

# A Digital Marketplace Combining WS-Agreement, Service Negotiation Protocols and Heterogeneous Services

Ralph Vigne  
Faculty of Computer Science  
University of Vienna  
Vienna, Austria  
ralph.vigne@univie.ac.at

Juergen Mangler  
Faculty of Computer Science  
University of Vienna  
Vienna, Austria  
juergen.mangler@univie.ac.at

Erich Schikuta  
Faculty of Computer Science  
University of Vienna  
Vienna, Austria  
erich.schikuta@univie.ac.at

**Abstract**—With the ever increasing importance of web services and the Cloud as a reliable commodity to provide business value as well as consolidate IT infrastructure, electronic contracts have become very important. WS-Agreement has itself established as a well accepted container format for describing such contracts. However, the semantic interpretation of the terms contained in these contracts, as well as the process of agreeing to contracts when multiple options have to be considered (negotiation), are still pretty much dealt with on a case by case basis. In this paper we address the issues of diverging contracts and varying contract negotiation protocols by introducing the concept of a contract aware marketplace, which abstracts from the heterogeneous offers of different services providers. This allows for the automated consumption of services solely based on preferences, instead of additional restrictions such as understanding of contract terms and/or negotiation protocols. We also contribute an evaluation of several existing negotiation concepts/protocols. We think that reducing the complexity for automated contract negotiation and thus service consumption is a key for the success of future service and Cloud infrastructures.

## I. INTRODUCTION

Over the last years, WS-Agreement (WSAG) [1], [2] has become a broadly used standard in the research field of Grid and Cloud computing for providing electronic contracts [3]–[6]. Their success is based on a flexible means to (1) define involved parties, (2) define the context of agreements, and (3) provide necessary terms for guarantees. This makes them observable and consumable and therefore enables automated service integration in Grid and Cloud based systems (see e.g. [7], [8]).

While the flexibility of WSAG allowed it to become the de-facto standard for defining electronic contracts, there is no similar standard to describe negotiation protocols. Research defines automated service negotiation (e.g. [6], [9]–[14]) as the process of establishing business relations between service providers and consumers. It builds on the fact that services providers can guarantee different levels of service to consumers (traditionally: bandwidth, cpu, ...). These service properties of course (1) have effects on each other, (2) lead to different prices, and (3) have to be matched to the preferences of service consumers. FIPA already standardized some

custom negotiation protocols (e.g. [15]) for the Grid / Cloud communities. These protocols often are very specific to their respective application domains, i.e. dealing with a specific set of properties, how to balance them, and how to match them to user preferences in order to maximize the benefit for all participants. However, for all these approaches, providers and consumers have to implement and conform to the negotiation protocol. A generic way to describe complex protocols is still missing.

The contribution of this paper is a means to integrate arbitrary negotiation protocols and enable service domains to leverage WSAG-based electronic contracts. The vision is to allow service consumers to participate in negotiations solely based on their preferences, not on their understanding of the negotiation protocol, thus allowing for the coexistence of negotiation protocols, instead of their standardization.

In order to achieve this feat we built on the Marketplace presented in Vigne et al. [16]. The basic idea behind this Marketplace is to provide a repository of heterogeneous services (different APIs, protocols) and a set of microflows (small code snippets in a process language like e.g. BPEL). While similar approaches (semantic repositories; UDDI+OWLS) concentrate on finding services through semantic annotations, the Marketplace focuses on standardizing the interaction with services for particular application domains. We further improve on this marketplace concept by ...

- 1) Utilizing the Marketplace information and conventions in order to automatically generate contracts. This allows participating service providers to benefit from WSAG without the burden of building WSAG handling and negotiation into their services.
- 2) Explaining how to facilitate the basic concepts of the Marketplace in order to standardize electronic contracts for similar services from different service providers.
- 3) Explaining how to use the concept of microflows (see above) in order to realize bilateral (one-to-one) and multilateral (one-to-many) automated service negotiation. The goal is to abstract from particular services, and instead allow service consumers to participate in a

negotiation solely based on their preferences, not based on their knowledge of negotiation protocols.

We further provide an evaluation of the viability of our approach by means of a case study. This case study consists of a categorization of analysis of current Grid and Cloud negotiation protocols and how they can be realized in the Marketplace framework.

Our research in this area is based on our experience with large scale data stores [17] and complex applications in Grids and Clouds [18]–[20], and strongly motivated by our focus on Web-based workflow optimizations [21], [22] and their respective management [23].

In Section II we provide a short recap about the used Marketplace and its provided concepts and functionality. In Section III we discuss related work from the fields of WSAG negotiation for Grid- and Cloud-based systems. How WSAG handling is integrated into the Marketplace is discussed in Section IV. A case study about bilateral and multilateral negotiation protocols and additionally an example for strategic negotiation is given in Section V. In Section VII we conclude this work and provide an outlook for future work.

## II. THE MARKETPLACE

As this work is based on the Marketplace presented by Vigne et al. [16], it is essential for further discussions to have a basic understanding of the implemented concepts and functionality.

