RC23265 (W0406-180) June 30, 2004
Computer Science

IBM Research Report

Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements

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

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements

Heiko Ludwig
IBM T. J. Watson Research Center
19, Skyline Drive
Hawthorne, NY 10532, USA
+1 914 784 7160
hludwig@us.ibm.com

Asit Dan
IBM T. J. Watson Research Center
19, Skyline Drive
Hawthorne, NY 10532, USA
asit@us.ibm.com

Robert Kearney
IBM T. J. Watson Research Center
19, Skyline Drive
Hawthorne, NY 10532, USA
firefly@us.ibm.com

ABSTRACT
Using services across domain boundaries, be they organizations or self-managing components of large distributed systems, requires the setup of an agreement between the parties involved, defining the terms of the service including interfaces, security and Quality of Service (QoS) properties. In an on-demand environment in which services are contracted on a short notice, the establishment of an agreement as well as the setup of agreement-fulfilling and monitoring systems of the parties involved must be spontaneous and, partially, automated. WS-Agreement is a standardization effort being conducted in the Global Grid Forum defining a simple agreement establishment protocol, an XML-representation of agreements and agreement templates as well as a runtime agreement monitoring interface, based on the WSRF set of standards.

WS-Agreement standardizes the interaction between the organizational domains. In addition, providers require an infrastructure to manage agreement templates, implement the interfaces, check availability of service capacity and expose agreement states at runtime. Also, agreement requesters need infrastructure to read templates, fill in templates to create suitable agreements, and monitor agreement state at runtime. Cremona (Creation and Monitoring of Agreements) proposes an architecture for the WS-Agreement-implementing infrastructure. In addition, the Cremona Java Library implements the WS-Agreement interfaces, provides management functionality for agreement templates and instances, and defines abstractions of service-providing systems that can be implemented in a domain-specific environment.

Categories and Subject Descriptors
C.2.4 **[Distributed Systems]** *Distributed applications, Client/server*
C.2.6 **[Internetworking]** *Standards*
C.4 [**Performance of Systems**]

General Terms
Management, Measurement, Performance, Reliability, Experimentation, Standardization, Legal Aspects.

Keywords
WS-Agreement, Web service, Grid service, contract, template, contract management, quality of service.

1. INTRODUCTION
The use of 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 a client, e.g., the request rate per minute in the case of a Web service. Hence, a service client and a provider must agree on when a client can access a service at a particular QoS level for a given rate of usage. Based on this agreement, a service provider can allocate the necessary resources to live up to the QoS guarantees. Hence, the traditional publish-find-bind approach is not sufficient for services with *customized* quality guarantees.

The notion of a "service" covers a broad spectrum. Depending on the environment, a service may be the processing of a compute-intensive job, a Web service being accessible via SOAP over HTTP, network bandwidth provided or a combination of services.

The WS-Agreement specification draft, which is being developed in the Global Grid Forum's GRAAP working group, defines a standard way of establishing agreement between a service provider and a service customer. The terms service provider and customer, however, comprise different autonomous components of a system as well as different organizations. Like many other standards in the environment of service-oriented architectures it is meant to apply both within an organization and in a business to business scenario.

Using the WS-Agreement specification, the publish-find-bind approach can be extended by an agreement step that facilitates providing QoS guarantees by

1. enabling the client to state its service capacity and QoS needs,
2. enabling the provider to derive resource requirements for the requested QoS level and capacity, additionally, prioritize allocation of resources when enough resources are not available to satisfy all requests, and

3. empowering the provider to accept or reject a request by a client based on the resource situation at the time the client requests the service.

Furthermore, WS-Agreement defines a runtime interface to monitor guarantee compliance, enabling service customers and providers to take agreement-level actions, e.g., changing to another provider or extending agreements if more capacity is needed.

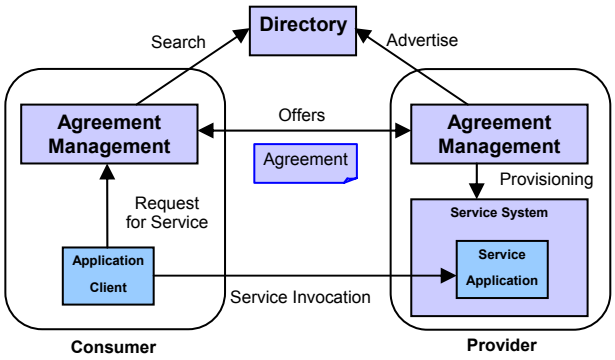

Figure 1: Binding by Agreement.

