Three-Phase Model for Negotiation in WS-Agreement
draft February 12, 2004

Alain Andrieux[†] Karl Czajkowski[†] Carl Kesselman[†] Heiko Ludwig[‡]

[†] Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292 U.S.A.

[‡] IBM T.J. Watson Research Center
Hawthorne, NY 10025 U.S.A.

Abstract

This memo proposes a three-phase model for stateful negotiation in WS-Agreement. The Global Grid Forum's GRAAP-WG is designing WS-Agreement to support a wide range of negotiated service management scenarios in the global Web. The proposed three-phase model divides negotiation into: optional, stateful Negotiation, *which can be completed by forming an observed* Agreement, *capturing domain-specific obligations between two parties during an ongoing service-provisioning process; optional, stateful* Renegotiation *allows revision of the terms of service in an existing Agreement without disruption in service. A peer-to-peer, symmetric negotiation model is described to create and manage the Negotiation, Agreement, and Renegotiation states through a sequence of* offer *exchanges rendered as Web service operations. Asymmetric restrictions to the negotiation model are allowed to support more limited deployment scenarios.*

1 Introduction

The GRAAP working group of the Global Grid Forum is drafting a specification for the management of resources and services using negotiated service agreements in a Web services environment (WS-Agreement). The previously proposed WS-Agreementarchitecture [**?**] has roles of *initiator* and *provider* in the negotiation system. Meaning to support a wide range of management scenarios in the Web, we rename the initiator-complementing role as *responder* to avoid confusion with domain-specific notions of "service provider." We also decompose the notion of a stateful Agreement into three distinct stateful entities: a preliminary (and optional) *Negotiation* state machine progresses until a stateful *Agreement* is created to represent ongoing service-delivery obligations; an optional *Renegotiation* state machine replicates the Negotiation model but updates an existing Agreement state instead of creating a new one.

We hope to support the use of the WS-Agreement signaling architecture in a wide range of usage scenarios. For the same kind of end resource or service, eg. computational job hosting, there may be different signaling strategies that apply based on the structure and policies of the service community. If WS-Agreement is to apply to these domains regardless of which strategy is preferred, we must not encumber the negotiation protocol with strategy-specific assumptions. For example, a *push* strategy for job submission might favor highly-available (or centralized) job systems receiving jobs from disconnecting users. On the other hand, a *pull* strategy for job acquisition might favor widely distributed job systems receiving jobs from highly-available (or centralized) users or brokers. In either case, the strategies will require signaling about the same sorts of domain-specific requirements—the job negotiation pertains to the nature and quality of a job execution. The Agreement terms, describing what domain-specific behaviors are desired, should use the same vocabulary regardless of which party initiates the negotiation.

The difference in signaling patterns captures different negotiation infrastructure capabilities but does not necessarily imply anything about the roles of the parties in domain-specific terms. For example, the job-execution domain's "service provider" is always the one executing the job, regardless of whether it is a well-known service point to which jobs are pushed by users or it is a pseudo-anonymous service point which pulls jobs from a broker.

Specifically, in this memo we propose the following architectural changes to WS-Agreement:

1. *Three-phase negotiation model.* The three process states of Negotiation, Agreement, and Renegotiation are rendered as Web service *resources* that can be used to qualify the use of WS-Agreement operations.

2. *Symmetric signaling model.* We show how a simplistic shared-state Agreement concept model maps into a distributed service environment where the actual negotiation state is split or *replicated* between the two negotiating parties in a

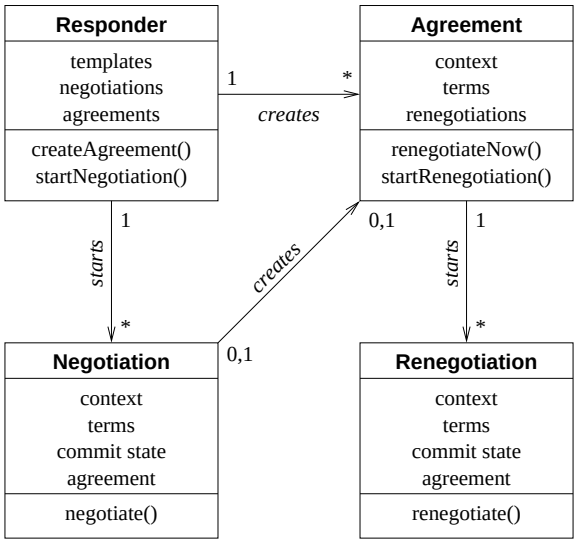

Figure 1. Three-phase resource-qualified endpoint model. The well-known Responder acts as an initial contact point for establishing pairwise Negotiation and Agreement states. The Negotiation resource optionally precedes Agreement, and the Agreement acts as a contact point for establishing Renegotiation states that may eventually update the Agreement state.

peer-to-peer fashion. The parties remain coherent with respect to our simplified state model through the exchange of *offers* that denote the state changes at each step. Furthermore, we show how client-server, asymmetric negotiation is a restriction of this model, where one of the pairwise states is not addressable by the other party through Web service messages.

3. *Whole agreement commitment and re-commitment.* The semantics of the negotiation model is an asynchronous, two-party commitment, where parties become obligated to one another and *observe* an Agreement. We define a protocol based on unilateral *soliciting*, *committing*, and *observing* offers.

4. *Extensible non-obligating negotiation.* The Negotiation and Renegotiation resource state models include *advisory* states wherein the two parties may discover terms that they may successfully observe. Advisory offer exchange precedes the whole-agreement commitment process. Advisory offers do not imply any obligation for the parties, i.e. either party can refuse to commit an Agreement with terms that they exchanged during the advisory negotiation.

WS-Agreement domains requiring multiple stages of obligation MUST define term languages that can be bundled into multiple *dependent* Agreements. Each such Agreement, through the base protocol model, will represent one of the stages of obligation. For example, an advance reservation of a job system would be represented as an Agreement that is referenced in a later job execution Agreement.

2 Namespaces

The following namespace prefixes and namespace values are used in this document:

Prefix	Specification	Namespace
wsag	WS-Agreement	http://ws-agreement *(temporary)*
wsa	WS-Addressing	http://schemas.xmlsoap.org/ws/2003/03/addressing
wsbf	WS-BaseFaults	http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults
wsgg	WS-ServiceGroup	http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup
wsrp	WS-ResourceProperties	http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties
xs/xsd	XML Schema	http://www.w3.org/2001/XMLSchema
wsdl	WSDL	http://schemas.xmlsoap.org/wsdl/

3 Agreement Structure

This section includes a synopsis of term-language work being done by other GRAAP-WG members. We describe it only for context in other discussions in this memo, because we use the agreement document structure to encode offers. The Agreement consists of two main parts:

1. *Context.* The context is static information defined at the beginning of the negotiation process which identifies, among other things, the parties who are negotiating for service.