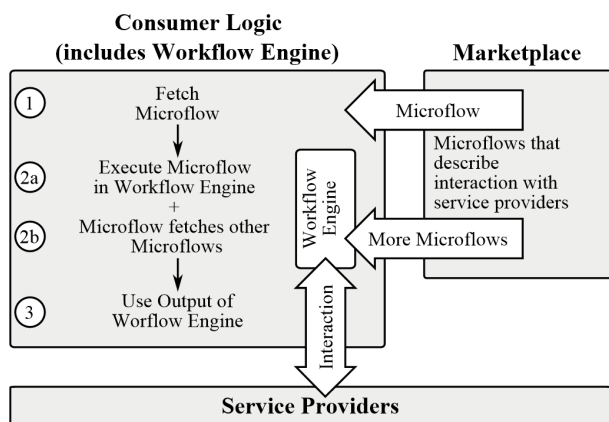


Fig. 1. The Marketplace

As can be seen in Figure 1, the Marketplace holds a set of microflows that describe **how to interact with services**. The Marketplace is strictly **passive**, connotes it does not act as a middleware or proxy. Its purpose is to allow a service consumer to implement a consumer logic against a standardized API for a particular application *domain*.

There are two kinds of microflows contained in the Marketplace:

- Service microflows that implement an interaction with a particular service regarding the requirements of the domain API. This group of microflows is from now on referred to as **instance level operations**.

- Domain microflows that implement logic to fetch and invoke service microflows to select particular services for particular inputs. Each microflow correlates with one operation in the domain API. Thus we will further on refer to this group of microflows as **class level operations**.

From the service consumer’s point of view, the process of doing business with service providers that participate in the Marketplace is totally transparent:

- ① The consumer logic fetches the microflow that represent a particular **class level operation** from the Marketplace.
- ②a The consumer logic instantiates a Workflow Engine (WFE) with the microflow, and invokes the microflow with a set of input parameters (the required input parameters are also defined at the class level, e.g. as a contract).
- ②b While the class level microflow is running, it fetches and executes further **instance level operations** from the Marketplace, in order to find the service provider best matching the consumer’s preferences and use it. This (potentially complex) selection of services is furthermore called **negotiation**, and is one of the subjects of this paper. An instance level microflow (also executed by the WFE) directly interacts with a service, and transforms / filters information to make it usable for class level microflows.
- ③ When a particular class level microflow is finished, its results can be used as part of the consumer logic.

For this paper we again reuse the Cinemas example presented in [16]. The cinema domain consist of three class level operations: (1) **Search** to search for shows of a particular movie, (2) **Book** to book tickets for a particular show and (3) **Search & Book** which combines the two (to demonstrate a complex, multi-part negotiation).

## III. RELATED WORK

As mentioned before, WS-Agreements have become the de-facto standard for electronic contracts in the web service context. In this section we give an overview of WSAG and autonomous Service Level Agreement (SLA) negotiation related research.

Since Andrieux et al. specified WSAG [2] in 2004 it has become a widely used standard in the area of web services to define contracts between two or more parties. Inside a WSAG, all agreed information is structured into different parts. The main parts are namely:

- 1) **Context:** This part contains the meta-data of the agreement e.g. participant’s and life-time.
- 2) **Terms:** This part contains data about the agreement itself. It consists of **Service Terms**, which represent the functionality that will be delivered (so called Items), and the **Guarantee Terms**, which define assurances on the service quality associated with the defined Service Terms.

This formalized and comprehensive way of describing agreements and the according guarantees make WSAG **automatically computable and observable** [8], [24]. Sakellariou

et al. [4] extended Guarantee Terms to be represented as the **result of a function** instead of static values and therefore making it more flexible when observing complex ones.

As WSAG is originally designed for SOAP-based services, Kuebert et al. [25] introduced a way to use them also for **RESTful services** which increased in numbers rapidly over the last few years.

Haq identifies in [5] the **compensation of the service consumer's high dependency on the service provider**, as a result of using WSAG, as an additional reason for their common use in the field of Grid and Cloud computing. He further states that the emerging of Composite Service Providers leads to **complex value chains** and therefore consist of aggregated WSAG. While our approach provides a simple concatenation of these WSAGs, he states that not every information, created during the composition, should be exposed to each participating party. Therefore he introduced **SLA-Views**, which are basically a customized view of the aggregated WSAG for each participating party.

Broberg et al. [3] gives an overview about **state-of-the-art and future directions** in the field of market-based Grid and Utility computing. Although they focus on negotiation strategies related to scheduling problems, they additionally conclude that it is important to overcome the restrictions in flexibility of negotiation protocols to exploit the benefits for service consumers generated by a market with competing service providers.

Zulkernine et al. [6], [26] developed a flexible system for WSAG negotiation with the focus on **multilateral negotiation strategies** based on decision support systems. They also conclude that a way to define flexible and provider specific protocols would further increase the quality of negotiation results.