The figure shows an agreement management (AgM) function involved before a service is being invoked, as introduced in [12]. An application client looking for a service requests its AgM for guidance which service to use. The AgM may then negotiate an agreement with potential providers, which it may have found using a directory. If and when a consumer's AgM has reached agreement, it provides the application client with the claiming information it needs to use the service on the agreed terms. A consumer's AgM may also "bulk-buy" services in one agreement and then hand out claiming information to application clients without making new agreements. This is a likely scenario for "small" services such as financial information Web services due to agreement establishment overhead. The decision-making involved in closing agreements is often difficult to automate. Hence, the AgM will also often comprise a buying and selling client, beyond the middleware aspects of the AM.

To establish contact, the provider can advertise the kinds of service it offers and exposes the interface to its AgM to potential consumers to submit offers. The provider AgM can analyze its prior agreements and its system's capacity to decide whether to accept or reject an offer or to counter-propose. If agreement is reached, the service provider provisions its service system for use by the client on agreed terms. In some cases, service consumers might advertise their need and providers take the initiative in establishing an agreement. The agreement compliance is monitored at runtime.

The WS-Agreement specification draft defines the interaction among a provider and a consumer AgM at binding time and, for purposes of compliance monitoring, at runtime. However, this interaction must be backed up by an infrastructure that enables a provider to offer services by agreement, enter agreements aware of resource situation and manage resources at runtime. Likewise, on the customer's side, infrastructure is needed to facilitate agreement-based service binding and monitoring at runtime.

The objective of the paper is to propose an agreement management architecture for customer and provider infrastructure, Cremona, that facilitates agreement-based service binding for a variety of types of services. The environment to provide and consume a service depends on the particular type of service and system choices that both parties make. The objective of the Cremona library is to implement those parts of the Cremona architecture that are independent of the particular application domain.

The paper is structured as follows: To understand the required level of abstraction needed for the Cremona architecture, section 2 introduces a set of example service scenarios. Section 3 provides an overview of the WS-Agreement specification draft. In section 4 we outline agreement-driven management to scope the Cremona architecture. The Cremona architecture and library are discussed in Sections 5 and 6, respectively. Finally, we conclude with a summary, related work and an outlook on future work.

2. SERVICE EXAMPLES

WS-Agreement is applicable to a wide variety of service environments, both within and across business organizations, spanning applications such as job scheduling, resource reservation, and web services.

It is important to understand the differences amongst the variety of services, their system environments and their binding mechanism to find the right level of abstraction for the elements of the Cremona architecture. The following set of cases illustrates this.

Resource Usage Agreement in Job Scheduling: Here, the client logic is a job submission system or a workflow execution system that uses a job scheduling environment for task execution. In order to provide a guarantee on resource availability to meet a job completion time objective and/or other execution environment requirement, an agreement is established by the job submission system with the execution environment. This may involve advance resource reservation for a subsequent job submission, setting up resource agreement for repeated job submissions or job submission with individualized resource request via individualized agreement. The scheduling system prioritizes (statically or dynamically) execution of pending jobs, matches resource requirements and/or dynamically allocate resources in order to meet agreement objectives. Within this paradigm, there are many nuanced scenarios. For example, the agreement may include guarantees on aggregate resource usage across many jobs. Also, the job scheduling system can be a broker and route to various end systems based on the resource usage agreement between the broker and the end systems.

Service Agreement: Again, this is a very typical use case of service invocation (i.e., transactional workload) with agreement on average response time for up to a pre-specified throughput level [6]. This can represent application outsourcing in a cross-organizational context. The provider logic must address controlling prioritization of service invocation (and/or dynamic resource allocation for the underlying execution environment) to manage the response time objective associated with individual services and/or user(s) as specified in agreements.

In distributed data center example, such agreements may be used across data centers to shift transactional workload, where the client logic represents a data center workload management system, which upon detecting a load surge or degradation in average response time, creates agreement with another data center for routing this workload.

Service Agreements across Resource Managers: Service agreements can also be used to manage complex large scale distributed environments consisting of many resource managers, managing different types of resources. Each resource/service manager is autonomous in managing its local resources. For example, an application server may rely on predictable behavior of network, storage or database services. A workload manager detects need for better storage or database response time as part of its management loop and establishes or modifies the corresponding agreement. Note that storage/database service may be shared across multiple application servers (transactional and other types).

The Cremona architecture and libraries are to be used in all three types of scenarios, where services can vary from providing client requested resource environment to providing Quality of Service for service invocation such as web services. The actual management of underlying resources is very different in these different scenarios. We will revisit this in Section 4 in describing Cremona architecture.

In the above scenarios, we did not identify whether or not a provider and consumer belongs to the same business organization. This is to emphasize the fact that the agreement is of technical nature involving automated interaction between service provider and consumer systems, rather than a business relationship. Agreements crossing business organization boundary raises many issues on trust, pricing, auditing and monitoring. Hence, for cross-organizational interaction for outsourcing or resource sharing, we assume a pre-established business relationship, allowing dynamic creation of technical agreement.