   Issue 1: Should the context also define statically the *domain model* or term language(s) of the agreement?

2. *Terms.* The terms capture the dynamic or negotiable states of the agreement. There are several categories of term:

   (a) *Service description* terms capture the domain-specific structure of one or more services that are being negotiated. These terms capture the functional nature of the service, eg. what kind of service it is and how that service is instantiated if the domain permits variations or functional parameterization.

   (b) *Guarantee* terms capture the non-functional or quantitative aspects of the negotiated service behavior. These terms describe behavioral characteristics that do not materially affect the kind of service being negotiated, eg. what level of availability or performance the service provides. Guarantee terms may use a generic value or constraint calculus, but their semantics are rooted in the domain-specific concepts of the service description.

   (c) *Monitoring* terms enumerate the measurable characteristics of the negotiated service that will be exposed by either party through some monitoring mechanism. This category of terms is not well understood, nor is it necessarily needed in the base WS-Agreement specification.

The context of the Agreement will be set at the time that a pairwise negotiation is initiated. The terms will evolve according to the stateful negotiation protocol described in this memo.

3.1 Separation of Roles

WS-Agreement defines a protocol to be used between two parties: the *initiator* who starts a stateful Negotiation and the *responder* with whom the negotiation was started. Both parties MAY provide offer-exchange endpoints with the same operations, though only one party MUST provide these endpoints. A special Responder endpoint is used to start the negotiation process. Both parties may be capable of acting as responders, but in any WS-Agreement interaction one of the parties must act as the initiating client toward the other party's Responder endpoint.

The WS-Agreement roles are not intended to imply any restriction to the domain-specific role(s) that a WS-Agreement party may take. The signaling roles for a given application should be chosen based on environmental requirements such as relative visibility, availability, or cardinality of the domain-specific roles. For example, a *push*-based job system would have the initiating user contact a well-known execution resource to establish a new Agreement that the responding resource will execute. Conversely, a *pull*-based job system would have the initiating execution resource contact a user (or broker) to establish a new Agreement that the initiating resource will execute for the responding user. In both cases, the execution resource is obliged to perform a job and the user might be obliged to pay money, supply data, etc.

Thus, part of a domain-specific model must be terms to designate the roles of the parties. These terms should be used to formulate the context of the Agreement, eg. to indicate in an Agreement that the initiating party is the "job submitter" and the responding party is the "job executor" or vice-versa.

Issue 2: Is there a special syntax requirement placed on term languages for specifying domain roles, or will the context syntax for party roles simply reference an arbitrary external concept by QName or some such?

4 Three-Phase State Model

The three-phase model for WS-Agreement negotiation involves non-obligating *Negotiation* and *Renegotiation* processes and one obligated *Agreement* process. A fourth Responder process is assumed to expose policies of the responder, but the Responder is relatively stateless as far as the negotiation process is concerned. An initial Negotiation represents a stateful conversation between a WS-Agreement initiator and responder. A commitment protocol terminates the Negotiation state machine in an *observed* state which simultaneously creates the Agreement in its observed state with the same terms that were committed in the Negotiation. An optional Renegotiation represents a stateful conversation between the parties of an existing Agreement. A commitment protocol terminates the Renegotiation state machine in an observed state which simultaneously updates the existing Agreement to the new observed state. Figure 1 illustrates these stateful resources and their relationships.

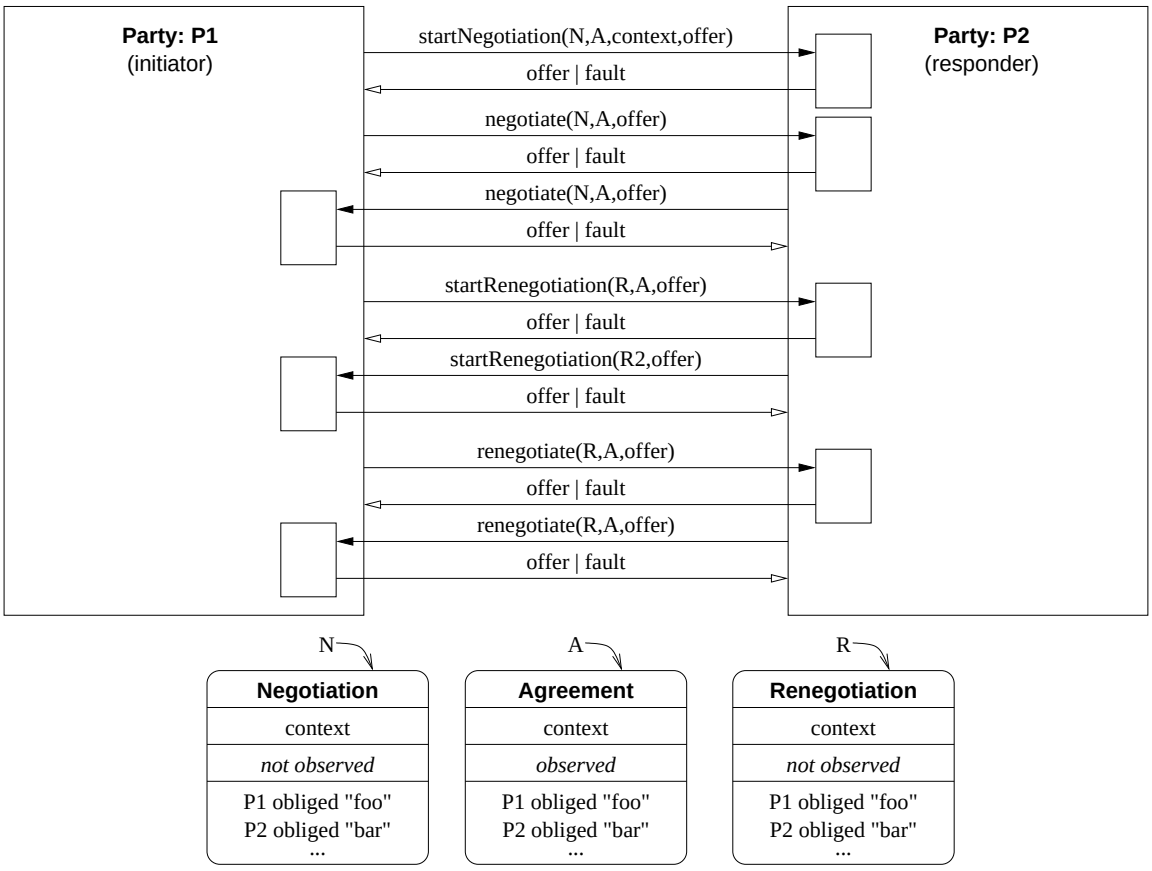

Figure 2. Shared-state negotiation. A simplistic view of negotiation where both parties manipulate a shared state. The parties take turns operating on the state through offers. However, a real Grid negotiation system must work with limited trust, where the parties would be unwilling to allow unilateral access to the shared state.