Faratin et al. [14] give a detailed discussion how such sets of decisions and preferences could look like. Within their example implementation of **autonomous service negotiation**, different offers are evaluated and counteroffers are created on the base of a value scoring algorithm. They define **negotiation tactic** as the short-term decision making focused on a narrow set of criteria while the **negotiation strategy** represents the weighting of criteria over time.

As more complex negotiation protocols come into focus of research, WSAG reaches its limitation as it offers **only two messages**, namely *offer* (as an input for negotiation) and *agreement* (as the output of an negotiation) and therefore makes it only feasible for the "Contract Net Interaction Protocol" [27] (see also Figure 4). Hung et al. elaborated that a formalized negotiation must consist of three different parts which they further used as groundwork when developing the "WS-Negotiation Specification" [28]. These three parts are namely:

- 1) **Negotiation Message** describes the messages which are sent and received during a negotiation. These messages can be of different types, namely *Offer*, *Counteroffer*, *Rejected*, *Accepted*, *Expired*, *SinglePartySigned*, *Signed*, and *Unsigned*.

- 2) **Negotiation Protocol** defines a set of rules and mechanisms the negotiating parties must follow. It uses negotiation primitives to describe different interactions between the parties and what pre- and post-conditions are associated to each primitive (similar to declarative workflow descriptions). The following primitives are defined: *Call for Proposal*, *Propose*, *Accept*, *Terminate*, *Reject*, *Acknowledge*, *Modify*, and *Withdraw*.

- 3) **Negotiation Decision Making** represents an internal and private decision making process based on a cost-benefit function and therefore represents the *negotiation strategy*.

Venugopal et al. [9] focused on the negotiation protocol and proposed Rubinstein's approach of "Alternating Offers" [29] (see also Figure 5) as one way to increase the quality of **bilateral** service negotiation. A **multilateral** adaptation of this protocol has been standardized by [15]. Yan et al. [7] further extended this protocol (see also Figure 6) because of its shortcomings in **negotiation coordination** i.e. coordinating multiple counteroffers for various service providers.

#### IV. WS-AGREEMENTS & THE MARKETPLACE

As stated in the introduction, we try to reuse the Marketplace to commoditize the utilization of contracts (in our case WS-Agreements (WSAG)). We envision this to work similar to the way the Marketplace deals with heterogeneous services (i.e. same application domain but different APIs, slightly different semantics).

##### A. Concepts

For services in the same application domain, it is possible to provide a unified contract template [30], that covers possible terms and restrictions for all services, but has a certain characteristic for particular services. We again propose a split analog to the class level / instance level differentiation described in Section II:

- By providing domain contracts that define the interaction with the Marketplace, as well as common criteria for selecting any service from the domain.
- By providing service contracts that all have the same structure and define the level of service (quality) that can be expected from particular services.

Contracts are not some information that is fetched from a repository, but they are something that is tightly coupled to the usage of services. They are created, negotiated, and accepted while using certain class level operations of the Marketplace. Contracts may of course differ from customer to customer, and are strongly depending on the kind of service that a service consumer tries to utilize (i.e. the input parameters).

Thus before the consumer logic can execute certain operations from the Marketplace (i.e. fetch microflows and execute them in a WFE as describe in Section II), it has to ensure that the properties of a service match its preferences. This can be done by means of a WSAG.

The typical effort for utilizing WSAG includes (1) fetching a contract template, (2) making an offer, and (3) receiving

the agreement (or rejection). Of course additional logic may be necessary to (a) ponder which services to actually make an offer, and (b) decide between multiple possible agreements (under the condition that it is not necessary to enact on all agreements).

Furthermore we have to consider that services may know nothing about WSAG, by still offer an API to achieve something similar. We conclude: it will be necessary to (1) transform WSAG, or (2) implement logic to create WSAG templates and / or decide if a service accepts an offer. This logic can, as explained in Section II be expressed in microflows at service level.

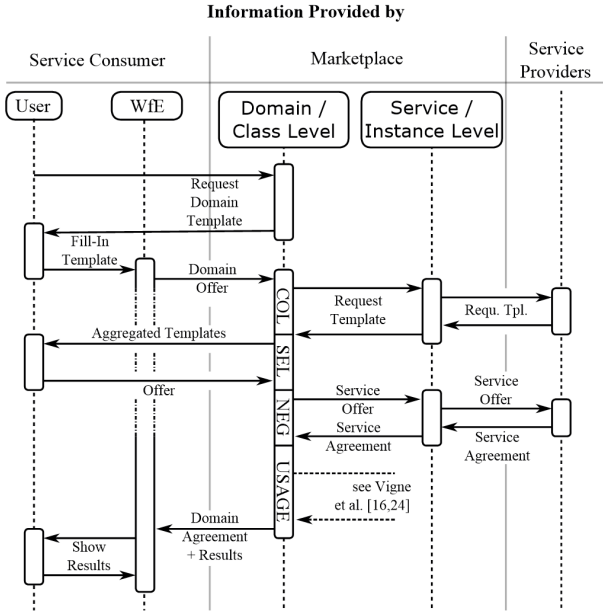


Fig. 2. Abstract Communication Overview of Using the Marketplace

For Figure 2 keep in mind, that the Marketplace is no middleware. It just depicts which kind of information have to be fetched by the WFE, and which kind of information has to be provided by the user and services in order to enable WSAG and SLA negotiation.