3. WS-AGREEMENT OVERVIEW

The objective of the WS-Agreement specification is to provide standard means to establish and monitor agreements on services independent of a particular application domain [1]. The specification draft comprises three major elements:

1. A description format for agreement templates and agreements;
2. A basic protocol for establishing agreements, and
3. An interface specification to monitor agreements at runtime.

This section provides a brief overview that focuses on those elements of the specifications that were particularly relevant to the design of the Cremona architecture.

3.1 Interfaces and Interactions

Agreements are set up between two roles, the *agreement initiator* and the *agreement provider*. These roles are independent of the roles of service provider and service consumer. Figure 2 outlines the main interaction structure.

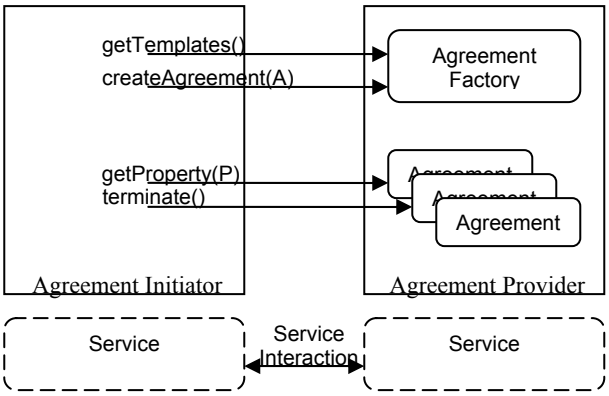

Figure 2: WS-Agreement Roles and Interactions

An Agreement Provider exposes an interface of an Agreement Factory, which offers an operation to create an agreement A and an operation to retrieve a set of agreement templates proposed by the agreement provider. Agreement templates are agreements with fields to be filled in, much like a pre-printed car sales contract. It helps an agreement initiator to create agreements that the agreement provider can understand and accept. The *createAgreement* operation returns the agreement, if accepted.

If the createAgreement operation succeeds, an agreement instance is created. The agreement instance exposes the terms of the agreement as properties P that can be queried. In addition, each term has a runtime state that can be introspected. For example, a guarantee can be *not determined* (cannot be measured), *fulfilled* or *violated*. An agreement initiator can also submit a *terminate* operation, requesting the agreement not to be in effect anymore. The Agreement provider either terminates service activity (i.e., providing or consuming) or rejects the termination. In addition to the agreement provider the initiator can also choose to provide an agreement instance to the other party.

Both, Agreement Factory and Agreement instances expose their interfaces as resources according to the Web Services Resource Framework (WSRF) [8], [7]. Terms and term states are properties that can be accessed with the standard get-operations.

3.2 Agreement and Template Content Model

Both, agreements and agreement templates are specified in an XML-based language and have the following structure:

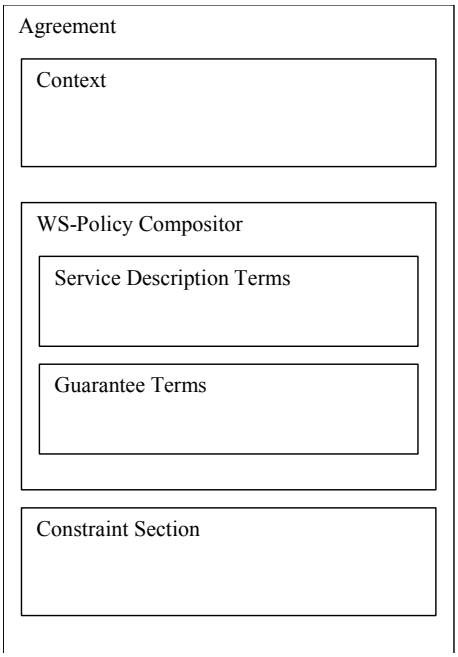

Figure 3: Structure of an Agreement document, based on [1], figure 2.

An agreement and an agreement template consist of a context section, the agreement terms and the constraint section. The agreement context contains the definition of the parties, including the end-point references of their planned agreement instances and their roles, and related prior agreements. The agreement terms represent contractual **obligations** and include description of the service as well as the specific guarantees given. A *service description term* (SDT) can be a reference to an existing service, a domain specific description of a service (e.g., a job using the Job Submission Description Language, a data service using Data Access and Integration Services, etc.), or a set of observable properties of the service. Multiple SDTs can describe different aspects of the same service. A *guarantee term* on the other hand specifies service level objectives as an expression over properties of the service, an optional qualifying condition under which objectives are to be met, and an associated business value specifying the importance of meeting these objectives. All terms can be composed using the compositors of the WS-Policy specification [3]. Finally, an agreement template is defined by adding constraints to be met in creating the agreement. A constraint comprises a named pointer to an XML element in the context or term sections of the agreement and a constraint expression defining the set of eligible values that can be filled in at this position.