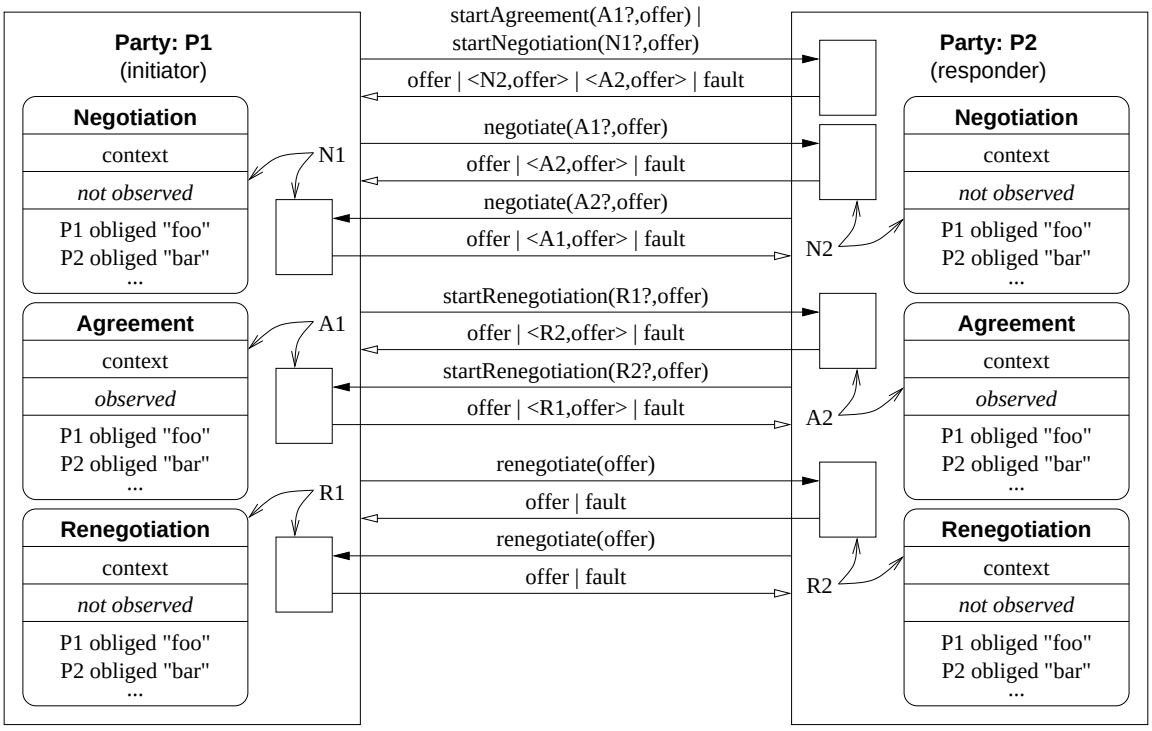

Figure 3. Distributed, peer-to-peer negotiation with a WSRF naming scheme. Localized identifiers for Negotiations, Agreements, and Renegotiations can be encoded as resource-qualified endpoint references to which messages are directed. Each party's view of the states are kept coordinated by the offer exchange model.

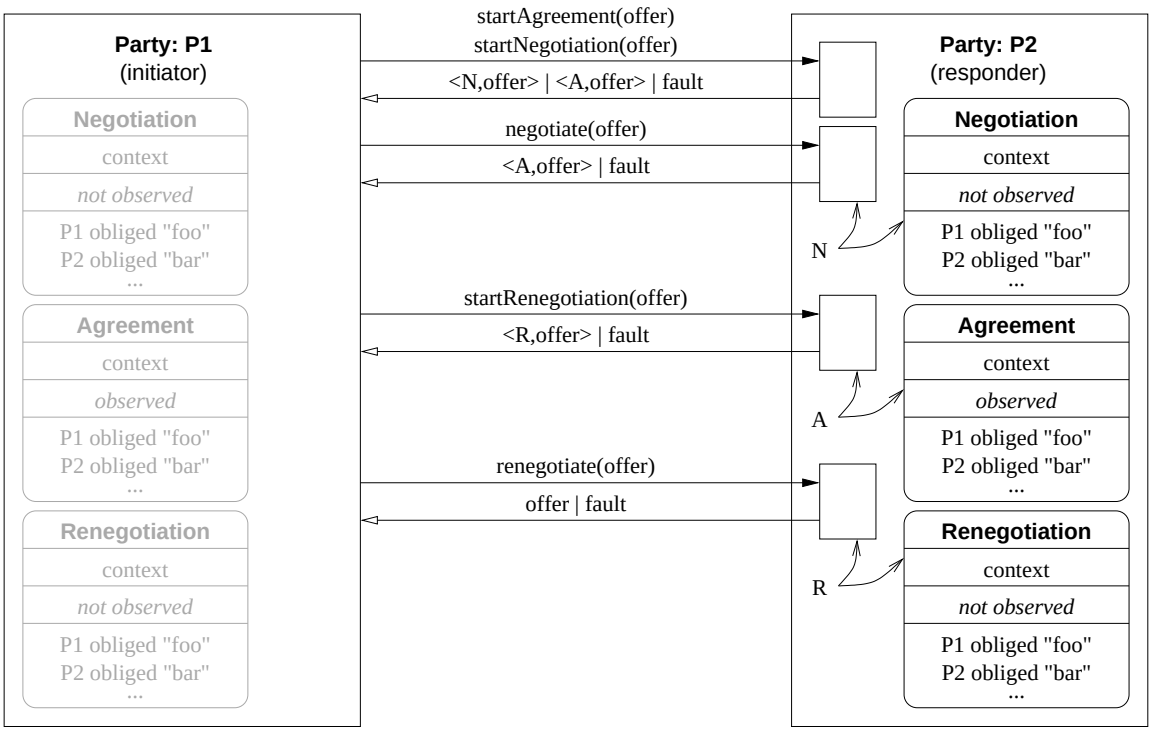

Figure 4. Client-server negotiation. The initiator can omit its endpoints and still refer to the Agreement state by the responder's resource-qualified endpoints. Thus client-server negotiation is a restriction of the general peer-to-peer negotiation model, where the responder is no longer able to directly target initiator state with operation invocations.