As illustrated, the service consumer, after having received a domain level template, a user has to provide the necessary information to create a *domain offer* i.e. WSAG offer. This domain offer is used as input for the according domain operation (microflow enacted by the WFE).

First the operation *collects* (COL) the *templates* of all service providers. This is done executing the according *class and succeeding instance level microflows*. When all templates are collected, they are filtered using the information provided via the *domain offer*.

Next, these *templates are aggregated* and presented to the user who *selects* (SEL) according to personal preferences. Now the selected templates are supplemented with information included in the *domain offer* and therefore become *service offers*.

During the *negotiation* (NEG) this *service offers* are used to negotiate with the according service providers (see Section

V for details about negotiation protocols) using a negotiating class level operation (microflow) and in succession the specific instance level operation. At the end of the negotiation, a *service agreement*, authorizing for the usage of the service, is defined.

Providing this *service agreement*, the service consumer's WFE can use the service (USAGE). How this is done is discussed in detail in [16].

After the service is used, a *domain agreement* is created, using among others the *service agreement* received during the negotiation. This *domain agreement* and the *execution results* of the service execution, representing the defined output of the domain operation, are made persistent for the user.

## B. Implementation

The implementation of a WSAG aware system requires utilizing existing information from the Marketplace about the (1) included services (their properties) and (2) their usage.

- Service properties are a set of static attributes that each service provider must provide to participate in a certain domain of the Marketplace. Examples for the cinema domain are address, number of cinemas/seats, smoking allowed, food corner available, ... Service properties lead to simple WSAG guarantee terms, with a particular unit and/or value range, that may be observable from the outside. Also these properties can be used to create the WSAG business values.
- Required input to class level operations (which is transformed to suitable input for services in instance level operations). Examples include the actual seat to book. The presence of such input parameters may be required or optional, and again expressed in particular units or as a value range. Thus such input parameters lead to WSAG qualifying conditions and/or creation constraints (how to fill out a WSAG template in order to create a valid offer).
- Output of class level operations are the most important part. For them the main purpose of WSAG, to monitor the quality of results, fully applies. They lead to WSAG guarantee terms with a particular unit and/or value range. The outside observability may of course be very different across a range of services. Thus it has to be linked to a class level operation, that in turn may invoke monitoring functionality at the instance level (e.g. Sakellariou et al. [4]).

The creation of a WSAG *domain template* is strictly internal, and fully transparent for the Marketplace consumers. For its implementation we are building on the existing mechanisms of the Marketplace, mainly the definition of properties, and input / output schemas.

Listing 1. Properties Schema

```

1 <rng:grammar xmlns:datatypeLibrary="..." xmlns:rng="..."
  xmlns:prop="..." xmlns:wsag="..." >
2 <rng:start>
3 <prop:properties xmlns:prop="..." xmlns:d="..." >
4 <rng:element name="prop:address">
5 <d:caption xml:lang="en">Address</d:caption>
6 <d:caption xml:lang="de">Adresse</d:caption>
7 <rng:element name="prop:ZIP" wsag:item="Zip">

```

```

8      <d:caption xml:lang="en">ZIP Code</d:caption>
9      <d:caption xml:lang="de">PLZ</d:caption>
10     <rng:data type="integer"><!-- 4 digits -->
11       <param name="minInclusive">1000</param>
12       <param name="maxInclusive">9999</param>
13     </rng:data>
14     <wsag:extension>
15       <!-- specific WSAG information -->
16     </wsag:extension>
17   </rng:element>
18   ...
19 </rng:element>
20 </prop:properties>
21 <wsag:general>
22   <!--
23     General information about the WS-A template
24   -->
25 </wsag:general>
26 </rng:start>
27 </rng:grammar>

```

As can be seen in Listing 1, we added a WSAG specific section to the already existing XML RNG Schema, to allow for WSAG specific information to be added (**wsag:item**). As properties are class level information, it is up to a service domain expert to shape this basis for all contracts.

Listing 2. Class Level Input Message Schema

```

1 <rng:grammar xmlns:datatypeLibrary="..." xmlns:rng="...">
2   <rng:start>
3     ...
4     <rng:element name="input-message">
5       ...
6       <rng:element xmlns:d="..." name="date" wsag:Item="
          DateOfShow">
7         <d:caption xml:lang="en">Date</d:caption>
8         <d:caption xml:lang="de">Datum</d:caption>
9         <rng:data type="date"/>
10        <wsag:extension>
11          <!-- specific WSAG information -->
12        </wsag:extension>
13      </rng:element>
14      <rng:element xmlns:d="..." name="seats" wsag:Item="
          NumberOfSeats">
15        <d:caption xml:lang="en">Seats</d:caption>
16        <d:caption xml:lang="de">Sitze</d:caption>
17        <rng:data type="integer">
18          <param name="minInclusive">1</param>
19          <param name="maxInclusive">6</param>
20        </rng:data>
21        <wsag:extension>
22          <!-- specific WSAG information -->
23        </wsag:extension>
24      </rng:element>
25    </rng:element>
26    <wsag:general>
27      <!--
28        General information about the WSAG template
29      -->
30    </wsag:general>
31  </rng:start>
32 </rng:grammar>

```

In Listing 2 an example class level input schema can be seen. Again it is possible to reuse datatype related information, but additionally units and/or expected ranges could be further described in WSAG.