The WS-Agreement specification only defines the overall structure of agreements and agreement templates. This outer structure must be complemented by means of expression suitable for a particular domain. For example, a guarantee terms is defined as comprising the elements scope, qualifying condition, service level objective, and business value. There are no language elements defined to specify a service level objective. Parties have to choose a suitable condition language to express the logic expression defining a service level objective, e.g., the OMG Object Constraint Language (OCL).

3.3 Discussion

WS-Agreement provides a simple protocol to establish agreements. This draft does not cover advertising a service nor does it comprise more advanced means of negotiation. The negotiation dialog can be either mapped onto the createAgreement operation or can be conducted outside the covered scope. In addition, WS-Agreement does not cover how to access a service according to an agreement. While in the case of job submission, no further activity might be needed from the service consumer, transactional services entail a client application to send SOAP messages to URL and it has to be agreed upon how to label messages as being subject to an agreement, for example by adding a contract ID to SOAP headers.

The agreement content definition provides an umbrella structure that must be complemented by other language to describe a service or to define guarantees. On the one hand, this is convenient since it allows parties to use existing languages such as WSDL to describe particular aspects of a service. On the other hand, parties must be able to deal with a variety of specifications. Hence, agreement initiators will rely on agreement templates published by agreement providers to create agreements that will be understood by providers.

Although WS-Agreement is open to which party will be agreement provider, in many cases it appears natural that the service provider will take this role. There is a tight relationship between the service systems that a service provider can configure and the agreement templates it can offer.

4. AGREEMENT-DRIVEN MANAGEMENT

The WS-Agreement specification covers the direct interaction between the two parties. This agreement-level interaction must be integrated in the overall life-cycle management of an offered or requested agreement and it must be integrated with the service-implementing and service-consuming systems. Each party must implement a number of functions, dependent on the role it takes in the agreement protocol, agreement initiator or agreement provider, and in the service relationship, service provider or service consumer.

Supporting WS-Agreement protocol and agreement-based service management, however, requires functions that go beyond core functions for retrieving templates, creation of an agreement and agreement state monitoring. These include tooling and setting up in both consumers and providers prior to creation of an agreement, management logic in determining agreement to be established, and finally managing the service including dynamic provisioning of resources in order to meet objectives specified in an agreement. Without reusable libraries, and management middleware, these functions need to be implemented for every service that requires QoS assurance, either in agreement factory or in the service itself. Cremona provides a layered agreement management architecture and a library for integrating system or domain-specific functions.

Examining the differing scenarios and execution environments presented earlier, the required functionalities can be classified into three categories (see Figure 4):

- Core functions based on web services and WSRF for supporting WS-Agreement protocol;

- A set of libraries, tools, agreement management and monitoring functions that are common to all environments for agreement based service management;
- Domain specific functions in managing services (e.g., job scheduling, workload management, etc.).

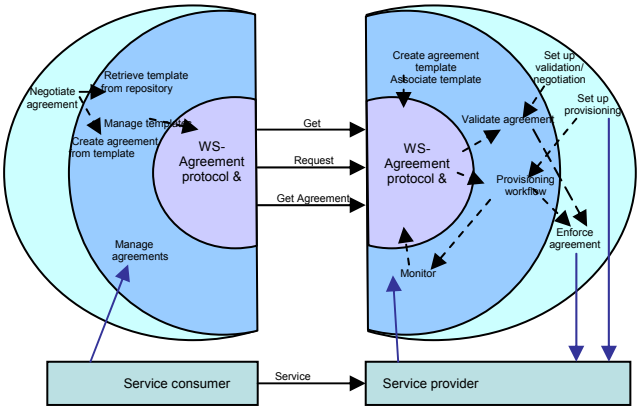

Figure 4: Layers of agreement management functions.

Figure 4 illustrates this layering of functionalities. At the very core are a set of functionalities (to be detailed later in Section 5) for supporting WS-Agreement protocol. The domain-independent middle core interfaces with the protocol layer and the domain specific logic. It provides management software for all phases of agreement life-cycle. Prior to creation of an agreement, for the provider side, it includes tools for creating new templates, associating a template with a factory, and managing templates. For the client side, it provides functions for managing templates retrieved from various factories. During agreement creation, it supports programmatic creation of an agreement document from a template where the agreement parameters are passed by domain specific logic, and provides a framework for evaluating an agreement based on policy (which in turn relies on current or projected capacity information provided by domain dependent functions). It also provides management software for provisioning that can be customized by domain specific information. Subsequent to agreement creation, it monitors agreements based on formal specification of agreement objectives and service state information.

In this paper, we focus on the core layer and a subset of these middle core functions.