5 Simplified Agreement Model

The underlying, abstract model for WS-Agreement is a shared-state negotiation model where the two parties agree to a set of terms that describe their obligations to one another. Figure 2 depicts this abstract view, with messages between the two parties and shared knowledge of Negotiation, Agreement, and Renegotiation states. This approach could be implemented directly using Web service concepts with the identifier for the shared state passed as message payload to the WS-Agreement operations. However, such an implementation does not address the distributed and federated environment of Grids. Also omitted from this discussion is how a global Agreement identifier might be obtained. The Simple Negotiation and Access Protocol (SNAP) [1] describes several approaches to naming negotiation state in a similar abstract protocol model. An alternate approach—managing localized names for the state, as used by each party—is discussed below.

The realities of distributed messaging complicate the abstract model. The two parties each have their own version of the negotiation state known locally, and the negotiation protocol in effect acts as the coherence protocol to keep these states from diverging more than expected for the asynchronous decision process which underlies the commitment model.

Figure 3 depicts this view with localized identifiers following the WS Resource Framework [**?**] which encodes state with associated message endpoints as a "resource qualified" endpoint address, referred to as an *end-point reference* (EPR). The EPR is similar to the *locator* concept from OGSI 1.0. Because resources are named by EPR, Figure 3 omits them from message payload where they would be redundant (where they already qualify the message addressing).

For certain scenarios, the initiating party may not wish, or be able, to implement the full WS-Agreement endpoint capabilities. With some loss in functionality, one party can be client-only to the other's service interface. Figure 4 depicts this limited scenario (as compared to Figure 3) where only the responder implements endpoints named by EPR. Note that the initiator still has an implied view of the Agreement state in his local implementation, but this state is no longer addressable in the architecture. Figure 5 depicts the more general set of asymmetric state combinations that are possible with WS-Agreement; these combinations are achieved by the parties each exercising the optional EPR-exchanging parts of the protocol. We expect that particular domains or deployment communities might choose to follow particular asymmetric conventions in support of their specific goals, but the general WS-Agreement protocol is designed to support all meaningful combinations dynamically.

Figure 5. Full range of possible asymmetric combinations. Asymmetry is not limited to pure client-server but can also surface through *role-reversal*, where the initiating party becomes the main contact point for further messages. Negotiation and Renegotiation asymmetry can be considered separately, with the impact of Negotiation resulting in symmetric or asymmetric Agreement between the phases. Four Renegotiation asymmetry scenarios are omitted because they are equivalent to one of the existing ones, but with the party names reversed.

6 Commitment Protocol

The purpose of the WS-Agreement negotiation and commitment protocol is twofold: it supports exchange of per-party preferences and capabilities information, and it finishes with a *two-party binary decision process* to determine whether the Agreement will be observed. The decision process is asynchronous and uses six types of message to change the state of the Agreement:

1. *Accepting* messages indicate that the sender of the offer has accepted a binding offer by the recipient, and the Agreement is observed.
2. *Rejecting* messages indicate that the sender is dismissing a binding offer made by the recipient. No obligations are held by either party.
3. *Committing* messages indicate that the sender of the offer is committed to the terms described in its payload. The party to send such commitment is *bound*, waiting for the other party to accept or reject the offer. The accepting party decides whether the agreement is observed.
4. *Soliciting* messages indicate that the sender of the offer requires a committing counter-offer. The party who sends a soliciting offer is able to precipitate a conclusion to negotiation without himself being bound to a distributed acceptance decision.
5. *Advisory* messages allow exchange of party preferences without any obligation or constraint on future message exchanges. This is an extensibility point to support more specialized or domain-specific exchange sequences that precede a commitment decision.
6. *Termination* messages indicate that the sender is abandoning the share negotiation state, regardless of what state it is in. Termination may not reverse obligations inherent in an existing observed agreement.

This handshake is designed to minimize ambiguity in the face of failures and delays, while clarifying the inherent risk or uncertainty of negotiation in a widely distributed, Byzantine fault environment such as the Web.

Just as we separate the distinction of initiator and responder roles from domain-specific obligation roles above, we also separate those roles from the committing and accepting roles here. There is no intrinsic need in the signaling protocol for either the initiator or responder to be the one taking the risk of binding commitment.

6.1 Extensibility

There are two ways to extend WS-Agreement negotiation patterns for complex domain requirements. The first way is to decompose service descriptions and guarantees into sets that can be negotiated sequentially as separate Agreements. This multi-agreement approach allows one to faithfully model environments where incremental or piecemeal obligations are assumed by the two parties as they negotiate a complex relationship. We expect to apply this technique to job submission with advance reservation; the advance reservation will be a preliminary Agreement that is utilized in subsequent job submission Agreements. The advance reservation Agreement obligates the job hosting provider to cooperate in a subsequent job submission Agreement that is consistent with the capabilities or conditions described in the advance reservation terms.

The second way to extend the negotiation process is through elaboration of the advisory negotiation process. The basic WS-Agreement state model defines an optional initial advisory state within which specialized protocol state machines can function to discover or exchange domain-specific information before attempting to complete the process via the standard committing message sequence.

6.2 Commitment and Risk

When either party is ready to finalize an Agreement, they decide to send a committing or soliciting offer. The choice depends on which party will be committing, and this choice depends on both the natural asymmetry of *risk* in a given service or resource domain and the policies of the negotiators. By risk, we mean that the committing party is waiting to hear the other party's acceptance decision. If the second party accepts, the agreement is observed. During the period between when the first party sends a committing offer and receives the reciprocal acceptance, he is uncertain whether the agreement will be observed.

If the agreement terms include a schedule for service performance, the observed agreement is in force for the duration of that schedule, *even if the committed party does not learn of the second party's acceptance until after the schedule has begun*

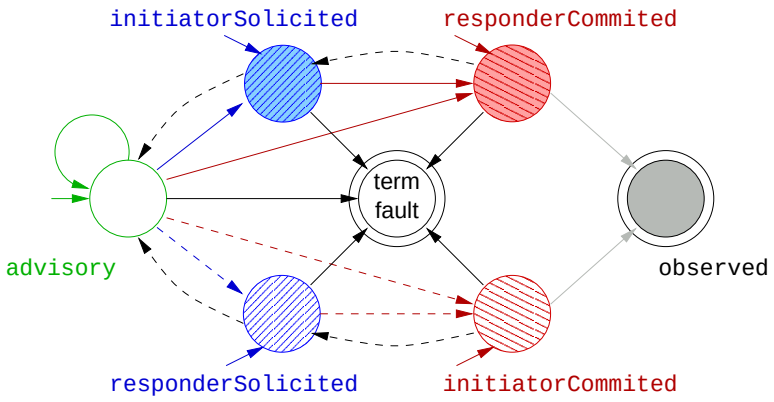

Figure 6. Negotiation States. The *advisory* start state is changed to *solicited* or *committed* by one of the parties sending an appropriate offer. The terminal *observed* state is reached by acceptance from one of the committed states. The terminal fault state is reached by explicit termination or by terminal faults. A synchronous continuing fault invalidates a state change implied by the faulted offer.

or completed. This is an unavoidable result of negotiating about real-time service terms in a messaging environment that does not provide reliable delivery with bounded delay. In effect, the committed party has to gamble during the uncertainty period. If he speculates that the other party will accept, he may commence honoring the agreement terms even before he receives acceptance. If he speculates that the other party will reject the offer, he may not have enough time to honor the terms properly if the party actually accepts. This is particularly troublesome when the activities needed to honor the agreement are not reversible, i.e. they must be performed without full rollback capabilities. There is potentially no "safe" choice where the committed party can wait for further information; in the context of scheduled delivery, waiting is itself a choice not to deliver.