### C. Aggregation

As defined in [16] the Marketplace allows to define class level operations (microflows) that in turn invoke instance level operations (microflows). Invocation does not mean that the microflows are executed by the Marketplace, but can be fetched from the marketplace in order to execute them in a Workflow Engine (WFE) at the service consumer's side.

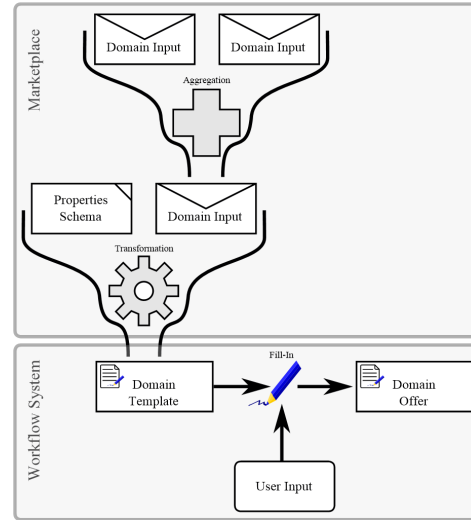


Fig. 3. Class Level Templates: Aggregation and Transformation of Interfaces

But for class level operations it is also allowed to invoke other class level operations in order to e.g. realize a *search & book* operation, which is the fusion of existing operations. As the domain designer has full control over the class level (API), it is not possible that semantically identical parameters with different names may be used (except as an error by the domain designer). Thus aggregating WSAG is straight forward, pieces of information can just be concatenated, duplicate information can be removed.

As properties are valid at class level for all operations, when concatenating the pieces, identifiers have to be prefixed.

Listing 3. Class Level Template

```

1 <wsag:Template xmlns:xsi="..." xmlns:xs="..."
2   xmlns:wsag="..." xsi:schemaLocation="..."
3   xmlns:prop="..." xmlns:msg="...">
4
5   <wsag:Name>Cinemas Domain: Search and Book</wsag:Name>
6   <wsag:Context/><!-- contains general information -->
7   <wsag:CreationConstraints>
8     <wsag:Item wsag:name="Zip">
9       <wsag:Location>
10        // wsag:ServiceDescriptionTerm/prop:address/prop:ZIP
11      </wsag:Location>
12      <xs:minInclusive xs:value="1000"/>
13      <xs:maxInclusive xs:value="9999"/>
14    </wsag:Item>
15    <wsag:Item wsag:name="DateOfShow">
16      <wsag:Location>
17        // wsag:ServiceDescriptionTerm/msg:date
18      </wsag:Location>
19    </wsag:Item>
20  </wsag:CreationConstraints>
21  <!-- additional generated information is placed here -->
22 </wsag:Template>

```

This results in a simple concatenation algorithm depicted in Listing 3. As already described in Subsection IV-A, the aggregated WSAG template can then be filled out by a user and utilized to make offers for a subset of services.

## V. CASE STUDY

In this Section we justify our approach by analyzing a set of negotiation protocols. As already mentioned in the



introduction, negotiation is the process of establishing business relations between service providers and consumers. In the context of this paper this means that a WS-Agreement (WSAG) can be derived, which can be used in order to call services. Existing class level operations can be extended to include the functionality to:

- (1) Collect WSAG templates by invoking instance level operations.
- (2) Present aggregated templates to users, by utilizing callbacks that can be injected into class level operations (microflows) as presented in [16]. It is important to note that all templates will have the same structure / terms included, but may contain different WSAG creation constraints.
- (3) Present offers to services by invoking instance level operations with filled out WSAG templates.
- (4) Enact service calls as a result of accepted offers.

The instance level operations (microflows) mentioned in (1) and (3) can contain any kind of calls to actual services. This allows for a high degree of flexibility: (a) for WSAG unaware services, the logic to create a WSAG could be either embedded in a microflow, or (b) for WASG aware services results can be just transformed.

Furthermore, as described in (2) the service consumer (user) can select a subset of templates that match his preferences. As the service consumer (user) has also an overview of its past decisions, it can employ strategic or tactic negotiation as described in the related work. The same goes for service providers as described in (3).

Because class level operations interact with all participating service providers, its microflow represents a **multilateral negotiation protocol**. A WSAG complaint class level operation has to define a WSAG offer (*domain offer*) as an input and zero to many WSAG agreements (*domain agreements*) as output.

On the other hand passing a particular WSAG offer to an instance level operation for a particular service, can be considered to be an example of a **bilateral negotiation protocol**.