5. Cremona Architecture

The Cremona architecture separates multiple layers of agreement management, orthogonal to the agreement management functions:

The collective of functions associated with an agreement protocol role, initiator or provider, is the **Agreement Protocol Role Management** (APRM). It comprises, on the agreement provider side, the agreement factory, the agreement instance implementations, the Web services container in which factory and instances are located and interfaces to an agreement template repository, decision-making functionality for createAgreement requests and the current state of terms. On the agreement customer side, it comprises proxy functions to interact with an agreement factory and created agreements, template processing functions to create agreement instance document from templates, and interfaces to components initiating agreement establishment, to functions deciding on how to fill an agreement template, and to guarantee monitors.

The **Agreement Service Role Management** (ASRM) is the collective of functions that deals with a party's role in the service relationship, provider or consumer, and connects it to the service system. On the service provider's side, this includes the mapping of agreements to provisioning specifications and other input to the service-implementing system – the agreement implementation plan [11]. In addition, the ASRM includes tools to create agreement templates and associated agreement implementation plan, decision-making functionality to admit new agreements, taking into account the current system status and committed capacity, and monitors that map that state of a service-implementing system to guarantee status. On service consumer's side, the ASRM provides functions to assign agreements to application client requests, decision-making functions that request or accept agreements based on forecast demand and agreement compliance monitors.

Strategic Agreement Management (SAM) refers to management functions beyond the scope of individual templates and agreements. This includes policy related to which other parties are chosen to enter agreements with, whether one or more providers are chosen, a provider's yield management policy, whether to act as agreement provider or initiator for a particular type of service and so forth.

All levels of agreement management are over and above the service system.

An import issue is the design of **interfaces** that the application client of a service consumer uses to ask for a service. This issue relates to the extent to which the application client of the service consumer is aware of agreement management, on the one hand, and to the variety of services that can be contracted using the AgM, on the other.

We can distinguish the case of a client stating requirements on a service and receiving claiming information from a case were a client is actively involved in selecting parties, agreement templates and deciding on offers. In the first case, a client uses the agreement request interface and is not aware of the structure of agreement management. In the second case, a client uses the administrative interface to the AgM. In many cases, applications setting up agreements and requesting service will be different but in some scenarios, such as submission of compute-intensive jobs, an integrated client may be a better solution.

Describing the service a client needs is generally a very domain-specific problem. The general agreement request interface is very general in the format of requests it accepts. In a particular domain, conventions must be established how to request a service. In a Web services scenario, for example, a client may request a port type in a WSDL definition. The resulting claiming information could be a WSDL containing binding information and an endpoint reference. While variety is a problem in general, a typical domain might not have so much variety, also because application clients are typically not written in a very flexible. A financial services company might have a foreign exchange information application using the same port type. The AgM might have agreements with two different providers on different QoS levels and returns the binding that suits best.

5.1 Agreement Protocol Role Management

The APRM is a middleware layer that can be used to create agreements and to access agreement state at runtime. The APRM has a different structure for agreement providers and for agreement initiators. The design objective is to implement the WS-Agreement protocol, to make it suitable for service providers and customer, to separate domain-independent from system-specific and domain-specific components, the outer layers of Figure 4, and to provide interfaces to administrative tools.

The **agreement provider** structure is outlined below in Figure 5. The APRM of an agreement provider comprises the following components:

- The *Agreement Factory* is a domain-independent implementation of the Agreement Factory interface.
- 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. It also routes status requests addressed using an 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 aggregate status of a SDT or a guarantee term. For example, in the job submission case, a status monitor of a service provider accesses the schedule to find out if a job is still waiting, in process or completed and replies correspondingly. In the case of a guarantee relating to a completion time, the status monitor would see if it is fulfilled in case of the job completed or violated or not determined if the SDT status is not completed.
- 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. It must be implemented in the ASRM.
- 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.

All objects are accessible through the Administrative Web Service Interface and by ASRM components.

Figure 5 illustrates the interaction among components processing the createAgreement request by an agreement initiator.

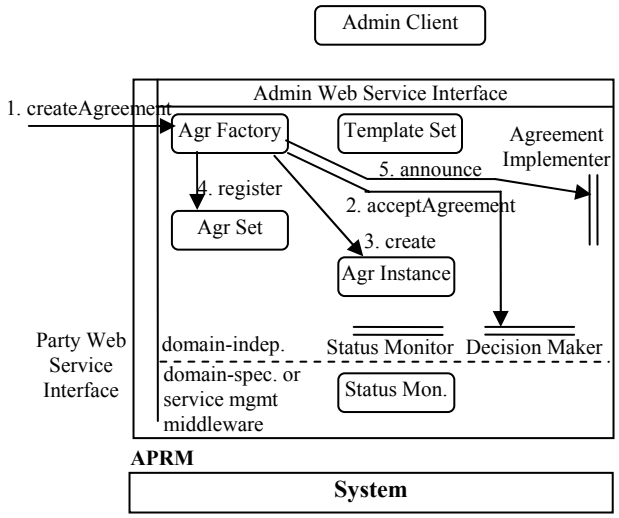

Figure 5: APRM - provider structure, createAgreement flow.