Since there is no way to resolve the conflict between the Byzantine fault model of the Web and quasi-realtime agreement semantics, we can only distinguish the at-risk and safe roles in WS-Agreement and leave it to applications to mitigate costs. As suggested above, a first step is to analyze a particular service domain and put the risk on the party that can safely speculate and roll-back as necessary. Secondly, if one can expect a certain statistical distribution of outcomes, the speculation can be biased to reduce average cost for successful and failed offers.

6.3 Offer Types and Negotiation State

There are six messages that can be provided to, or returned from operations in the WS-Agreement model. Four of them are variants of the generic agreement element. For simplicity, each offer type corresponds directly to a state in the negotiation state-machine, as depicted in

Figure 6. The offer is a suggestion to **enter** the state named in the offer. The offer type is encoded using the /@commitment attribute:

```
<agreement commitment="offer type">
   ...
</agreement>
```

The six message types are represented as follows:

1. *Advisory offers* bear the wsag:advisory value and indicate no obligations or restrictions on further negotiation.
2. *Soliciting offers* indicate no obligations but require that a counter-offer be committed. There are role-specific solicitation offer types:
   (a) *Initiator-solicited* offers are sent by the initiator and bear the wsag:initiatorSolicited value.
   (b) *Responder-solicited* offers are sent by the responder and bear the wsag:responderSolicited value.
3. *Committing offers* indicates that the sender is obligated to the offer terms if the recipient decides to observe. There are role-specific commitment offer types:
   (a) *Initiator-commited* offers are sent by the initiator and bear the wsag:initiatorCommited value.

(b) *Responder-commited* offers are sent by the responder and bear the wsag:responderCommited value.

4. *Accepting offers* bears the wsag:observed value and indicates that the sender accepts the offer that has been committed by the recipient.

5. *Termination* uses the underlying WS-RF termination mechanisms and indicate a destruction of all shared Negotiation, Agreement, or Renegotiation state. Third-party resolution, outside the scope of WS-Agreement, may still be used to resolve obligations from terminated Agreements.

6. *Rejection* uses the underlying WS-RF fault mechanisms to signal rejection of an offer, without losing the shared state that existed prior to the rejected offer.

The protocol state machine and operation message requirements restrict the conditions and means by which these offers may be delivered. The protocol states are named according to offer type. Practically speaking, the state of the sender changes when he decides to send an offer of that type, and the state of the receiver changes when he processes a received offer of that type.

Issue 3: Faults are unavoidable in widely distributed systems; we do not wish WS-Agreement to be fragile in the face of such faults, so we include the rejection mechanism. Does there need to be a way to reject offers through an input message, in addition to the fault-response as an output message?

7 Standard Faults

There are two base faults that can be returned from operations in the WS-Agreement model:

1. *Terminal faults* signal the unwillingness of the invoked party to continue the negotiation. In other words, not only did the operation fail, but the invoked Negotiation or Renegotiation resource should be assumed to be terminated.

2. *Continuing faults* signal the unwillingness of the invoked party to accept the delivered message, but permit continued negotiation. The invoking party may choose to try again, or explicitly terminate the negotiation if desired.

Terminal faults indicate termination of the shared negotiation state. Continuing faults encode offer rejection as described above.

Issue 4: Terminal faults will be proposed as a new feature in WS-BaseFaults part of the WSRF specification set, to augment the usual continuing fault model.

References

[1] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *Lecture Notes in Computer Science*, 2537:153–183, 2002.

Issue 5: The following are intended to become part of the specification document. The preceding material is a mixture of primer material and some material that may belong in introductory parts of the specification.

A General Notes about the Port Types and Operations

1. Each port type expose the three operations specified in the WS-ResourceProperties specification so as to let clients obtain its resource properties. None of the port types exposes the standard operation to set resource properties, as the state associated with each port type is meant to be modified only via WS-Agreement operations such as wsag:startAgreement or wsag:negotiate.Thes operations are not detailed in the textual definition below.
2. wsag:Responder and wsag:Agreement are service groups according to the definition in the WS-ServiceGroup specification and as such, feature the corresponding mandatory resource properties explained briefly below.
3. Each operation can potentially throw an UnknownResource fault if the resource associated with the targeted service (in the WS-ResourceProperties implied resource pattern) does not exist.
4. local elements and attributes from the WS-Agreement schemas are unqualified.

B Port Type: wsag:Responder

Because wsag:Responder is a persistent, reusable service, faults from its operations are normally continuing faults. A terminal fault MAY be returned to indicate extreme failure conditions, which the client SHOULD interpret as a signal to abandon this wsag:Responder.

B.1 Operation: wsag:startAgreement

The wsag:startAgreement operation is used to directly generate an Agreement without any intervening Negotiation.

B.1.1 Input

```
<wsag:startAgreementInput>
   <initiatorAgreementEndpointReference>EPR1</initiatorAgreementEndpointReference>?
   <agreement commitment="wsag:initiatorCommitted">
      ...
   </agreement>
</wsag:createAgreementInput>
```

The optional /initiatorAgreementEndpointReference element (EPR1) provides a contact point where the responder can send messages pertaining to this negotiated Agreement. The offer type /agreement/@commitment MUST be wsag:initiatorCommitted. The responder MUST NOT invoke operations on EPR1 after returning a fault on this operation.

B.1.2 Result

The successful result of wsag:startAgreement is a combination of the optional EPR of a newly created responder Agreement and the acceptance of the initiator's offer:

```
<wsag:startAgreementResponse>
   <createdAgreementEndpointReference>EPR2</createdAgreementEndpointReference>?
   <agreement commitment="wsag:observed">
      ...
   </agreement>
</wsag:startAgreementResponse>
```

The /agreement offer MUST be textually equivalent to the input offer except that the /agreement/@wsag:commitment offer type MUST be wsag:observed. The optional /createdAgreementEndpointReference element (EPR2) MAY appear if the optional EPR1 appeared in the input and MUST appear if EPR1 was omitted from the input. If the EPR2 result appears, it is the EPR to a newly created Agreement bearing the same observed terms.

