RC23949 (W0605-010) May 2, 2006
Computer Science

IBM Research Report

WS-Agreement Concepts and Use - Agreement-Based Service-Oriented Architectures

Heiko Ludwig
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

WS-Agreement Concepts and Use - Agreement-Based Service-Oriented Architectures

Heiko Ludwig

1 Introduction

The use of Web services in an enterprise environment often requires quality guarantees from the service provider. Providing service at a given quality-of-service (QoS) level consumes resources depending on the extent to which the service is used by one or more clients of a service customer, e.g., the request rate per minute in the case of a Web service. Hence, a service client and a provider must agree on the time period in which a client can access a service at a particular QoS level for a given request rate. Based on this agreement, a service provider can allocate the necessary resources to live up to the QoS guarantees. In more general terms, the service provider commits to – and a service customer acquires – a specific **service capacity** for some time period. We understand service capacity as a service being provided at a given QoS for a specific client behavior constraint. This constraint is typically the number of requests per minute for specific Web service operations and may include constraints on the input data for computationally intensive operations.

In the information technology (IT) services industry, agreements, specifically Service Level Agreements (SLAs), are a widely used way of defining the specifics of a service delivered by a service provider to a particular service customer. This includes the service provider's obligations in terms of which services at which quality, the modalities of service delivery and the quantity, i.e. the capacity of the service to be delivered. Agreements also define what is expected by the service customer, typically the financial compensation and the terms of use. We will use the shortcut SLA for any type of agreement between organizations in the context of this paper although others sometimes use it as only the part of an agreement that relates to quality of service.

While SLAs have been often applied to low-level services such as networking, server management and manual services such as helpdesks, software-as-a-service, accessed through Web or Web services interfaces is becoming more attractive to organizations. In a Saugatuck Technology study published in Network World, the questioned CIOs believe that in 2005 14% of their software application budget is spent on software-as-a-service, and it is believed that this proportion will increase to 23% by 2009.

The Web services stack as defined by WSDL, SOAP, UDDI, the WS-Policy framework and other specifications primarily addresses the issue of interoperability across application and domain boundaries. It enables a client to learn about a service and its usage requirements, bind to it in a dynamic way, and interact as specified, i.e. potentially in a secure, reliable and transactional way. The organizational notions of service provider and service customer are beyond the scope of the specifications and SLAs between autonomous domains are dealt with out of band, typically in a manual process.

The WS-Agreement specification is defined by the Grid Resource Allocation Agreement Protocol (GRAAP) working group of the Global Grid Forum (GGF). It enables an organization to establish an SLA in a formal, machine-interpretable representation dynamically as part of a Service-Oriented Architecture (SOA). WS-Agreement provides the standard specification to build an agreement-driven SOA in which service capacity can be acquired dynamically as a part of the service-oriented paradigm and the corresponding programming model. The specification comprises an XML-based syntax for agreements and agreement templates, a simple agreement creation protocol and an interface to monitor the state of an agreement.

This chapter outlines the concept of agreement-driven SOA, explains the elements of the WS-Agreement specification, and discusses conceptual and pragmatic issues of implementing an agreement-driven SOA based on WS-Agreement.

1.1 Motivation

Web services aim at flexibly and dynamically establishing client service connections in a loosely coupled environment. Traditional SLAs between organizations define the quantity and the quality of a service that one organization provides for another as well as the financial terms of the relationship. However, traditional SLAs are usually quite static due their high cost of establishment and hence their, typically, long runtime. The traditional way of establishing SLAs hampers the dynamicity of Web services in a cross-domain environment.

Consider the following simplified scenario:

A **financial clearing house** provides 3rd party services to buyers and sellers at one or more securities exchanges. The customers of the clearing house maintain accounts with the clearing house and use a Web services interface to manage their accounts, i.e. inquire the current balance and the list completed and pending transactions, and transfer funds to and from the account. This service is called the *account management service*. In addition, clients can submit trades they made to be settled by the clearing house through a Web services interface. They can monitor the state of the clearing process and request notification of events such as the completion of the clearing process or exceptions that occurred. This Web service interface is called the *settlement service*. For the clients of the clearing house, performance and availability of the service is essential. Quality-of-service (QoS) guarantees, along with pricing and penalties are defined in an SLA. The performance parameters relate to the response time for account operations and clearing requests (the submission), notification times of clearing events, and the total time of the clearing process (from submission to completion). For clearing requests, an absolute maximum response time per request is defined as well as the maximum average response times in a 5 minute and one day window. The availability is measured as a time-out of 30 seconds of a Web service request. The different clients of the clearing house have different requirements regarding the service's QoS parameters. Since better QoS guarantees require more resources to implement a Web service, QoS guarantees are capped to a certain number of requests per time span, e.g., 1000 requests per minute and 100000 per trading day. If more requests are submitted, either another – lower – set of QoS guarantees applies or no performance guarantees are given. Finally, clients' service capacity demands vary over time. Triggered by certain events such as end-of-year trades, tax dates, initial public offerings and acquisition of new customers by the clients, settlement capacity requirements temporarily or permanently change over time. To differentiate the clients' Web services requests, each client is given different Endpoints.

The clearing house implements their Web services on a multi-tiered cluster of servers comprising HTTP servers, application servers and database servers. The cluster is connected to a storage area network (SAN) where data is managed. The clearing house maintains also an off-site backup data center to which all data is mirrored in real time. To connect to customers and to the off site data center, the clearing house rents VPN bandwidth from 3 different networking companies. A monitoring system gathers response time data from the application servers' instrumentation and checks the compliance of each SLA's QoS guarantees. Penalties are deducted from the clients' monthly bills. The clearing house wants to keep costs low by minimizing resource consumption, i.e. the number of servers and storage it owns in main and off-site data center and the amount of bandwidth buys. To minimize resource consumption, the clearing house shares resources between its clients. All requests are routed to a central workload manager and then prioritized according to its QoS level. If current demand exceeds the cluster's capacity, those requests which entail the least penalty are delayed. The cluster's capacity and the network are adjusted to maximize profit, not to serve peak demand.

Today, clients request changes in capacity by phone. A call center agent checks the management system whether the change can be accommodated at the requested time. If additional capacity is needed and provisioning time permits, it checks whether it can buy the additional cluster servers, notification nodes and networking capacity. Based on this information, the agent potentially submits orders for the additional capacity, if required and possible, and schedules a configuration change of the central dispatcher node to take the additional workload into account. The clearing house wants to automate the request for additional Web services capacity at varying QoS levels based on a standard interface that all clients can use.

The clients of the clearing house are **brokerage companies**. Brokerage companies can use different clearing houses at the securities market. They receive trades from various clients, e.g., private individuals, institutional investors such as mutual funds or pension funds, or companies issuing and managing bonds. Client demand is volatile in a brokerage house. Changes in demand can be foreseeable, such as the end-of-year business, or spontaneous, such as news that trigger stock and bond market activity. For planned changes, brokerage companies want to buy clearing services capacity in advance. However, brokerage companies also want to manage sudden onset of demand. If market activity increases to or beyond the capacity currently acquired, brokerage companies want to buy additional clearing house capacity from the clearing house offering the best conditions at the time. This short time acquisition process should be fully automated to accommodate the unplanned increase of Web service client traffic.

1.2 Requirements and Assumptions

The introductory discussion and the example scenario illustrate a number of interesting observations and requirements related to capacity and performance of Web services:

- First, performance QoS parameters such as response time vary with the client workload submitted to the Web service, given the set of resources stays constant. If a Web service provider wants to guarantee a QoS level, it has to anticipate the client workload and add resources correspondingly. In an inter-organizational scenario, an SLA is a viable means of conveying future capacity requirements to a service provider and the mechanism to establish SLAs must provide this function.
- From a service provider's perspective, its ability to provide service at a given QoS level is bounded by the number and capacity of the resources available at the time of service. The service provider must be able to decline requests for further capacity if it would endanger satisfying the SLAs that have been established already, and to reject the corresponding Web services requests.
- Given the resource dependency of performance QoS properties, a service customer wants to establish SLAs ahead of time to ensure that its clients will receive the QoS they require.
- If Web service clients require more capacity at runtime, service customers will want to shop around multiple providers to find the required capacity at the best price. They need a standard mechanism to search partners and create agreements.
- To accommodate short term capacity requests as outlined in the example, the service capacity acquisition mechanism must have the ability to be fully automated, at least for cases of simple decision-making.
- Service customers must monitor their Web service clients' activity internally to identify additional service capacity requirements. Finally, service customers must be able to direct Web client requests to Web service providers according to the contracted capacity.