As already mentioned in the introduction, this separation between class and instance level is perfectly suited to map arbitrary per service negotiation protocols in a transparent fashion. The service consumer can just fetch and execute microflows, without having to worry about transforming input or results.

To further exemplify the invoked principles, we evaluated several common concepts in more detail.

#### A. Bilateral Negotiation Protocols

Bilateral negotiation protocols represent how a service consumer is able to negotiate with a **specific service provider (one-to-one)**. As for all negotiations, at the beginning is a *service offer (SO)* and at the end there is either a *service agreement (SA)* or a cancellation of the negotiation. What happens in between is covered by the negotiation protocol (e.g. *send offer*, *confirm agreement* or *quit negotiation*). In our concept, bilateral negotiation protocols are always defined using instance level operations. This is completely in line with

their original intent to provide service specific information to the service consumer which is also considered as one-to-one information.

In the following, we illustrate our concept with two commonly used bilateral negotiation protocols.

1) *Contract Net Interaction Protocol*: The Contract Net Interaction Protocol was introduced by Smith [27] in 1980 and represents the protocol which is today often referred to as “Supermarket Approach”. This means that the service consumer offers a contract (*send offer*) for the requested functionality and the service provider either accepts this offer or rejects it, without providing any counteroffer (“take it or leave it”). Because there is no counteroffer provided for the service consumer, its strategic negotiation service is not necessary. As an example how a microflow, representing this protocol in its simplest form (without any parameter transformation), could look like see Figure 4.

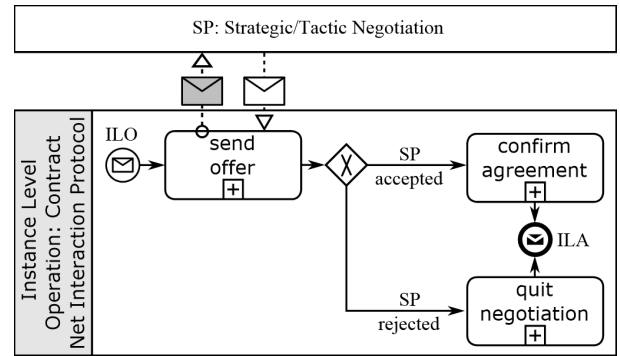


Fig. 4. Bilateral Negotiation: Contract Net Interaction Protocol [27]

We decided to include this protocol in our case study as it is widely used by today’s real world services (e.g. online-shops, cinema service providers, ...). Furthermore is it referred in a lot of today’s research (e.g. [2], [6], [12], [13]) and thus relevant for the Marketplace.

2) *Alternate Offers Protocol*: The “Alternate Offers Protocol” was defined by Rubinstein [29] in 1982 and provides the possibility for service providers (SP) and service consumers (SC) to bargain about the delivered functionality and usage conditions. As illustrated in Figure 5 the protocol allows both parties to propose counteroffers (*SC: create counteroffer* and *SP: request counter template*) if one is not willing to accept the current offer. In WSAG, each term defines permissible values [2] used as constraints when creating the counteroffer. This cycle goes on until both parties either accept the current offer or one party quits the negotiation process.

Because this protocol allows counteroffers, the service consumer must provide a *strategic negotiation* logic, to decide how the negotiation should continue and create counteroffers (see Section V-C for details). Doing so allows each service consumer to realize his own bargaining strategy without customizing the instance level operation defined by the service provider.

Although, as to our best knowledge, not much of today’s real world services support this protocol, it is quite popular



As an example for a negotiation strategy they allege that, in the beginning of multilateral negotiations, the behavior of competing service providers may influence the counteroffer more than the value of a single criteria. Therefore behavior representing tactics are weighted higher than other tactics. However, at the end of the negotiation phase, single criteria may gain importance. How tactics and strategy have to be combined to achieve the best negotiation results, can be deduced from various sources e.g. historical data or user's preferences.

In our approach all this logic is provided on behalf of the service consumer, thus not directly affects the Marketplace.

## VI. EXTENDED CINEMAS EXAMPLE

So far we introduced our concept for WSAG integration and flexible bilateral and multilateral negotiation protocols. In this Section we integrate them into the cinemas example introduced in Section II and [16] as illustrated in Figure 7.

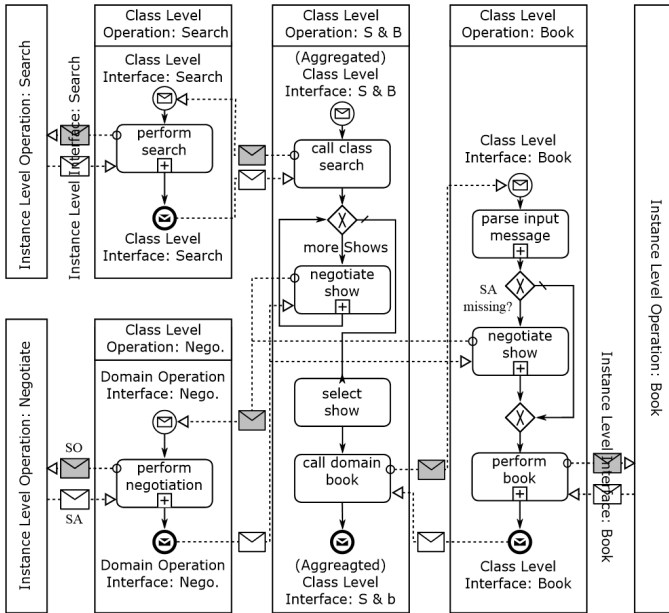


Fig. 7. Multilateral Negotiation: Cinemas Example

In order to use this operation, the service consumer provides a *domain offer* for the class level operation *Search & Book* as only input.

