. Goh, and J. Zhang, “Autonomous service level agreement negotiation for service composition provision,” *Future Generation Computer Systems*, vol. 23, no. 6, pp. 748–759, Jul. 2007.

[18] J. Odell, S. Poslad, and R. Levy, “Fipa iterated contract net interaction protocol specification,” Dec. 2002.

[19] S. A. White, “Introduction to BPMN,” *IBM Cooperation*, pp. 2008–2029, 2004.

[20] W. Vickrey, “Counterspeculation, auctions, and competitive sealed tenders,” *The Journal of Finance*, vol. 16, no. 1, pp. 8–37, 1961. [Online]. Available: http://dx.doi.org/10.1111/j.1540-6261.1961.tb02789.x

[21] D. Corkill, “Blackboard Systems,” *AI Expert*, vol. 6, no. 9, January 1991.

[22] K. Kofler, I. Haq, and E. Schikuta, “User-Centric, heuristic optimization of service composition in clouds,” in *16th European Conference on Parallel and Distributed Computing (Euro-Par 2010)*, ser. Lecture Notes in Computer Science, P. D'Ambra, M. Guarracino, and D. Talia, Eds., vol. 6271. Springer Berlin / Heidelberg, 2010, p. 405–417, 10.1007/978-3-642-15277-1_39. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15277-1_39

[23] E. Vinek, P. P. Beran, and E. Schikuta, “A dynamic multi-objective optimization framework for selecting distributed deployments in a heterogeneous environment,” in *International Conference on Computational Science (ICCS 2011)*, ser. Procedia Computer Science series. Singapore: Elsevier Science, June 2011.

[24] P. P. Beran, E. Vinek, E. Schikuta, and M. Leitner, “An adaptive heuristic approach to service selection problems in dynamic distributed systems,” in *13th ACM/IEEE International Conference on Grid Computing (Grid 2012)*. Beijing, China: IEEE, 2012, pp. 66–75.

[25] WLCG Project, “The Worldwide LHC Computing Grid (WLCG),” http://wlcg.web.cern.ch/, 2012. [Online]. Available: http://wlcg.web.cern.ch/

[26] E. Schikuta, H. Wanek, and I. U. Haq, “Grid workflow optimization regarding dynamically changing resources and conditions,” *Concurrency and Computation: Practice and Experience*, vol. 20, no. 15, p. 1837–1849, 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.1317

[27] A. Osterwalder and Y. Pigneur, *Business model generation: a handbook for visionaries, game changers, and challengers*. Wiley, 2010.