Given the requirements of Web service providers and Web service customers to manage Web service capacity and to acquire capacity in a dynamic way, we need a means of establishing SLAs for Web services in an automated and standardized way that is integrated in the Service-Oriented Architecture.

For the scope of this chapter, we will ignore all issues of security and malicious use of the SLA mechanism. We assume that these issues can be address in a manor orthogonal to the conceptual discussion here.

1.3 Related Work

There are multiple approaches to use the concept of – formalized – agreements (contracts) in the context of electronic services, both in specifying contractual agreements as well as in architectures and systems that deal with agreements.

A number of approaches address various issues of the process of agreement creation, fulfillment and monitoring independently of the particular technology environment of the electronic service. In the context of the ODP Enterprise Language, a high-level model for the representation of contractual obligations has been proposed but no explicit syntax has been specified by the ISO [12]. There are multiple non-standardized proposals for specific languages, which have not found widespread adoption, for example [7]. There are multiple other approaches to representing and formalizing contractual content independent of any standard approaches, e.g., SORM [19], a model of contractual rights and obligations representing enforcement and monitoring-related aspects of a contract, and policy-related work as conducted by Milosevic et al. [7], [22]. Further work on architectures for agreement-based systems has been presented independently of Web services contexts [16], [10].

Some approaches address specifically SLAs for Web services. The Web Service level Agreement (WSLA) approach proposes a language and monitoring framework for Web and other services [18], [13]. Parties can define service level parameters and specify service level guarantees as logic expressions over them. The semantics of service level parameters is described by a functional specification of the way in which high-level metrics are computed from well known resource-level metrics. It can be specified which party to an agreement, provider, customer or 3rd party collects them and when metric values and compliance and violation messages are to be sent to other parties. The WSLA Measurement Service interprets the WSLA specification and sets up a corresponding distributed measurement environment. While this approach comprises many elements required for monitoring SLAs, there is no establishment mechanism and the mapping from an SLA to an implementation infrastructure requires substantial additional effort, as discussed in [4] and [6].

The Web Services Offer Language (WSOL) proposes a language to represent Web services-related SLAs [24]. It covers a similar scope than WSLA but uses ontologies and low-level languages to define metric semantics. It has not been shown how to derive a service implementation from a WSOL representation. Bhoj et al. propose another XML-based SLA representation. Like in WSOL, the metric semantics is described in an executable language such as SQL or Java [2].

A number of publications address the issue of representing performance-oriented and other QoS parameters for the selection of services, e.g., in GlueQoS [25], and offer related categorizations and ontologies, e.g. [21], [22] and [23], primarily used for match-making services.

Other related approaches propose agreements formats and infrastructure to facilitate interaction and coordination between parties, e.g., tpaML/BPF [3] and CrossFlow [9]. However, these approaches are not suitable for service capacity reservation.

2 Agreement-Driven Service-Oriented Architectures - Extending the SOA Paradigm

To address the issue of service capacity, we have to extend the architectural model of the service-oriented approach. In the traditional model of SOA, service properties are published to a directory such as UDDI. Clients looking for service can pick services depending on their capabilities described in the directory or as meta-information at the service endpoint. When decided, clients can bind to it by establishing the transport protocol relationship, which can be as simple as sending SOAP messages over HTTP or establishing a secure connection, e.g., via SSL prior to using it. In this traditional model of binding to a Web service, it is assumed that any client can bind to a service if it has the capabilities required in the service's meta-information.

In a capacity-aware SOA, the organization owning clients, our example's brokerages, and the organization owning the Web service, e.g., the clearing house, agree prior to service usage on the conditions under which clients of the customer organization can use the service of the provider organization. Clients are not serviced without agreement. The main elements of this Agreement-Driven Service-Oriented Architecture are outlined in Figure 1.

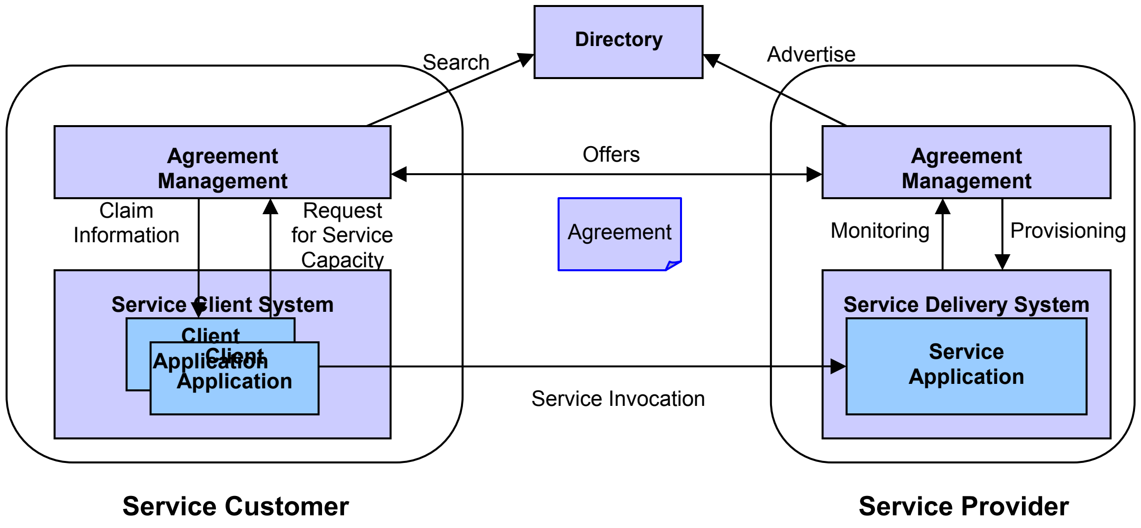

Figure 1: Agreement-driven service oriented architecture.

Like in a traditional SOA, Web services *Client Applications* run on a *Service Client System*, the execution environment of the client application, e.g., a managed network of PCs. The client application invokes a *Service Application* implementing a Web service in a *Service Delivery System*, e.g., the managed cluster of servers of our clearing house. However, the client application's way to bind to a Web service is different.

In an agreement-driven SOA, Web services and clients belong to organizations. Web services invocations between organizations are governed by an agreement. To obtain authorization to use a Web service, the client application derives its requirements for a service, including the QoS properties and the capacity it needs and submits a Request for Service Capacity (RSC) to the Agreement Management.

The Agreement Management component of a Service Customer searches for suitable Service Providers and negotiates an SLA for the required service capacity. It returns the service claiming information back to the client application. The claiming application describes how to access the service according to the contract and might be either a specific Endpoint, adding a claiming token to the SOAP header of requests or mechanisms. Using the claiming information, the client application can then access the Web service on the terms negotiated in the agreement, e.g., adhering to the request rate cap and receiving the agreed performance. If the service performance does not live up to what has been guaranteed, the client can report the problem to the Agreement Management component, which can handle the dispute with the Service Provider and either seek correction of the problem or financial recourse. If the problem cannot be corrected, a new agreement with another service provider can be negotiated.

The Agreement Management component of a Service Provider advertises the Service Provider's capabilities and negotiates agreements. When receiving agreement offers from potential Service Customers, the Agreement Management component assesses the available service capacity of its service delivery system. If the capacity is available at the conditions specified in the agreement, it schedules the provisioning of the service at these conditions and returns the corresponding claiming information back to the Service Customer. When the service is due to be delivered, it provisions the service, i.e. allocates servers, installs service applications, and configures network components and workload managers.

The implementation of an agreement-driven SOA differs in a number of ways from the implementation of a traditional SOA:

Client applications must be able to determine their QoS and capacity requirements. This is often the case in a high-performance, Grid-like application scenario such as gene sequencing. However, in the context of business applications such as the clearing house's clients, it is impractical to have each client determine its requirements independently. The capacity management function can be implemented separately from the actual business logic of the client and then also take into account the collective requirements of all clients for a service. Furthermore, clients must be able to interpret claiming information and potentially add claiming details to their service requests. Again, this function can be separated into a gateway that intercepts service requests from the client and adds the additional information.

Service applications delivering Web services also require some additional functionality. Service requests must be differentiated by their agreement. Resource must be assigned to agreements or groups of agreements – classes of service. Compliance with QoS guarantees must be monitored and, in case of conflict, yield management decisions must be made. Some service providers already implement different classes of service which they differentiate on basis other than agreements. Hence, typically, preparing a service delivery system for an agreement-driven SOA is less effort than the client-side changes.

The Agreement Management component is a new element of an SOA implementation architecture. Various issues have to be addressed: Managing the life-cycle of agreements, i.e. creation, monitoring, expiry management and renewal, is a common issue for both Service Customer and Service Provider. Also, both parties have to design the access their respective service systems to monitor agreement compliance. A Service Provider must design a decision making function that derives the resource requirements for a particular agreement in question and assesses whether the agreement is feasible and economically viable. The Service Customer has to devise a decision-making mechanism that decides how to allocate capacity between multiple Service Providers.