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 APRM of an **agreement initiator** mirrors the components of the agreement provider APRM and complements it with initiator-specific components. Figure 6 outlines these components.

- The *Agreement Initiator* component is the central element of the initiator APRM. It mediates the interaction on behalf of a component or user client that wants to create a new agreement.
- The *Factory Set* maintains to factories to be used.
- The *Agreement Set* maintains references to the agreements that the initiator can use.
- *Factory Proxy* and *Agreement Instance Proxy* maintain connections to their respective counterparts on the provider side APRM.
- 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.

The initiator APRM does not contain domain-specific components. Figure 6 illustrates the use of initiator APRM components

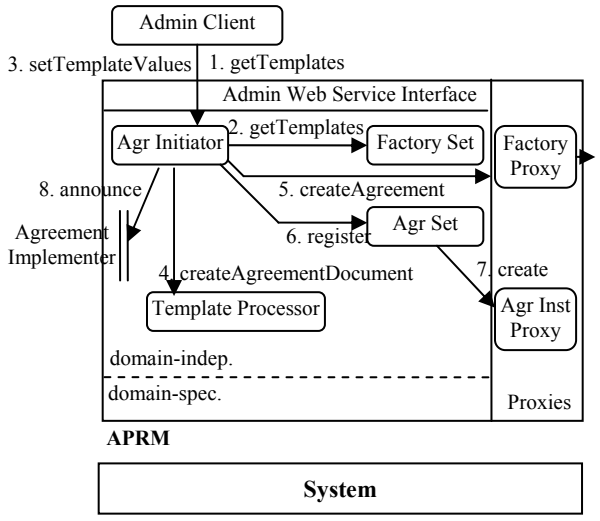

Figure 6: APRM – initiator structure, createAgreement flow.

In the case illustrated above, 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 announce 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. This might often be the case for standardized job scheduling agreements where mainly job specification information, e.g., in the Job Submission Description Language (JSDL) must be filled in and guarantees mainly relate to completion time and resource availability.

Beyond the createAgreement flow, the initiator APRM 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.

5.2 Agreement Service Role Management

The ASRM builds on the APRM and relates it to the service-implementing or service-consuming system. It provides the components required to trigger the agreement-driven provisioning of a service and to monitor compliance at runtime as well as components that direct a client to a suitable contracted when needing a service. The ASRM is different for the service provider and the service consumer. Service provider and service consumer ASRM must be able to build on both, an agreement provider and an agreement initiator APRM.

The **Service Provider ASRM** is structured as illustrated in Figure 7. Since by its very nature it interacts with the service-implementing system, many components are domain and system-specific. However, some components can be implemented in a domain-independent way if they are driven by explicit specifications of behavior such as workflows or policies, which can be run on a generic execution engine. We call a specification that is derived from an agreement and that can be interpreted by a provisioning system an *Agreement Implementation Plan* (AIP). For example, an AIP could be a BPEL4WS specification of a provisioning workflow. The Cremona ASRM architecture defines a set of interfaces that can be implemented either way, domain-specifically or by generic components driven by an AIP.

- *Admission Control* implements the Decision Maker interface of the agreement provider APRM. It can interact with the service-implementing system to find out whether capacity is available. To do so, it must translate the SDTs and guarantees of the agreement requested into specific resource requirements. Typically, it is implemented uses the capacity planning component for this purpose.
- In case the service provider ASRM uses an agreement initiator APRM, the *Load Acquisition* component triggers agreement creation through the Agreement Initiator if spare capacity can be solicited.
- If a new Agreement is created, the Agreement Implementer Interface is invoked. In case of an AIP-driven provisioning system, the *AIP Generator* implements the Agreement Implementer interface. For each agreement template an AIP is defined. It has a template structure similarly to agreement templates. The template body contains the AIP specification with "holes" to be filled in based on agreement contents and system status. The template section defines the location of the holes as fields and how to fill them in. Fields can be filled in either by agreement elements, identified by xpointers into the agreement document. In addition, capacity decision-makers can be defined that compute a field's value based on the system status and the agreement. Capacity decision makers are system-specific.
- The *AIP Template Set* can be managed using an administrative client to edit AIP templates.
- The *Capacity Decision-Maker* is an interface to a system-specific and agreement-aware *Capacity Planner*.
- The *Provisioning* interface is an abstract interface to an AIP-driven provisioning system, which is invoked using a system-specific *Provisioning Proxy*.
- During the validity of an agreement, a *Compliance Monitor* uses the agreement instance of the APRM – or its proxy on the agreement initiator's side – to check whether guarantees are met. A sophisticated compliance monitor, e.g., eModel, not only analyzes current violations but also predicts future violations and takes corrective actions in advance.
- If guarantees are violated the compliance monitor invokes the *Compliance Manager Interface*, the

implementation of which is system-specific. In many cases, this involves a change in system configuration or schedule.