Calling the class level operation *Search* is the same as described in [16]. Only the necessary parameter transformation before and after the call are different, as the interface of *Search & Book* and the internal data representation has changed.

Now that all offered shows are known, negotiations for each show are performed. To do so the according *class level operation* is called (*negotiate show*). It is defined inside the *domain interface* of this operation that a reference to a particular show must be included in the input and agreements for each show are the output. Using this input message, the operation is capable to deduce the correct service provider and calls its *instance level operation* for negotiation (*perform negotiation*). Also parameter transformations to create *service*

*offers (SO)* and to compute *service agreements (SA)* must be performed.

After all shows are negotiated, the service consumer (acting as the *strategic negotiation service*) selects the preferred show (*select show*). Further must all *service agreements*, expect the selected one, be canceled. This is done as described in the agreement e.g. sending the agreement to a specific URI. It must be noted that we excluded this message flow in our illustration to keep it neat.

In the end, the selected *service agreement (SA)* is used as an input for the class level operation *Book*. In our example this operation defines in its class level operation interface that either a *service agreement* or a reference to a particular show is valid input. Depending on this input, the operation decides (*parse input message*) if it must negotiate the show (*negotiate show*) before using it or not. In the end the service operation *Book* of the according service provider can be called. It defines in its *instance level operation interface* that a *service agreement (SA)* must be included to authorize the usage. As output the *execution results* of the service are define.

This *execution results* and the selected *service agreement* represent the defined output of *Search & Book*.

## VII. CONCLUSION AND FUTURE WORK

In this paper we presented flexible ways to (a) define WSAG contracts for arbitrary services, and to (b) transparently allow different services to utilize custom negotiation algorithms and APIs. We showed that both goals could be achieved by building on an existing Marketplace concept, that promotes microflows (small pieces of code executable by a Workflow Engine) to achieve a high level of abstraction for service consumers.

We first described how to extend the concept of unified interfaces to create unified contracts. We then introduced how to use the concepts of class (domain) and instance (service) level to provide flexible negotiation protocols. Finally we provided a case study to justify our approach.

Within the case study we discussed how (1) bilateral negotiation protocols are defined using instance level operations, (2) how multilateral negotiation protocols are defined using class level operations, and (3) how service consumers can include customized negotiation strategies. We presented how to implement the “Contract Net Interaction Protocol” (because of its prevalence for real world services) as well as the “Alternate Offers Protocol” (as prerequisite for mature markets). For multilateral negotiation protocols we presented the “Iterated Contract Net Interaction Protocol” in order to allow various service providers to compete about business. Additionally we presented one negotiation strategy based on a value scoring algorithm.

The conclusion we can draw from this work is, that by embracing existing workflow-related concepts, it is possible to build a lightweight solution for dealing with electronic contracts, in an environment where not all services support them (or support them in the same way).



Future work will be to exploit the complexity of different types of negotiation protocols and to implement some custom negotiation strategy services. The possibilities to monitor complex guarantee terms using class level operations in the context of service composition is also promising. This can lead to much improved error handling and service selection, as it may allow for the prediction of service behavior.