3 WS-Agreement Concepts

The WS-Agreement specification addresses the interfaces and interaction between service provider and consumer at the Agreement Management level. These interfaces are based on the Web Services Resource Framework (WSRF), each interface being defined as a resource with properties and addressed by an Endpoint reference. While this is not important on the conceptual level of this section we will use WSRF terminology in describing the main conceptual elements of the WS-Agreement specification, which are outlined in Figure 2.

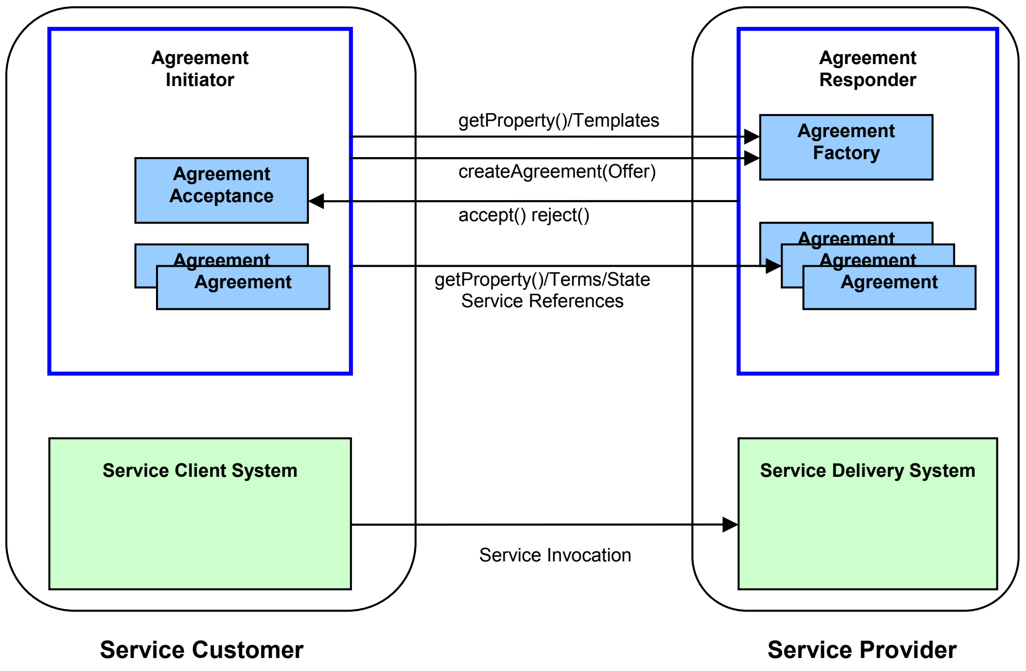

Figure 2: WS-Agreement overview.

WS-Agreement defines to parties in the dialog to establish an agreement: the *Agreement Initiator*, and the *Agreement Responder*. These roles are entirely orthogonal to the roles of a Service Provider and Service Customer. Service Providers and Service Customers can take either role, depending on the specific setup of an application domain.

An Agreement Responder exposes an *Agreement Factory*. The Agreement Factory provides a method to create an agreement by submitting an *Agreement Offer* in the offer format defined by the WS-Agreement XML schema. The Agreement Factory can also expose a set of *Agreement Templates* that Agreement Initiators can retrieve to understand which kind of agreements the Agreement Responder is willing to enter. Agreement Templates are agreement prototypes that can be modified and completed according to rules by an Agreement Initiator. Upon receiving an Agreement offer, an Agreement Responders decide whether they accept or reject an offer. If an Agreement Responder decides immediately it returns the decision as a response to the synchronous call. Alternatively, it can use an Agreement Initiator's *Agreement Acceptance* interface to convey the decision. A response to an agreement creation request contains an Endpoint reference to an *Agreement*. An Agreement is created by an Agreement Responder and exposes the content of the agreement and its runtime state. The runtime state comprises the overall state of the agreement, which is pending, observed, rejected and completed, and the state of compliance with individual terms of the agreement, which we will discuss in detail in the subsequent sections of this chapter. The runtime status information of the agreement can be used by both parties to manage their service delivery and service client systems, respectively. The Agreement can also provide a list of references to the services that are subject to the agreement.

In addition to the Agreement Responder, the Agreement Initiator can also expose an Agreement instance to make term compliance status information available to the Agreement responder. This may be useful in cases where measurements determining the term compliance status are taken at the system of the Agreement Initiator. For example, in the case of an Agreement Initiator being the Service Provider, throughput measurements and server-side response times are gathered at the Service Provider's instrumentation and it may be easier to evaluate at the Service Providers if a guarantee has been complied with.

Based on the early experience gathered in WS-Agreement deployments, Service Providers typically assume the role of Agreement Responders and provide the Agreement resource. This also holds for our scenario. The brokerage companies ask for capacity and hence assume the role of the Agreement Initiator, the clearing house is the Agreement Responder. However, scenarios in which Service Providers take the initiative to create agreements are common in Grid computing environments and we may see such a development in a Web services context in which Service Providers monitor the service usage by their Service Customers and suggest new agreements when they reach the capacity limits of their agreements.

WS-Agreement does not define the interaction between the Agreement Management level and the Service level of the agreement-driven SOA, neither for Service Providers not Service Customers. How to provision a service from an agreement depends on the particular application domain and the specific implementation of the service delivery system used and is beyond the domain-independent scope of WS-Agreement.

4 Agreement Model

To facilitate the negotiation, establishment and monitoring of an agreement, the parties involved in the process must have a common understanding of the agreement content. WS-Agreement defines a standard model and high-level syntax for the content of Agreement Offers and Agreement Templates. This section provides an overview of the agreement model, the details of the agreement elements being described in the language section.

The main structure of an agreement – and of offers and agreements template – is outlined in the following UML diagram.

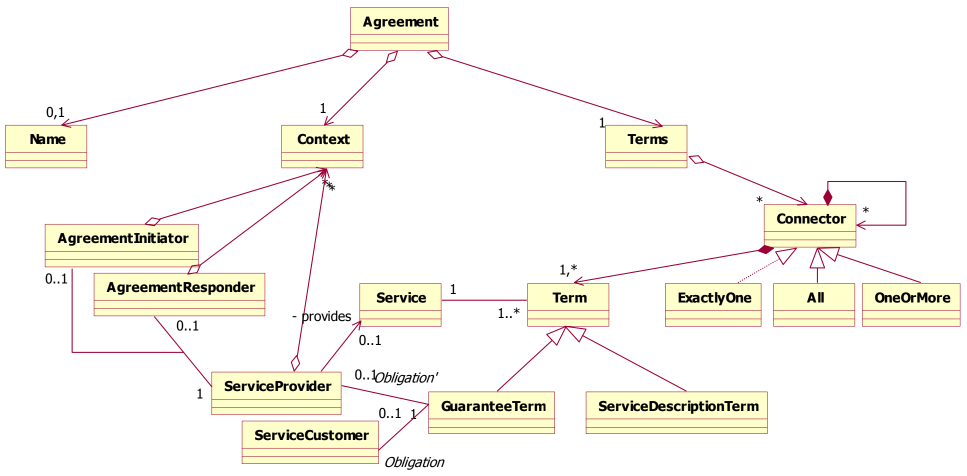

Figure 3: Agreement model.

An agreement has 3 structural components: Its descriptive *Name*, its *Context*, which comprises those elements of an agreement that don't have the notion of obligations, and the *Terms* section, which contains the main part of the agreement. The Context contains descriptions of the Agreement Responder and the Agreement Initiator. Furthermore, it defines which one of the parties is the Service Provider and the expiry time of the agreement. The parties can add other elements to the Context. The Terms section contains the Terms grouped by *Connectors*. Connectors express whether how many of the contained terms must be fulfilled to make the agreement as a whole fulfilled. The connectors are modeled after their WS-Policy equivalents and comprise ExactlyOne, All, and OneOrMore.

The WS-Agreement model defines two term types, Service Description Terms and Guarantee Terms. Each Term describes one Service and a Service may be described by multiple Terms.

Service Description Terms primarily describe the functional aspects of a service, e.g., its interface description and the endpoint reference where it is available. A Service follows the following state model, which is exposed as the state of its terms in the Agreement resource representing it:

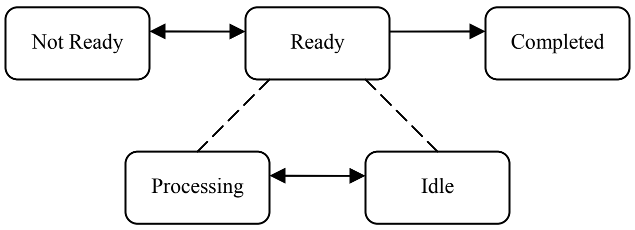

Figure 4: Service state model.