B.1.3 Faults

A fault response indicates that the offer was rejected and may also indicate domain-specific reasons. A fault response also indicates that EPR1 will be discarded without use (if one was provided).

Issue 6: Should we advocate reuse of EPR1 after a fault, or require that "dead" EPRs not be reused?

B.2 Operation: wsag:startNegotiation

The wsag:startNegotiation operation is used to generate a stateful Negotiation.

B.2.1 Input

```
<wsag:startNegotiationInput>
   <initiatorNegotiationEndpointReference>EPR4</initiatorNegotiationEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:startNegotiationInput>
```

The input MAY contain /initiatorNegotiationEndpointReference (EPR4), providing a contact point where the Responder can send messages pertaining to this stateful negotiation. The offer type MAY be wsag:advisory orwsag:initiatorSolicited.

The responder MUST NOT invoke operations at the EPR4 **until** the stateful negotiation has progressed to an observed agreement.

B.2.2 Result

The successful result of wsag:startNegotiation is a combination of the EPR of a newly created Agreement or wsag:Negotiation and any counter-offer leading to its creating:

```
<wsag:startNegotiationResponse>
   <createdNegotiationEndpointReference>EPR5</createdNegotiationEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:startNegotiationResponse>
```

The response MAY have /createdNegotiationEndpointReference (EPR5). The offer MAY be nil if EPR5 appears, but MUST NOT be nil otherwise. If EPR4 did not appear as input, EPR5 MUST appear.

B.2.3 Faults

A fault response indicates that no wsag:Negotiation was created and may also indicate domain-specific reasons.

B.3 Resource Properties

B.3.1 Resource Property: templates

The templates resource property represents 0 or more templates of offers that can be accepted by the wsag:Responder operations in order to create an Agreement at once or start a Negotiation. A template defines a grouping of certain agreement terms along with negotiability constraints.

Note: the definition of the template XML Schema type remains to be defined.

B.3.2 Resource Property: wsag:entry

The wsag:Responder port type can create new resource-qualified endpoint references to services (with associated resources) of port types wsag:Agreement (creation of a new agreement at once) or wsag:Negotiation (beginning of a new negotiation in order to eventually reach an agreement). The wsag:Factory port type is modeled as a service group with respect to the WS-ServiceGroup specification. As a service group, when the wsag:Responder acts as a factory it records information about each newly created service-resource pair as a member of the service group in a new wsag:entry resource property instance. The entry typically includes the EPR of the new qualified service plus some optional extra information (see the WS-ServiceGroup specification for more information).

In the context of the wsag:Responder port type, two one-to-zero-or-more relationships are modeled via the wsag:entry resource property, which is of cardinality zero or more:

1. some of the entries refer to wsag:Agreement services.
2. the other entries refer to wsag:Negotiation services.

B.3.3 Resource Property: wsag:membershipContentRules

The wsag:membershipContentRules resource property contains a set of wssg:MembershipContentRule elements that specify the intensional constraints on each member service of the service group (see resource property wsag:entry). Each wsgg:membershipContentRule specifies at least a port type that every member service in the service group must implement.

In the context of the wsag:Responder, there must be one wsgg:membershipContentRule specifying wsag:Agreement as the member port type, and one wsgg:membershipContentRule specifying wsag:Negotiation as the member port type.

```
<wsag:membershipContentRules>
    <wsgg:MembershipContentRule MemberInterface="wsag:Agreement ContentElements="qnames"/>
    ...
    <wsgg:MembershipContentRule MemberInterface="wsag:Negotiation" ContentElements="qnames"/>
    ...
</wsag:membershipContentRules>
```

A member of the wsag:Responder service group must implement at least one of the two port types specified in wsag:membershipContentRules. See the WS-ServiceGroupspecification for more information one the wsgg:MembershipContentRuleType.

C Port Type: wsag:Negotiation

C.1 Operation: wsag:negotiate

The wsag:negotiate operation is used to continue a wsag:Negotiation started with the wsag:startNegotiation operation.

C.1.1 Input

```
<wsag:negotiateInput>
   <newAgreementEndpointReference>EPR7</newAgreementEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:negotiateInput>
```

The offer type /agreement/@commitment MUST be one of wsag:advisory, wsag:initiatorSolicited, wsag:responderSolicited, wsag:initiatorCommitted, wsag:responderCommited, or wsag:observed governed by the protocol state machine depicted in Figure 6. If the offer type is wsag:observed, the optional /createdAgreementEndpointReference MAY appear to present a contact-point for the invoked party to communicate with the invoking party regarding this observed Agreement.

C.1.2 Result

The successful result of wsag:negotiate is a combination of the (optional) EPR of a newly created Agreement and any counter-offer:

```
<wsag:negotiateResponse>
   <createdAgreementEndpointReference>EPR8</createdAgreementEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:negotiateResponse>
```

If the input EPR7 appeared, the optional /createdAgreementEndpointReference (EPR8) MAY appear. If the input EPR7 did not appear but EPR8 does, the agreement MUST NOT be nil. If EPR8 does not appear, the agreement child element MAY be nil. A non-nil result agreement MUST bear an offer type, governed by the protocol state machine depicted in Figure 6.

C.1.3 Faults

A continuing fault indicates that the wsag:Negotiation was unable to accept the input offer and the state remains unchanged from before the invocation. A terminal fault indicates that the wsag:Negotiation was unable to accept the offer and the wsag:Negotiation will terminate immediately.

> Issue 7: Should all the offer methods have timeout inputs to control the synchrony of responses?

C.2 Resource Properties

C.2.1 Resource Property: wsag:context

The wsag:context resource property is of type wsag:AgreementContextType (which is currently not defined in this document). The context is static information about the negotiation such as the parties involved in the negotiation. See the section in this document about the agreement context for more information.

C.2.2 Resource Property: wsag:terms

The wsag:terms resource property is of type wsag:AgreementType (which is currently not defined in this document). This property specifies the terms being currently negotiated.

> Issue 8: Do we want to create a specific TermSetType that would represent set of terms for negotiation as opposed to a more complete AgreementType representing a commited agreement that would include the context as well?

C.2.3 Resource Property: wsag:commitStatus

Note: This property is currently not defined by itself as it is part of the AgreementType. We might want to model that differently.

> Issue 9: Should the AgreementType (and/or the TermSetType if any) include a field for the commit state or should the commit state also (or instead?) be a distinct resource property? Should we care about redundancy?

C.2.4 Resource Property: wsag:agreementServiceEPR

The wsag:agreement resource property is an EPR to an agreement service created by the wsag:Negotiation after a successful negotiation process.

```
<wsag:agreementServiceEPR>
    EPR
</wsag:agreementServiceEPR>
```

D Port Type: wsag:Agreement

D.1 Operation: wsag:renegotiateNow

The wsag:renegotiateNow operation is used to update the terms of an existing Agreement without disrupting the service described in the wsag:Agreement.