Figure 7 illustrates the flow of invocations affecting service provider ASRM components upon a createAgreement request:

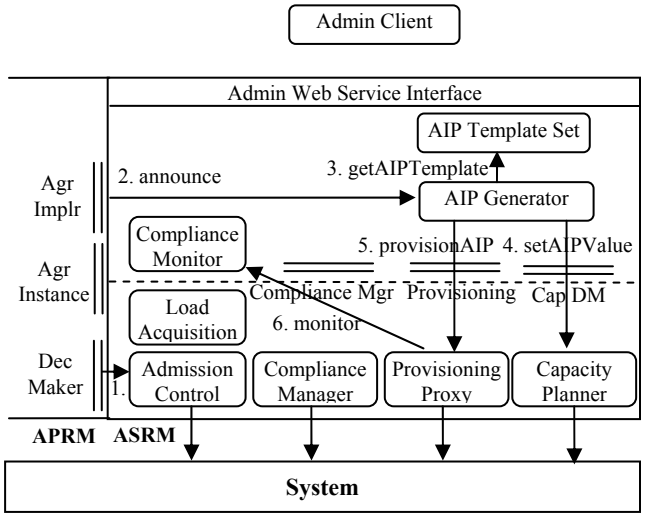

Figure 7: ASRM - service provider structure.

First, the admission control is invoked through the decision maker interface. It verifies feasibility and available capacity of the proposed agreement. If admitted, the AIP generator receives the announcement through the agreement implementer interface; it retrieves the corresponding AIP Template and processes it. AIP fields are filled in either by agreement elements or capacity planners are invoked. When the AIP is completed, it is submitted to the provisioning system through the proxy exposing the domain-independent interface. Finally, the compliance monitor is activated.

The primary purpose of the **Service Consumer ASRM** is to provide an application client an interface to retrieve information how to access a contracted service based on information about the client's requirements.

The service consumer ASRM comprises the following components:

- The *Agreement Request Interface* is invoked by an application client to obtain information how to access a service that it can use. The input to the request is a list of service descriptions or pointers to it and a set of name-value pairs detailing what is meant specifically. In the case of a Web service, this would be a WSDL definition and, for example, the name of a port type, if multiple port types are defined. The response to the request is a list of service information items. In the case of a Web service, this would be a WSDL definition containing binding information and potentially an endpoint reference. Generally, this interface has a very open structure. However, for a particular type of service conventions have to be established between application clients and service consumer APRM on the input parameters and replies.
- The *Service Mapping* maintains which agreement templates can be used to acquire a given service based on service descriptions as expected in the agreement request interface. For each service description entry the mapping contains references to agreement templates of different providers. In the case of Web services, the mapping can be established on the basis of a WSDL file reference and a port type name. For other domains other ways the mapping can be established on other ground. The mappings are set by an administrator.
- The *Client Agreement Selector* is the core of the service consumer ASRM. It implements the agreement request interface, uses the service mapping to find matching agreement templates, looks for agreements based on those templates, and decides which agreement to choose for a particular request. The decision-making for a particular agreement can be based on many aspects such a current guarantee compliance, of an agreement, use of an agreement by other clients and past performance. This component is inherently domain-specific.

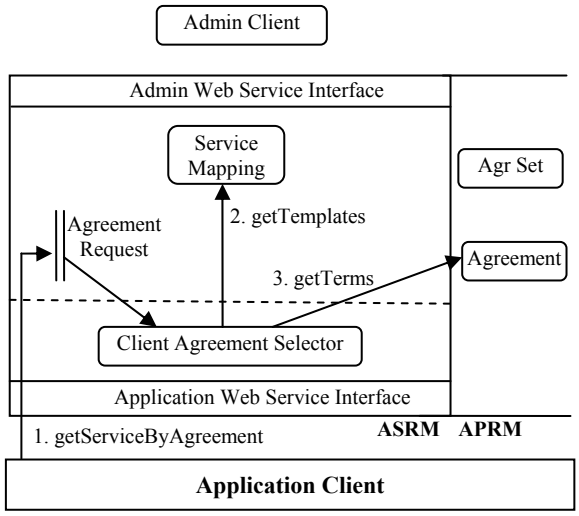

Figure 8: ASRM - service consumer structure.

Figure 8 outlines the basic flow in this component.