Not ready means the service cannot be used, *ready* means that it can, and potentially is used, and *completed* means that it cannot be used anymore. Each top-level state can be sub-stated. As a standard, the ready state has the sub-states *processing* and *idle*. Specific domains can introduce sub-states as they see fit, e.g., to distinguish success or unsuccessful completion. The Service Provider is obligated to deliver the services as a whole. Service Description Terms cannot be violated individually.

Individually monitorable aspects of a service that can fail independently of the functioning of the underlying "core" service are captured in Guarantee Terms. Guarantee Terms can relate to a service as a whole or to a sub-element, e.g., an operation of a Web service. They have a qualifying condition that defines the circumstances under which the guarantee applies, e.g., during business hours, a logic expression defining the actual guarantee, referred to as *Service Level Objective*, and a business value, usually a penalty for non-compliance, a reward, or a non-monetary expression of priority. Guarantee Terms follow the following state model:

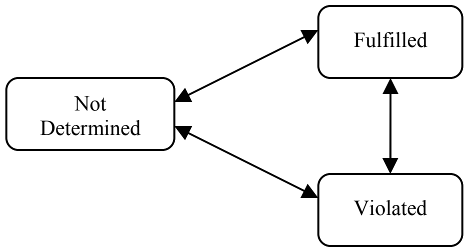

Figure 5: Guarantee term state model.

If the service is not ready or the Guarantee Term cannot be evaluated its state is *not determined*. Otherwise, if precondition and Service Level Objective (SLO) are true, or the precondition is false, the Guarantee's state is fulfilled, if precondition is true and SLO is false, the Guarantee term is violated. How long or how often a Guarantee term must be violated for a penalty to apply is defined in the details of the business value part. This state model can also be extended to the needs of a particular domain by sub-stating.

The agreement model of the WS-Agreement specification provides a high-level structure and state models that can be adapted to the needs of a specific application environment by additions to the Context, introduction of new term types and specialization of the state models associated to terms. The representation of the model in an agreement offer or agreement template as well as the specific mechanisms to apply domain-specific extensions are discussed in the Language section.

5 Offer and Template Language

The WS-Agreement language defines the representation of offers that an Agreement Initiator submits to an Agreement Responder and of agreement templates. As discussed above, the agreement model is extensible and, hence, the WS-Agreement language does not provide a complete syntax to describe every type of content in an agreement. It provides, however, the main concepts of an agreement and language elements to describe the main elements at a top level. The details of agreement offers can then be filled with other languages as seen fit. For example, WS-Agreement may contain a WSDL definition to define an interface or use a domain-specific expression language to define a response time guarantee. By being flexible in including existing languages into the overall model and syntax of WS-Agreement, it becomes useful to a wide range of applications while still providing enough structure to develop applications-independent middleware.

5.1 Agreement offer structure

A WS-Agreement Offer has three main elements:

1. An *ID* and a descriptive *Name*.
2. A *Context* element defining those parts of an agreement that don't have the character of rights and obligations, namely the parties to the agreement, which party has the role of a service provider, the ID of the template that was used to create it, the agreement's expiration time, and other elements of the agreement that have the character of definitions.
3. The core part of the Agreement Offer is the *Terms* section. It defines the obligations of the two parties to the agreement. The specification defines two types of terms: *Service Description Terms* and *Guarantee Terms*. Terms are related using *Term Compositors*. While the WS-Agreement specification defines these two types of terms, additional ones can be added by users of WS-Agreement.

Figure 6 outlines the structure of an Agreement Offer XML document.

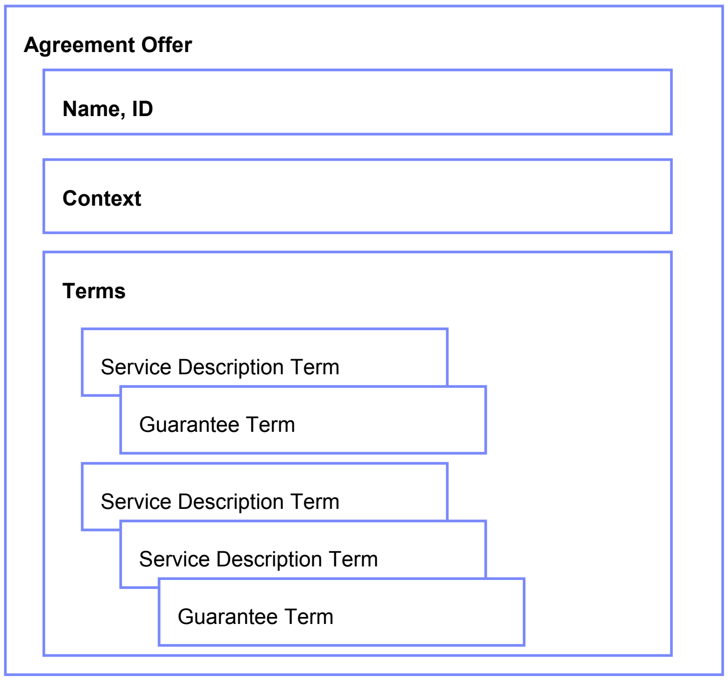

Figure 6: WS-Agreement offer structure.

The following example illustrates the XML structure of the content of an Agreement Offer:

```
<wsag:AgreementOffer AgreementId=”SettlementCapacity123”>
  <wsag:Name>SupplementalAgreementInDecember</wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator>http://www.abroker.com/</wsag:AgreementInitiator>
    <wsag:AgreementResponder> http://www.thisclearinghouse.com/</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:ExpirationTime>2005-11-30T14:00:00.000-05:00</wsag:ExpirationTime>
    …
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceDescriptionTerm name=”s1” …> … </wsag:ServiceDescriptionTerm>
      <wsag:ServiceDescriptionTerm name=”s2” …> … </wsag:ServiceDescriptionTerm>
      <wsag:ExactlyOne>
        <wsag:GuaranteeTerm name=”g1” …> … </wsag:GuaranteeTerm>
        <wsag:GuaranteeTerm name=”g2” …> … </wsag:GuaranteeTerm>
      </wsag:ExactlyOne>
    …
    </wsag:All>
  </wsag:Terms>
</wsag:AgreementOffer>
```

The Agreement Offer has the ID ClearingCapacity123, which must be unique between the parties. The Name element can illustrate further what is meant. The example Context contains the Agreement Initiator and Agreement Provider, which can be described by any content. URIs are convenient. In our example, the Agreement Responder is the Service Provider. The expiration time is given in the XML date-time format.

An Agreement Offer can contain any number of terms. The term compositors structuring them are equivalent to the WS-Policy compositors All, ExactlyOne and OneOrMore. In our example, all Service Description Terms and one of the Guarantee Terms must be observed.

5.1.1 Service Description Terms

Service Description Terms describe the services that the Service Provider will render to the Service Customer. This means the Service Provider, as defined in the Context, is liable to deliver what is promised in all the Service Description Terms. Multiple Service Description Terms can be used to describe different aspects of a service, e.g., one terms for its WSDL, one for an Endpoint Reference where the service will be available and one for additional policy information that is not contained in the WSDL. The objective of Service Description Terms is to specify between the two parties to the agreement what services will be rendered by the Service Provider.

A Service Description Term has a Name and a Service Name attribute to indicate which service it describes, as the following example snippet outlines.

```
<wsag:ServiceDescriptionTerm Name=”SettlementServiceInterface”
                             ServiceName=”SettlementRequest”>
  <wsdl:Definition …>
    …
  </wsdl:Definition>
</wsag:ServiceDescriptionTerm>
```

A Service Description term may contain any content, as the WSDL definition in this example.

A special type of Service Description Term is the *Service Reference*. It contains a pointer or a handle to the service in question or a pointer to a description of a service, rather than including the description into the agreement. The following example outlines a Service Reference containing an Endpoint Reference to a Web service:

```
<wsag:ServiceReference Name=”SettlementServiceReference”
                       ServiceName=”SettlementRequest”>
  <wsa:EndpointReference>
    <wsa:Address>http://www.thisclearinghouse.com:9090/services/settlement/</wsa:Address>
    <wsa:ReferenceProperties>
      <clearing:Account>abroker.com</clearing:Account>
    </wsa:ReferenceProperties>
  </wsa:EndpointReference>
</wsag:ServiceDescriptionTerm>
```

In this example, the service reference contains an Endpoint Reference according to the WS-Addressing specification. It points to the URL of the service at the clearing house’s service delivery system and uses the brokers name as a reference property.