D.1.1 Input

The only input to wsag:renegotiateNow is a committing offer:

```
<wsag:renegotiateSimpleInput>
   <agreement commitment="commit type">
      ...
   </agreement>
</wsag:renegotiateSimpleInput>
```

The agreement MUST NOT be nil and the commit type /agreement/@commitment MUST be wsag:initiatorCommitted or wsag:responderCommited, as governed by the protocol state machine.

D.1.2 Result

The successful result of wsag:renegotiateNow is the Observing offer that updates the state of the wsag:Agreement:

```
<wsag:renegotiateResponse>
   <agreement commitment="wsag:Observed">
      ...
   </agreement>
</wsag:renegotiateResponse>
```

D.1.3 Faults

A fault response indicates that the Agreement state was not changed and may also indicate domain-specific reasons.

D.2 Operation: wsag:startRenegotiation

The wsag:startRenegotiation operation is used to initiate a stateful wsag:Renegotiation process.

D.2.1 Input

```
<wsag:startRenegotiationInput>
   <newRenegotiationEndpointReference>EPR9</newRenegotiationEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:startRenegotiationInput>
```

The offer type /agreement/@commitment MUST be one of wsag:advisory, wsag:initiatorSolicited, or wsag:responderSolicited, as governed by the protocol state machine. The optional /initiatorNegotiationEndpointReference (EPR9) element provides a contact point where the invoked party can send messages pertaining to this stateful renegotiation. The invoked party MUST NOT invoke operations at EPR9 if generating a fault response to this invocation.

D.2.2 Result

The successful result of wsag:startNegotiation is an optional EPR of a newly created wsag:Renegotiation and any counter-offer leading to its creation:

```
<wsag:startRenegotiationResponse>
   <createdRenegotiationEndpointReference>EPR9</createdRenegotiationEndpointReference>?
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:startRenegotiationResponse>
```

The response MAY have either an /createdRenegotiationEndpointReference (EPR9). If a non-nil agreement appears it MUST have a type as governed by the protocol state machine.

D.2.3 Faults

A continuing fault response indicates that no wsag:Renegotiation was created and that the Agreement remains in the state it was in before the invocation, and may also indicate domain-specific reasons. A terminating fault SHOULD be interpreted as a severe failure and the invoking party should not expect the Agreement to respond to further messages.

D.3 Resource Properties

D.3.1 Resource Property: wsag:context

The wsag:context resource property is of type wsag:AgreementContextType (which is currently not defined in this document). The context is static information about the agreement such as the parties involved in the agreement. See the section in this document about the agreement context.

D.3.2 Resource Property: wsag:terms

The wsag:terms resource property is of type wsag:AgreementType (which is currently not defined in this document)

> Issue 10: Do we want to create a specific TermSetType to distinguish from a complete AgreementType that would include the context as well?

This property specifies the terms of the agreement.

D.3.3 Resource Property: wsag:entry

The wsag:Agreement port type can create new resource-qualified endpoint references to services (with associated resources) of port type wsag:Renegotiation when executing the operation wsag:startRenegotiation in order to start renegotiation of the existing agreement. The wsag:Agreement port type is modeled as a service group with respect to the WS-ServiceGroup specification. As a service group, when the wsag:Agreement acts as a factory and creates a new wsag:Renegotiation, it records information about the newly created wsag:Renegotiation service-resource pair as a member of the service group in a new wsag:entry resource property instance. The entry typically includes the EPR of the new qualified service plus some optional extra information (see the WS-ServiceGroup specification for more information).

In the context of the wsag:Agreement port type, one one-to-zero-or-more relationship is modeled via the wsag:entry resource property, which is of cardinality zero or more:

1. all service group entries refer to wsag:Renegotiation services.

D.3.4 Resource Property: wsag:membershipContentRules

The wsag:membershipContentRules resource property contains a set of wssg:MembershipContentRule elements that specify the intensional constraints on each member service of the service group (see resource property wsag:entry). Each wsgg:membershipContentRule specifies at least a port type that every member service in the service group must implement.

In the context of the wsag:Agreement, there must be at least one wsgg:membershipContentRule specifying wsag:Renegotiation as a port type implemented by the member services.

```
<wsag:membershipContentRules>
    ...
    <wsgg:MembershipContentRule MemberInterface="wsag:Renegotiation" ContentElements="qnames"/>
    ...
</wsag:membershipContentRules>
```

See the WS-ServiceGroup specification for more information one the wsgg:MembershipContentRuleType.

E Port Type: wsag:Renegotiation

E.1 Operation: wsag:renegotiate

The wsag:renegotiate operation is used to continue a wsag:Renegotiation started with the wsag:startRenegotiation operation.

E.1.1 Input

```
<wsag:renegotiateInput>
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:renegotiateInput>
```

The offer type /agreement/@commitment MUST be one of wsag:advisory, wsag:initiatorSolicited, wsag:responderSolicited, wsag:initiatorCommitted, or wsag:responderCommited, governed by the protocol state machine depicted in Figure 6.

E.1.2 Result

The successful result of wsag:renegotiate is a any counter-offer:

```
<wsag:renegotiateResponse>
   <agreement commitment="offer type">
      ...
   </agreement>
</wsag:renegotiateResponse>
```

The the agreement child element may be nilled, if the wsag:Renegotiation wishes to return successfully without issuing a counter-offer. In that case, the wsag:Renegotiation is in the state defined by the input offer. A non-nil result agreement MUST bear an offer type governed by the protocol state machine.

E.1.3 Faults

A continuing fault indicates that the wsag:Renegotiation was unable to accept the input offer and the state remains unchanged from before the invocation. A terminal fault indicates that the wsag:Renegotiation was unable to accept the offer and the wsag:Renegotiation will terminate immediately.

E.2 Resource Properties

E.2.1 Resource Property: wsag:context

The wsag:context resource property is of type wsag:AgreementContextType (which is currently not defined in this document). The context is static information about the agreement (and the renegotiation) such as the parties involved in the negotiation. See the section in this document about the agreement context for more information.

E.2.2 Resource Property: wsag:terms

The wsag:terms resource property is of type wsag:AgreementType (which is currently not defined in this document)

This property specifies the agreement terms being currently renegotiated.

E.2.3 Resource Property: wsag:commitStatus

Note: This property is currently not defined by itself as it is part of the AgreementType. We might want to model that differently.