## REFERENCES

- [1] H. Ludwig, "WS-Agreement concepts and Use-Agreement-Based Service-Oriented architectures," *IBM Research Division*, 2006.
- [2] 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.
- [3] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented grids and utility computing: The state-of-the-art and future directions," *Journal of Grid Computing*, vol. 6, no. 3, p. 255–276, 2008.
- [4] R. Sakellariou and V. Yarmolenko, "On the flexibility of WS-agreement for job submission," in *Proceedings of the 3rd international workshop on Middleware for grid computing*, 2005, p. 1–6.
- [5] I. Ul Haq, A. Huqani, and E. Schikuta, "Aggregating hierarchical service level agreements in business value networks," in *Business Process Management: 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings 7*. Springer, 2009, pp. 176–192.
- [6] F. H. Zulkernine and P. Martin, "An adaptive and intelligent SLA negotiation system for web services," *IEEE Transactions on Services Computing*, vol. 4, no. 1, pp. 31–43, Jan. 2011.
- [7] 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.
- [8] H. Ludwig, A. Dan, and R. Kearney, "Cremona: an architecture and library for creation and monitoring of WS-agreements," in *Proceedings of the 2nd International Conference on Service oriented computing*, 2004, p. 65–74.
- [9] S. Venugopal, X. Chu, and R. Buyya, "A negotiation mechanism for advance resource reservations using the alternate offers protocol," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, 2008, p. 40–49.
- [10] M. Comuzzi and B. Pernici, "An architecture for flexible web service qos negotiation," in *Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*. IEEE, 2005, pp. 70–79.
- [11] M. Parkin, D. Kuo, and J. Brooke, "A framework and negotiation protocol for service contracts," in *2006 IEEE International Conference on Services Computing (SCC'06)*, Chicago, IL, USA, Sep. 2006, pp. 253–256.
- [12] P. Hasselmeyer, H. Mersch, B. Koller, H. N. Quyen, L. Schubert, and P. Wieder, "Implementing an SLA negotiation framework," in *Proceedings of the eChallenges e-2007 Conference, Hague, Netherlands*, 2007.
- [13] A. Pichot, P. Wieder, O. Wäldrich, and W. Ziegler, "Dynamic SLA-negotiation based on WS-Agreement," in *Proceedings of the 4th International conference on Web Information Systems and Technologies (WEBIST 2008)*, 2008, p. 38–45.
- [14] P. Faratin, C. Sierra, and N. R. Jennings, "Negotiation decision functions for autonomous agents," *Robotics and Autonomous Systems*, vol. 24, no. 3-4, p. 159–182, 1998.
- [15] J. Odell, S. Poslad, and R. Levy, "FIPA iterated contract net interaction protocol specification," Dec. 2002. [Online]. Available: <http://www.fipa.org/http://fipa.org/specs/fipa00030/SC00030H.pdf>
- [16] R. Vigne, J. Mangler, E. Schikuta, and S. Rinderle-Ma, "A structured marketplace for arbitrary services," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 48–57, 2012.
- [17] E. Schikuta, T. Fuerle, and H. Wanek, "Vipios: The vienna parallel input/output system," in *Euro-Par'98 Parallel Processing: 4th International Euro-Par Conference*. Springer, 1998, pp. 953–958.
- [18] W. Mach and E. Schikuta, "A generic negotiation and re-negotiation framework for consumer-provider contracting of web services," in *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, 2012, pp. 348–351.
- [19] E. Schikuta and T. Weishaupl, "N2grid: neural networks in the grid," in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 2. IEEE, 2004, pp. 1409–1414.
- [20] E. Schikuta, F. Donno, H. Stockinger, H. Wanek, T. Weishäupl, E. Vinek, and C. Witzany, "Business in the grid: Project results," in *1st Austrian Grid Symposium*. OCG, December 2005. [Online]. Available: <http://eprints.cs.univie.ac.at/745/>
- [21] E. Schikuta, H. Wanek, and I. Ul Haq, "Grid workflow optimization regarding dynamically changing resources and conditions," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 15, pp. 1837–1849, 2008.
- [22] K. Kofler, I. ul Haq, and E. Schikuta, "A parallel branch and bound algorithm for workflow qos optimization," in *2009 International Conference on Parallel Processing*. IEEE, 2009, pp. 478–485.
- [23] G. Stuermer, J. Mangler, and E. Schikuta, "Building a modular service oriented workflow engine," in *2009 IEEE international conference on service-oriented computing and applications (SOCA)*. IEEE, 2009, pp. 1–4.
- [24] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour, "Semantic WS-agreement partner selection," in *Proceedings of the 15th international conference on World Wide Web*, 2006, p. 697–706.
- [25] R. Kübert, G. Katsaros, and T. Wang, "A RESTful implementation of the WS-Agreement specification," in *Proceedings of the Second International Workshop on RESTful Design*, 2011, p. 67–72.
- [26] F. Zulkernine, P. Martin, C. Craddock, and K. Wilson, "A Policy-Based middleware for web services SLA negotiation," in *IEEE International Conference on Web Services, 2009. ICWS 2009*. IEEE, Jul. 2009, pp. 1043–1050.
- [27] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *Computers, IEEE Transactions on*, vol. 100, no. 12, p. 1104–1113, 1980.
- [28] P. C. Hung, H. Li, and J.-J. Jeng, "Ws-negotiation: an overview of research issues," in *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*. IEEE, 2004, pp. 10–pp.
- [29] A. Rubinstein, "Perfect equilibrium in a bargaining model," *Econometrica: Journal of the Econometric Society*, p. 97–109, 1982.
- [30] I. Brandic, D. Music, P. Leitner, and S. Dustdar, "VieSLAF framework: Enabling adaptive and versatile SLA-Management," in *Grid Economics and Business Models*, J. Altmann, R. Buyya, and O. F. Rana, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5745, pp. 60–73. [Online]. Available: <http://www.springerlink.com/content/1380u04815284287/>
- [31] M. B. Chhetri, J. Lin, S. K. Goh, J. Yan, J. Y. Zhang, and R. Kowalczyk, "A coordinated architecture for the agent-based service level agreement negotiation of web service composition," in *Software Engineering Conference, 2006. Australian*, 2006, p. 10–pp.