A further special type of Service Description Term is the *Service Properties* term. Service properties are aspects of a service that can be measured and may be referred to by a Guarantee Term. In our clearing house example, this includes the average response time or the availability. The purpose of a Service Properties definition is to clarify to what a particular service property relates to and what metric it represents. The following example illustrates the use of Service Properties:

```
<wsag:ServiceProperties Name=”ClearingServiceProperties”
                        ServiceName=”ClearingRequest”>
  <wsag:Variable Name=”RequestRate” Metric=”clearing:RequestsPerMinute”>
    <wsag:Location>//wsag:ServiceDescriptionTerm/[@Name=”ClearingServiceReference]
    </wsag:Location>
  </wsag:Variable>
  <wsag:Variable Name=”AverageResponseTime”
                 Metric=”clearing:ResponseTimeAveragePerMinute”>
    <wsag:Location>//wsag:ServiceDescriptionTerm/[@Name=”ClearingServiceReference]
    </wsag:Location>
  </wsag:Variable>
</wsag:ServiceDescriptionTerm>
```

Each Variable definition represents a service property. It is given a unique name and a metric that is commonly understood by the parties to the agreement. In our example, we assume that the clearing house establishes a set of metrics that it can measure and it is willing to include in guarantees, such as the clearing:RequestsPerMinute and clearing:ResponseTimeAveragePerMinute metrics. The Location element contains XPath expressions that point to contents of Service Description Termsn and defines to what part of the service the variable relates to. In the example it relates to the Service Reference pointing to the clearing service. The granularity of a service is often not sufficient. We want to be able to distinguish, for example, response times for different operations of a service. In this case, we can point to a specific operation of a service in its WSDL definition.

5.1.2 Guarantee Terms

One key motivation to enter SLAs is to acquire service capacity associate with specific performance guarantees. A Guarantee Term defines an individually measurable guarantee that can be fulfilled or violated.

A Guarantee Term has the following elements:

- The *Service Scope* defines to which service or part of a service the guarantee applies. This can be a Web service Endpoint as a whole, e.g., for guarantees on availability, or individual operations of a service, which is more suitable for response time guarantees. Referring to sub-elements of a service requires additional language elements not covered by WS-Agreement.
- The *Qualifying Condition* contains a Boolean expression that defines under which condition the guarantee applies. Again, the parties to the agreement use a suitable language applicable in their domain.
- The *Service Level Objective* defines what is guaranteed, using a suitable expression language.
- The *Business Value List* defines the valuation of this guarantee. In cross-organizational scenarios, Penalty and Reward are the most common forms of expressing value. There are also the options to express importance abstractly as an ordinal number or to express the relative importance of guarantees among the Guarantee Terms of an agreement. This helps to decide trade offs between guarantees if not all guarantees can be fulfilled.

Guarantees can be given by both, Service Providers and Service Customers. The obliged party is defined in each Guarantee Term. This enables a Service Provider to give guarantees if the guarantee fulfillment also depends on the performance of the Service Customer. In a high-performance computing environment, the commitment to complete a computation at a given time may depend on the Service Customer providing the data input in time as the stage-in file. This dependency can be defined as a Guarantee Term owed by the Service Customer.

Consider the following example:

```
<wsag:GuaranteeTerm wsag:Name="SettlementResponseTime">
  <wsag:ServiceScope>
    <wsag:ServiceName>SettlementService</wsag:ServiceName>
  </wsag:ServiceScope>
  <wsag:QualifyingCondition>
    <exp:And>
      <clearing:BusinessHours/>
      <exp:Less>
        <exp:Variable>RequestRate</exp:Variable>
        <exp:Value>1000</exp:Variable>
      </Less>
    </exp:And>
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
    <exp:Less>
      <exp:Variable>AverageResponseTime</exp:Variable>
      <exp:Value>5</exp:Variable>
    </exp:Less>
  </wsag:ServiceLevelObjective>
  <wsag:BusinessValueList>
    <wsag:Penalty>
      <wsag:AssessmentInterval>
        <wsag:TimeInterval>P60S</wsag:TimeInterval>
        <wsag:ValueUnit>USD</wsag:ValueUnit>
        <wsag:ValueExpr>
          <exp:Value>100</exp:Value>
        </wsag:ValueExpr>
      </wsag:AssessmentInterval>
    </wsag:Penalty>
  </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

This example defines a response time guarantee for the settlement service. It applies to the entire service. The Qualifying Condition bounds the guarantee to business hours and a request rate, as defined before in the Service Properties, less than 1000. All expressions in this Guarantee Term are represented in the PMAC expression language, which is a convenient language for logic and algebraic expressions [11] and can incorporate custom predicates and functions. In our example, we included the custom predicate BusinessHours define by the clearing house. The Service Level Objective requires the average response time to be less than 5 – seconds, as defined in the Service Properties. The Business Value List specifies a Penalty. For each assessment interval of 60 seconds (in XML Schema duration syntax) $100 is charged if the Service Level Objective is not met.

5.2 Templates

The WS-Agreement offer language provides significant flexibility to define a rich set of offers. This is particularly the case because domain-specific languages are used to represent expressions and to describe services. While this approach provides great expressiveness to the parties of an agreement, it makes it difficult for an Agreement Initiator to create an Agreement Offer that the Agreement Responder will understand, using a common subset of domain-specific languages. Also, interpreting an arbitrary agreement is complex and it is generally difficult to derive resource requirements from an arbitrary set of guarantees, even if the semantics is well understood [5].

Templates simplify the process of creating commonly understood and acceptable Agreement Offers. As discussed in the previous section, Templates are made available by an Agreement Responder at the Agreement Factory.

Agreement templates are prototype agreements with an additional section, the *Creation Constraints*, which describe how the template content may be changed by an Agreement Initiator. Creation Constraints contain individual fill-in *Items* and global *Constraints*.

- An Item has a name, a Location, which is a pointer to a particular part of the agreement prototype that can be modified, and an Item Constraint, which defines how this particular Item may be changed. Item Constraints are expressed in the XML Schema language, providing reach means of restricting contents in a standard way.
- Global Constraints relate to more than one Item and can be represented in any suitable language. In Constraints we can express that for request rates greater than 100, the average response time in an operation must be greater 2 seconds.

Figure 7 illustrates the structure of agreement templates and the concept of Location pointers into the agreement prototype.

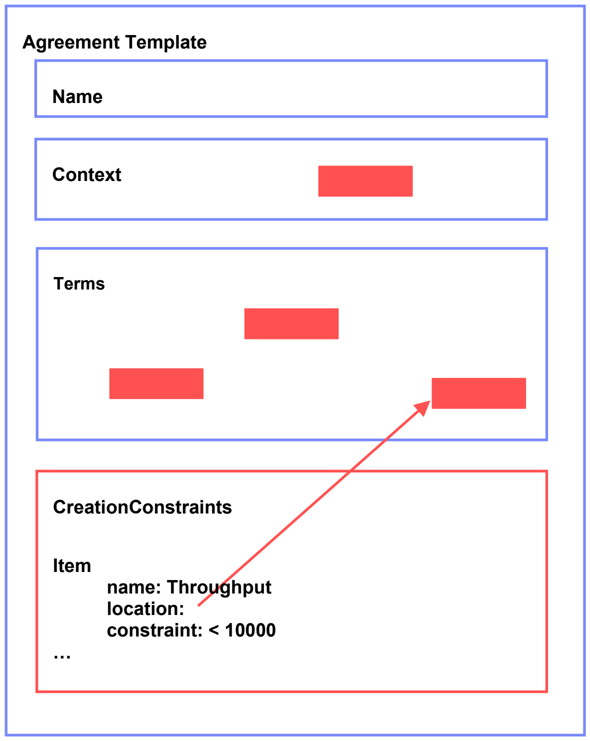

Figure 7: WS-Agreement template structure.

The Creation Constraints are the additional part following the prototype content. The red “fields” represent locations in the XML structure of the

Consider the following agreement template example:

```
<wsag:Template TemplateId=”SettlementTemplate>
  <wsag:Name></wsag:Name>
  <wsag:Context>
    <wsag:AgreementInitiator></wsag:AgreementInitiator>
    <wsag:AgreementResponder>http://www.thisclearinghouse.com/</wsag:AgreementResponder>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    …
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      …
      <wsag:GuaranteeTerm wsag:Name="SettlementResponseTime">
        <wsag:ServiceScope>
          <wsag:ServiceName>SettlementService</wsag:ServiceName>
        </wsag:ServiceScope>
        <wsag:QualifyingCondition>
         …
        </wsag:QualifyingCondition>
        <wsag:ServiceLevelObjective>
          <exp:Less>
            <exp:Variable>AverageResponseTime</exp:Variable>
            <exp:Value>3</exp:Variable>
          </exp:Less>
        </wsag:ServiceLevelObjective>
         …
      </wsag:GuaranteeTerm>
    </wsag:All>
  </wsag:Terms>
  <wsag:CreationConstraints>
    <wsag:Item name=”AgreementName”>
```