5.3 Strategic Agreement Management

Strategic Agreement Management (SAM) is the meta-layer to APRM and ASRM. If automated, it decides which agreement templates and AIP templates to use and may set policies for the behavior of components, if they are policy-enabled. Decisions in the area of SAM always involve understanding the specific domain and a specific system. Based on monitoring agreement compliance and usage, measured through APRM and ASRM components and the service system itself, SAM components can analyze behavior of agreement parties, usage of templates and so forth and can either modify autonomously templates and component policies, which is non-trivial, or the analysis can be interpreted by an administrator and lead to changes. Cremona does not propose a specific component model for the SAM layer of management.

6. CREMONA LIBRARY

The Cremona library supports the implementation of an Agreement Manager as a Java Servlet. It provides implementa-

tions of domain-independent components and interfaces for APRM and ASRM in Java.

Using the APRM classes, a WS-Agreement provider and initiator can be implemented by implementing the status monitor. In addition, the AIP template set and the AIP generator provide a significant part of a AIP-driven provider ASRP. On the service consumer side, the service mapping supports the implementation. Finally, the structure imposed by the set of interfaces solves conceptual problems for an implementer of an ASRP and also administrative and application clients.

7. CONCLUSIONS AND OUTLOOK

7.1 Summary and Discussion

Cremona enables parties to provide and acquire services by agreement as defined in the WS-Agreement standards draft. The Cremona agreement management architecture defines mechanisms to implement WS-Agreement interactions and connects them to the service providing and consuming systems. By separating the service role of a party from its role in the WS-Agreement protocol, both service providers and consumers can initiate agreement creation using Cremona. WS-Agreement addresses a large variety of services, requiring partially domain and system-specific implementations of a subset of agreement management components. The Cremona architecture identifies domain-specific components such as system monitors and provisioning systems and defines their interfaces in a domain-independent way, hence making it applicable to a wide range of service environments. The Cremona library provides implementations of the domain-independent components in Java and defines interfaces that can be implemented in a domain-specific manner. An implementation of the Cremona architecture acts as an agreement middleware for service providers and consumers, facilitating service binding by agreement.

7.2 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.

In many cases, those approaches are specific to a particular aspect of a service. For example, WSLA and WSOL are specifications of QoS agreements for Web services [12],[13],[16]. WLSA has also been used to drive a system provisioning [6],[11]. However, both approaches are restricted to Web services and do not propose solutions for service consumers. Other approaches propose agreements formats and infrastructure to facilitate interaction and coordination between parties, e.g., tpaML/BPF and CrossFlow [5], [9]; also a specific aspect.

In the context of the ODP Enterprise Language a model for the representation of contractual obligations has been proposed but no language has been specified by the ISO [10], along with other work on the formalization of contractual obligations and rights [15], [14].

The Cremona architecture and library enable users to provide or use various kinds of services sold and acquired using the WS-Agreement standard while being able to complement the infrastructure with domain and system-specific components.

7.3 Outlook

WS-Agreement as it stands is suitable for a wide range of services and different aspects of a service such as QoS, interfaces, etc. However, its set of obligation types is not very rich and the set of obligations is static, no new obligations can be added by, e.g., exercising an option. This is likely to change in the future development of WS-Agreement and will be reflected in Cremona. Furthermore, Cremona does not support relationships between agreements, which will be addressed in future versions. More work is needed on the domain-specific adaptation of the application client interface of the agreement management, the relationship of agreement management to advertisement of a service, negotiation support and intermediaries, and the metering of service use on consumer and provider side. Finally, we will work on additional support for layer 2, providing middleware to reduce domain-specific implementation effort.

8. Acknowledgements:

We would like thank Bob Filepp, John Rofrano and Daniela Rosu for their helpful comments and feedback on the paper.

9. REFERENCES

[1] J. Aman, C. K. Eilert, D. Emmes, P. Yocom, and D. Dillenberger, “Adaptive algorithms for managing a distributed data processing workload”, *IBM Systems Journal*, Volume 36, 2, 1997, pp. 242-283.

[2] A. Andrieux, C. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu. *Web Services Agreement Specification (WS-Agreement). Version 1.1, Draft 20,* June 6th 2004.

[3] D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, J. Shewchuk. *Web Services Policy Framework (WS-Policy)*. May 28th, 2003.

[4] J. Cole, J. Derrick, Z. Milosevic, and K. Raymond: Policies in an enterprise specification. In M. Sloman (ed.). *Proceedings of the Policy Workshop*, 2001.

[5] 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.

[6] 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.

[7] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, S. Weerawarana. *Modeling Stateful Resources with Web Services. Version 1.0*. May 1st, 2004.

[8] S. Graham, K. Czajkowski, D. Ferguson, I. Foster, J. Frey, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, S. Tuecke, S. Weerawarana. *Web Services Resource Properties (WS-ResourceProperties). Version 1.0,* May 1st, 2004.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] H. Ludwig, M. Stolze: Simple Obligation and Right Model (SORM) - for the runtime management of electronic service contracts. *IBM Research Report*, RC22765, April, 2003.

[15] 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.

[16] 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.