```
      <wsag:Location>/wsag:Template/wsag:Name</wsag:Location>
    </wsag:Item>
    <wsag:Item name=”Initiator”>
      <wsag:Location>/wsag:Template/wsag:Context/wsag:AgreementInitiator</wsag:Location>
    </wsag:Item>
    <wsag:Item name=”ResponseTime>
      <wsag:Location>/wsag:Template/wsag:Terms/wsag:All/wsag:GuaranteeTerm…
                     [@Name=SettlementResponseTime]/wsag:ServiceLevelObjective/exp:Less/…
                     wsag:Value/
      <wsag:Location>
      <xsd:restriction>
        <xsd:enumeration xsd:value=”1”/>
        <xsd:enumeration xsd:value=”3”/>
        <xsd:enumeration xsd:value=”10”/>
      </xsd:restriction>
    </wsag:Item>
  </wsag:CreationConstraints>
</wsag:Template>
```

The template contains the usual elements of an Agreement Offer and the Creation Constraint part. There are three Items in the Creation Constraints: The first points to the Name of the agreement offer that an Agreement Initiator must set. The next Item points to the Agreement Initiator element that must best. Finally, the ResponseTime Item points to the value of the Guarantee Term setting the threshold for the average response time. An Agreement Initiator can only choose between 1, 2 and 9, with the default set to 3 in the prototype.

5.3 Dealing with Extensions and Agreement Variety

Templates are a good mechanism to reduce choices and complexity in the agreement establishment process. Rather than analyzing an Agreement Template or Offer from scratch, parties can rely on known structures and deal with a limited parameter set, the Items. Dealing with a templates having a low number of Items enables parties to automate all or parts of their decision making and service system configuration functions.

However, in a given application domain, there might be many Service Providers or Customers offering Templates through their Agreement Factories. This may entail that Agreement Initiators that interact with multiple Agreement Responders still have to analyze the structure of template each time they encounter a new Agreement Responder. Due to the flexibility of the WS-Agreement language this significantly increases decision-making complexity and reduces the capability to automate.

The solution to this problem is more organizational than technical. A particular industry can develop a set of standard agreement templates that can be either provided centrally to all parties or published through Agreement Factories. Choosing the latter way, Agreement Responders could further restrict item and global constraints without reducing the decision complexity faced by Agreement Initiators.

6 Designing Agreement Management Interaction

The WS-Agreement specification provides a number of port types that must be configured to meet the requirements of a particular application domain. The main configuration approaches are combination, extension, and reduction, i.e. only implementing a subset of operations of a service. These approaches can be applied in any combination.

There are multiple ways to design synchronous and asynchronous agreement establishment mechanisms based on the combination of available WS-Agreement port types. The WS-Agreement factory-related Port Types are: AgreementFactory, returning an immediate decision for createAgreement calls, PendingAgreementFactory, which creates an Agreement resource but doesn’t return a decision yet, and AgreementAcceptance, which enables an Agreement Initiator to receive an accept or reject answer asynchronously. There are two port types related to Agreements: The Agreement port type offers static information, i.e., the agreement content and the references to services, and the AgreementState port type, which exposes the agreement state.

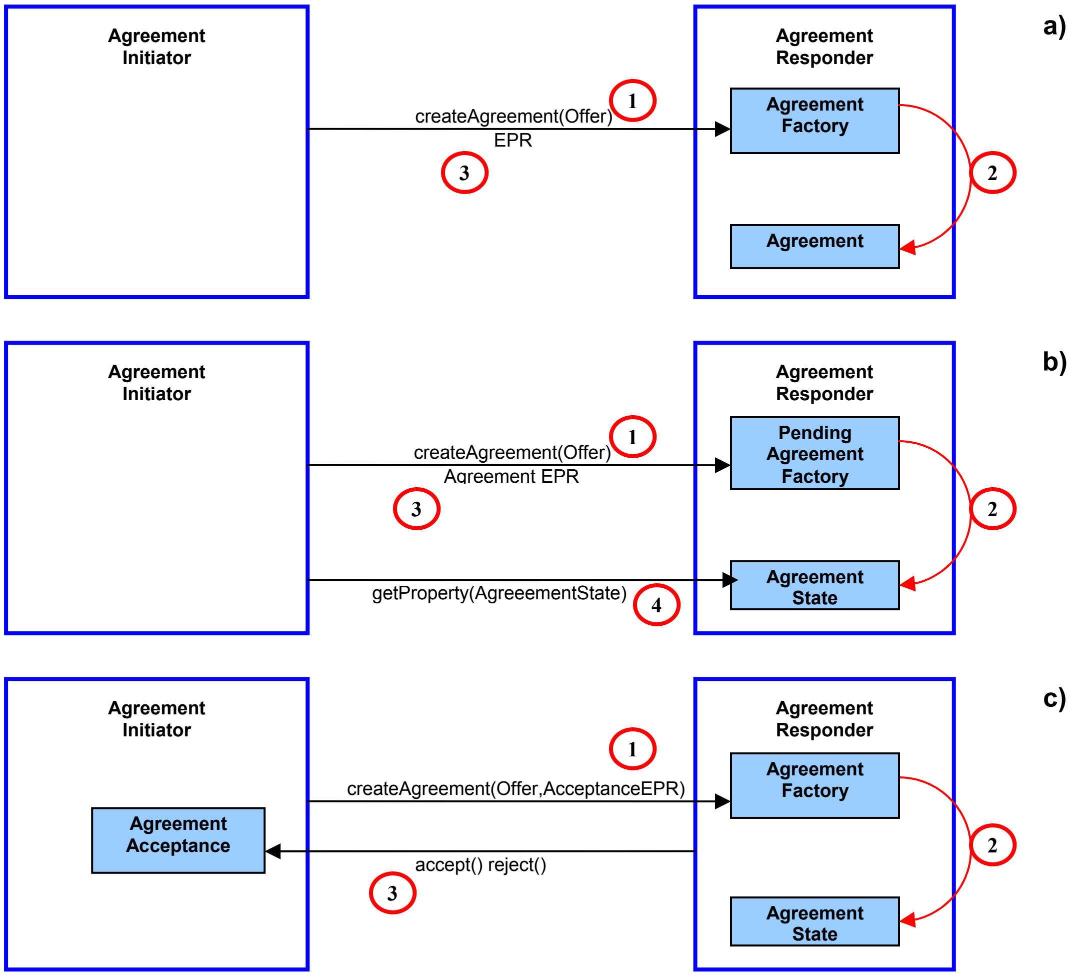

Figure 8: Agreement establishment alternatives.

Figure 8 illustrates various alternatives of agreement establishment: In alternative a), an agreement is created using a synchronous call. The approval decision is made synchronously, the agreement is established and its EPR is returned. The creation of the Agreement resource itself might be deferred, though. Alternative b) outlines a simple asynchronous protocol. The Agreement Initiator requests an asynchronous createAgreement operation. The Agreement Provider creates an Agreement State resource and returns an EPR. However, no decision is made yet and the state of the agreement is pending. The Agreement Initiator can then retrieve the state of the agreement and proceed to using it if and when it is approved. Finally, in alternative c), the Agreement Initiator can add an AgreementAcceptance EPR to its asynchronous createAgreement request to be notified by the Agreement Responder when a decision is made.

Furthermore, both Agreement Initiator and Agreement Responder can expose Agreement state port types that enable the other party to monitor each other's view of guarantee compliance. This is particularly necessary if the Agreement Initiator is the Service Provider.

In Grid environments, we often find a situation where an agreement is being made for the execution of a single large compute job. In this case, it may not be necessary to expose a separate Web service to submit jobs. One could either extend the Agreement resource by operations that receive the input data of a compute job and start the execution or to include all job-relevant information in the Agreement and interpret the create agreement operation also as the start of the service, if approved.

7 Implementing WS-Agreement based systems

As discussed in the section on Agreement-Driven Service Oriented Architectures, implementing capacity-aware, agreement-driven systems requires a set of new functions from service providers and service customers. This comprises functions that manage the agreement life-cycle for either party and functions that manage how service client requests consume acquired service capacity and how services associated service requests with agreements and treat them accordingly.

WS-Agreement standardizes interaction for agreement establishment and compliance monitoring in a domain independent way. Every WS-Agreement deployment, however, will take place in a particular application domain and will deal with domain-specific languages embedded into agreement offers and connecting to a specific Service Delivery System. Furthermore, each implementation will connect to a specific Service Delivery System or Service Client System. Implementing WS-Agreement from scratch for each Agreement Initiator and Agreement Responder, however, is tedious. Providing a domain and technology independent implementation of core features of WS-Agreement that are extensible or can be used by specific implementation parts greatly reduces the effort of implementing a WS-Agreement system.

A first implementation framework for WS-Agreement based systems is Cremona (Creation and Monitoring of Agreements) [17]. Cremona is a middleware layer that can be used to create agreements and to access agreement state at runtime. We will discuss it in more detail in this section as an example of how domain-independent parts of WS-Agreement can be implemented and can be used and extended by the specific aspects of an implementation. It has a different structure for Agreement Responders and Agreement Initiators. The design objective is to implement the synchronous WS-Agreement establishment and monitoring protocol, to make it suitable for service providers and customers, to separate domain-independent from system-specific and domain-specific components and to provide interfaces for administrative tools.

The **Agreement Responder** structure is outlined below in Figure 9, comprising the following components: The *Agreement Factory* is a domain-independent implementation of the Agreement Factory port type. The *Template Set* maintains the collection of currently valid agreement templates that initiators can use to submit createAgreement requests. The *Agreement Set* component administers the collection of agreement instances and routes requests addressed to a specific agreement endpoint reference to the corresponding agreement instance. An *Agreement Instance* exposes the terms and context of an agreement as well as the runtime status of service description and guarantee terms. The Agreement Instance uses a *Status Monitor* interface to retrieve the status of its terms. The *Status Monitor Implementation* is specific to the system providing or using the service. It accesses system instrumentation on service provider or service consumer side to gather relevant basic measurements and derives from them the status a term according to its state model. In the clearing house's case, the status monitor implementation must access the instrumentation of the application servers of the clearing house and derive term status from low-level instrumentation data, e.g., time stamps and counters. The *Decision Maker* interface is used by the Agreement Factory to decide whether to accept a createAgreement request. The decision maker implementation depends on the service role and is domain-specific. In our example scenario, the decision maker component of the clearing house must assess the resource usage at the requested time of service. It must derive the set of resources needed, i.e. HTTP servers, applications server, and database servers and make its decision dependent on whether these additional resources can be obtained at the time requested. The *Agreement Implementer* Interface is used to announce a new agreement. Its service role-specific implementation takes the necessary measures to provide or consumer a service according to the agreement, e.g., provision a system or schedule the job. Applied to the clearing house services, this means that a process to provision the set of servers and to configure central dispatchers and the backup mechanism. All objects are accessible through the Administrative Web Service Interface. Figure 9 illustrates the interaction among components processing the createAgreement request by an agreement initiator.

The interfaces Decision Maker, Agreement Implementer and Status Monitor provide the access to domain and technology-specific implementations that connect the Agreement Management to the Service Delivery or Service Client System. These implementations may bear significant complexity and may involve automated as well as manual process steps, e.g., for decision-making and provisioning.

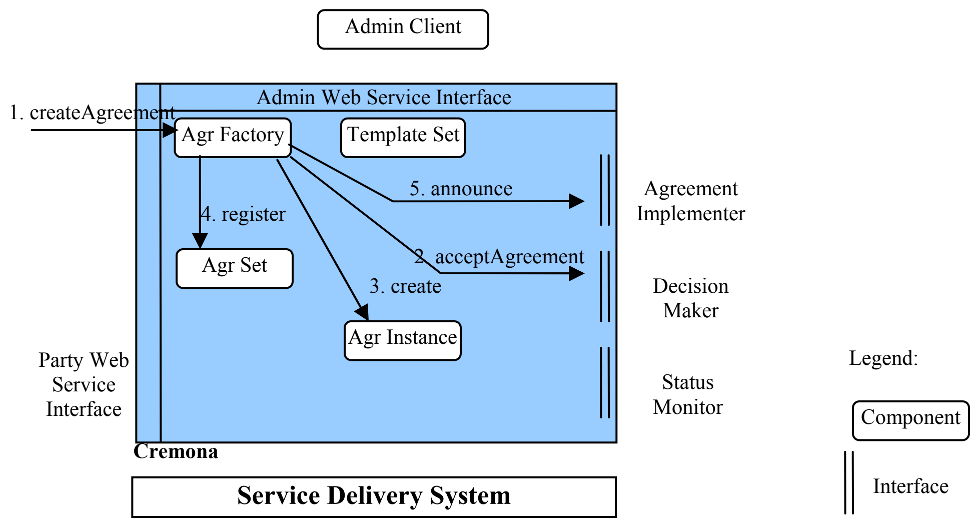

Figure 9: APRM – Agreement Responder structure, createAgreement flow.

Upon receiving a createAgreement request, the agreement factory requests the decision maker whether the agreement can be accommodated. If so, it creates the agreement instance and registers it with the agreement set. Subsequently, it is announced to the agreement implementer, which returns if the agreement is set to be used under the terms defined in the agreement. This does not require that the service is provisioned. The system must be set to be ready when the agreement requires it, which can be much later. Finally, the request is returned to the agreement initiator. Operations to retrieve templates and obtain term status and content on factory and agreement instance are implemented by simpler interaction sequences.

The Cremona components of an **Agreement Initiator** mirror those of the agreement provider and complement them with initiator-specific components. Figure 10 outlines these components. The *Agreement Initiator* component is the central element. It mediates the interaction on behalf of a component or user client that wants to create a new agreement. The *Factory Set* maintains the factories to be used. The *Agreement Set* maintains references to the agreements that the Agreement Initiator can use to claim service. *Factory Proxy* and *Agreement Instance Proxy* maintain connections to their respective counterparts on the Agreement Provider side. The *Template Processor* facilitates the creation of agreement instance documents from agreement templates. It fills in values in constraint items and validates constraints. The Agreement Implementer interface is used to publish the availability of a new agreement, equivalent to the use in the provider APRM.

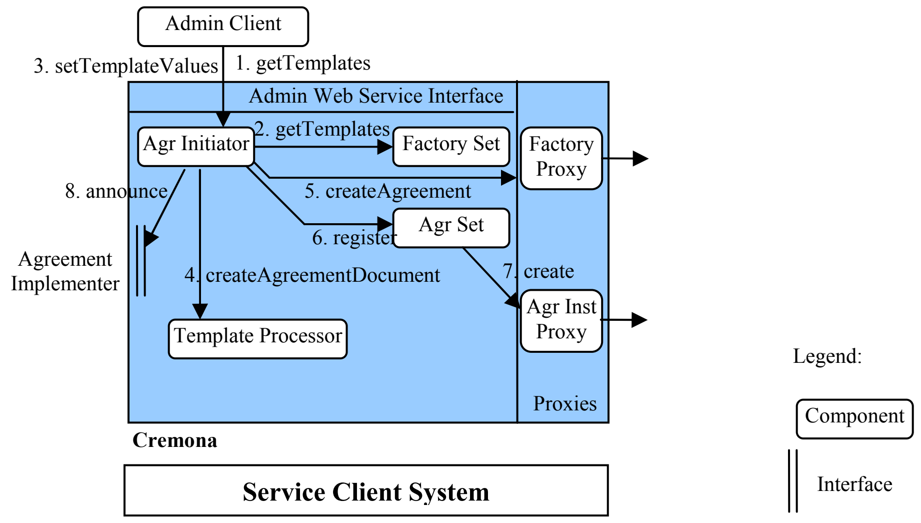

Figure 10: APRM – Agreement Initiator structure, createAgreement flow.

In the case illustrated in Figure 10, a user client requests templates, wanting to initiate a new agreement. The Agreement initiator requests the set of templates from the factory set, which in turn receives it from their respective agreement providers through the factory proxies. Having decided on the template and its values, the client submits the chosen values through the agreement initiator to the template processor, which constructs an agreement instance document. If valid, the agreement initiator invokes the proxy of the factory in question to submit a createAgreement request. If the return is positive, it registers the endpoint reference of the new agreement with the agreement set, which then creates a proxy connected to the agreement provider's agreement interface. Otherwise, the client can revise the values set in the template based on the provider's response and try it again. Finally, the new agreement is announced through the agreement implement interface whose implementation must make sure that it can be used. The agreement initiator component can also be used by a component other than a user client, i.e. an automated component, if the decision-making task to fill in a template is simple. Beyond the createAgreement flow, the Cremona components can be used to add new factories to the factory set and use the agreement proxies to query the agreement terms and their current status.

As outlined, Cremona provides implementation support for the WS-Agreement protocol and the management of WS-Agreement artifacts such as templates and agreements. Furthermore, Cremona provides interfaces that can be implemented in an implementation-specific way, e.g., to trigger provisioning when having accepted a new agreement or to implement status requests. By defining a set of narrow interfaces between the Cremona responder and initiator components and the domain-specific parts of a system, Cremona can be used in situations in which the Service Provider is Agreement Responder or Initiator.

Obviously, the set of components and interfaces and their interaction model defined by Cremona is just one first approach. Other component structures and corresponding programming models may provide implementation support beyond the one offered by Cremona.

8 Conclusion

Agreement-based Service-Oriented Architectures enable service customers to reserve service capacity ahead of the time of service use and thereby enable service providers to plan their resource consumption to meet performance requirements. This is a key requirement for scenarios of dynamic and performance sensitive Web services such as the clearing house example. Agreement-based SOAs add the notion of service capacity, agreement and organization to the traditional SOA stack. Prior to using a Web service, a service customer establishes an agreement that its clients can access a service with a service provider at a given performance level and request rate.

A key enabler for an agreement-based SOA is a standard way to represent agreements (SLAs) and conduct agreement-level interaction for creating and monitoring agreements. WS-Agreement provides a standard definition of an agreement structure that can be amended by domain-specific concepts and language elements. Furthermore, WS-Agreement defines interfaces for establishing agreements and monitoring their compliance state according to their terms.

While WS-Agreement addresses the agreement-related interaction between organizations, and implementation support, e.g., Cremona, is available, many issues related to the relationship between the agreement layer of an agreement-driven SOA and the service layer remain to be addressed. Service client systems have to manage the use of the acquired service capacity by their clients and respond to dynamic client request changes, e.g., by buying more capacity, routing capacity to different provider, and delaying requests. A service delivery systems has to manage its contractual obligations and manage the yield of its agreements, sometimes by incurring a small penalty in one agreement for a larger one on another. Foremost, though, a service provider has to derive a service delivery system from an agreement with a service customer in an efficient way. In addition, there is no established standard to claim service against an agreement. While there are multiple options such as different Endpoints for different agreements, shared service use will require some agreement identification in a SOAP header of a Web service request in a universally understood way. Finally, important issues arise in the context of domain heterogeneity. How do parties know which domain-specific language another party understands? How can a service customer effectively deal with different service providers, and vice versa. These issues of ontology and heterogeneity present further research challenges.

In summary, though, WS-Agreement provides the key enabler to capacity and performance aware agreement-based service-oriented architectures enabling meaningful inter-organizational Web services.

9 References

1. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu: Web Services Agreement Specification. Version 1.1, GGF GRAAP working Group Draft, August 15, 2005.
2. P. Bhoj, S. Singhal, S. Chutani: SLA Management in Federated Environments. In Proceedings of the Sixth IFIP/IEEE Symposium on Integrated Network Management (IM '99), Boston, MA, USA, pp.293 - 308, IEEE Publishing, May, 1999.
3. A. Dan, D. Dias, R. Kearney, T. Lau, T. Nguyen, F. Parr, M. Sachs, and H. Shaikh. Business-to-Business Integration with tpaML and a B2B Protocol Framework. *IBM Systems Journal*, 40(1), February 2001.
4. A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kübler, H. Ludwig, M. Polan, M. Spreitzer, A. Youssef. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, Vol. 43 (1), 2004.
5. A. Dan, C. Dumitrescu, M. Ripeanu: Connecting Client Objectives with Resource Capabilities: An Essential Component for Grid Service Management Infrastructures. In: *Service-Oriented Computing - ICSOC 2004, Second International Conference*, New York, NY, USA, Proceedings, pp. 57-64, ACM 2004.
6. A. Dan, H. Ludwig, G. Pacifici: Web Services Differentiation with Service Level Agreements. IBM Software Group Web Services Web site (ftp://ftp.software.ibm.com/software/ websphere/webservices/webserviceswithservicelevelsupport.pdf). 2003.
7. J. Cole, J. Derrick, Z. Milosevic, and K. Raymond: Policies in an enterprise specification. In M. Sloman (ed.). *Proceedings of the Policy Workshop*, 2001.
8. D. Dubie: Pay-as-you-go pricing picks up. In: Network World, vol. 22, No. 30, August 1, 2005. Southborough, MA, 2005.
9. Grefen, P.; Aberer, K.; Ludwig, H.; Hoffner, Y.: CrossFlow: Cross-Organziational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. In *IEEE-CS Data Engineering Bulletin*, 2001.
10. Y. Hoffner, S. Field, P. Grefen, H. Ludwig. Contract-driven creation and operation of virtual enterprises. In *Computer Networks 37*, pp. 111 - 136, Elsevier Science B.V. 2001.
11. IBM Corporation: PMAC Expression Language Users Guide. Alphaworks PMAC distribution, www.alphaworks.ibm.com, 2005.
12. ISO/IEC JTC 1/SC 7. *Information Technology - Open Distributed Processing - Reference Model - Enterprise Language: ISO/IEC 15414 | ITU-T Recommendation X.911, Committee Draft.* 8. July 1999.
13. A. Keller, H. Ludwig: The WSLA Framework – Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and System Management (11), Nr. 1, Special Issue on E-Business Management*. Plenum Publishing Corporation, March 2003.
14. R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. N. Tantawi, A. Youssef. Performance Management for Cluster Based Web Services. *IFIP/IEEE 8th International Symposium on Integrated Network Management (IM 2003)*. IFIP Conference Proceedings 246, Kluwer Academic Publisher, 2003, pp. 247-261.
15. H. Ludwig: Web Services QoS: External SLAs and Internal Policies – Or: How do we deliver what we promise? In: *Proceedings of the Web Information Systems Engineering Workshops: 1st Web Services Quality Workshop*. IEEE Computer Society, pp. 115 – 120, Rome, 2003.

16. H. Ludwig. A Conceptual Framework for Building E-Contracting Infrastructure. In R. Corchuelo, R. Wrembel, A. Ruiz-Cortez (eds.): *Technologies Supporting Business Solutions*. Nova Publishing, New York, 2003.

17. H. Ludwig, A. Dan, R. Kearney: Cremona: an architecture and library for creation and monitoring of WS-Agreements. In: *Service-Oriented Computing - ICSOC 2004, Second International Conference*, New York, NY, USA, Proceedings, pp. 65 – 74, ACM 2004.

18. H. Ludwig, A. Keller, A. Dan, R. King, R. Franck: A Service Level Agreement Language for Dynamic Electronic Services. *Electronic Commerce Research* (3), Nr. 1, pp. 43 – 59, Kluwer Academic Publishers, 2003.

19. H. Ludwig, M. Stolze. Simple Obligation and Right Model (SORM) - for the runtime management of electronic service contracts. In C. Bussler, D. Fensel, M. Orlowska, J. Yang (eds). *Web Services, E-Business, and the Semantic Web. 2nd International Workshop, WES 2003, Lecture Notes in Computer Science 3095*, Springer-Verlag, Berlin, 2003.

20. C. Marchetti, B. Pernici, P. Plebani: A quality model for e-Service based multi-channel adaptive information systems. In: *Proceedings of the Web Information Systems Engineering Workshops: 1st Web Services Quality Worksho*p. IEEE Computer Society, pp. 165 – 172, Rome, 2003.

21. E. Maximilien, M. Singh: Toward Autonomic Web Services Trust and Selection. In: *Service-Oriented Computing - ICSOC 2004, Second International Conference*, New York, NY, USA, Proceedings, pp. 212 – 221, ACM 2004.

22. Z. Milosevic, A. Barry, A. Bond, K. Raymond. Supporting business contracts in open distributed systems. *Workshop on Services in Open Distributed Systems (SDNE '95)*. Whistler, Canada, 1995.

23. M. Tian, A. Gramm, T. Naumowicz, H. Ritter, J. Schiller. A Concept for QoS Integration in Web Services. In: *Proceedings of the Web Information Systems Engineering Workshops: 1st Web Services Quality Worksho*p. IEEE Computer Society, pp. 149 – 155, Rome, 2003.

24. V. Tosic, B. Pagurek, K. Patel. WSOL – A Language for the Formal Specification of Classes of Service for Web Services. In *Proc. of ICWS'03 (International Conference on Web Services)*, pp. 375-381, CSREA Press, 2003.

25. E. Wohlstadter, S. Tai, T. Mikalsen, I. Rouvellou, P. Devanbu. GlueQoS: Middleware to Sweeten Quality-of-Service Policy Interactions. In: *Proceedings of the 26th International Conference on Software Engineering (ICSE 2004)*, IEEE, May 2004.