E.2.4 Resource Property: wsag:agreementServiceEPR

The wsag:agreement resource property is an EPR to an agreement service created by the wsag:Negotiation after a successful negotiation process.

```
<wsag:agreementServiceEPR>
    EPR
</wsag:agreementServiceEPR>
```

F Basic Types

Issue 11: Every effort has been made to maintain the following XML Schema and WSDL in synch with the preceding operation discussions. However, if conflicts are found, please consider the operation discussions as "authoritative" for the purpose of debate and the following will be repaired to match.

F.1 Commitment Status

```
<xsd:simpleType name="CommitmentType">
    <xsd:restriction base="xsd:QName">
        <xsd:enumeration value="wsag:Uncommitted"/>
        <xsd:enumeration value="wsag:InitiatorSolicited"/>
        <xsd:enumeration value="wsag:ResponderSolicited"/>
        <xsd:enumeration value="wsag:InitiatorCommitted"/>
        <xsd:enumeration value="wsag:ResponderCommitted"/>
        <xsd:enumeration value="wsag:Observed"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:attribute name="commitment" type="wsag:CommitmentType"/>
```

F.2 Faults

Note: the wsbf prefix is mapped to the WS-BaseFaults URI. The normative Schema definition of the base type BasicFaultType from that specification could not be read at the time of this writing.

```
<xsd:complexType name="TerminalFaultType">
    <xsd:complexContent>
        <xsd:extension base="wsbf:BasicFaultType"
        xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ContinuingFaultType">
    <xsd:complexContent>
        <xsd:extension base="wsbf:BasicFaultType"
        xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
```

```
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="terminalFault" type="wsag:TerminalFaultType"/>
<xsd:element name="continuingFault" type="wsag:ContinuingFaultType"/>
```

There currently is no boolean field in wsbf:BasicFaultType to indicate if a fault is terminal or continuing. Furthermore, according to the WS-BaseFaults specification, a distinct type of fault must be expressed as a separate type derived by extension from wsbf:BasicFaultType, and a separate element must be defined. Here we therefore define two distinct (type, element) pairs, one for continuing faults and one for terminal faults.

F.3 AgreementContext

F.4 Agreement

The Schema type wsag:AgreementType is defined elsewhere.

G WSDL

Notes:

1. No comments explaining the operation preconditions and postconditions and the operation parameter and return type invariants were embedded in the current WSDL. We will wait for this proposal to be part of the specification in order to do so. In this way there we avoid the issue of maintaining the text documentation and the WSDL-embedded commments in synch.

2. In the WSDL below, the XML Schema encoding of the operational input and output types tries to strike balance between simplicity and enforcement of invariants expressed in the textual definition. Simplicity is important since none or very few XML Schema parsing tools support all the features and validity constraints of the XML Schema specification. For instance, complex forms of derivation by restriction have been avoided even though they would have permitted to express certain type invariants. The WSDL in this proposal might change in order to enable some tooling to support the digestion of the types. In any case it is important that WS-Agreement implementations do not rely on the typing and XML validition implied by the normative WSDL but try to implement the verification - when appropriate - of the rules expressed textually.

3. Notification and subscription for notification when resource properties are modified need to be specified, and expressed in the port types using WS-Notification mechanisms.

4. Asynchronicity of certain operations need to be defined and explained. Currently WSDL1.1 lacks standard ways of specifying asynchronous operations so the very use of that specification may restrict the expression of operation asynchronicity.

> Issue 12: Should some or all of the resource property types for the respective port types have an extensibility element in the form of an XML Schema wildcard as it is allowed by the WS-ResourceProperties specification?

G.1 wsag:ResponderPortType

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:wsag="http://ws-agreement"
        xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
        targetNamespace="http://ws-agreement">
```

```
    <import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    location="http://www.ibm.com/developerworks/library/ws-resource/WS-ResourceProperties.wsdl"/>
    <import namespace="http://ws-agreement" location="agreement_types.xsd"/>
    <types>
        <xs:schema targetNamespace="http://ws-agreement" xmlns:wsag="http://ws-agreement"
        xmlns:wssg="http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup">
            <xs:import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup"
            schemaLocation="ws-servicegroup.xsd"/>
            <!-- no published WS-ServiceGroup schema yet? -->
            <!--Resource property element declarations-->
            <xs:element name="template" type="wsag:AgreementType"/>
            <!-- TBD: specific template type? -->
            <!--======================-->
            <!--WS-ServiceGroup properties-->
            <xs:element name="membershipContentRules" type="wssg:MembershipContentRulesType">
                <xs:annotation>
                    Contains at least one membershipContentRule1 element such that
                    membershipContentRule1/@memberInterface="wsag:Negotiation"
                    and one membershipContentRule2 element such that
                    membershipContentRule2/@memberInterface="wsag:Negotiation"
                </xs:annotation>
                <!--or can we define restriction of MembershipContentRulesType ?-->
            </xs:element>
            <xs:element name="entry" type="wssg:EntryType">
                <xs:annotation>This element represents a negotiation service or an agreement
                    service created by the responder.</xs:annotation>
            </xs:element>
            <!--Resource property document declaration-->
            <xs:element name="responderProperties" type="wsag:ResponderPropertiesType"/>
            <xs:complexType name="ResponderPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:template" minOccurs="0" maxOccurs="unbounded"/>
                    <!-- cardinality rules and access to the item: TBD -->
                    <xs:element ref="wsag:membershipContentRules"/>
                    <xs:element ref="wsag:entry" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema targetNamespace="http://ws-agreement" xmlns:wsag="http://ws-agreement"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <xs:element name="startAgreementInput" type="wsag:StartAgreementInputType"/>
            <xs:element name="startAgreementResponse" type="wsag:StartAgreementOutputType"/>
            <xs:element name="startNegotiationInput" type="wsag:StartNegotiationInputType"/>
            <xs:element name="startNegotiationResponse" type="wsag:StartNegotiationOutputType"/>
            <xs:complexType name="StartAgreementInputType">
```

```
                <xs:sequence>
                    <xs:element name="initiatorAgreementEndpointReference"
                    type="wsa:EndpointReferenceType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="StartAgreementOutputType">
                <xs:sequence>
                    <xs:element name="createdAgreementEndpointReference"
                    type="wsa:EndpointReferenceType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="StartNegotiationInputType">
                <xs:sequence>
                    <xs:element name="initiatorAgreementEndpointReference"
                    type="wsa:EndpointReferenceType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="StartNegotiationOutputType">
                <xs:choice>
                    <xs:sequence>
                        <xs:element name="createdAgreementEndpointReference"
                        type="wsa:EndpointReferenceType"/>
                        <xs:element name="agreement" type="wsag:AgreementType"/>
                    </xs:sequence>
                    <xs:sequence>
                        <xs:element name="createdNegotiationEndpointReference"
                        type="wsa:EndpointReferenceType"/>
                        <xs:element name="agreement" type="wsag:AgreementType"
                        nillable="true"/>
                    </xs:sequence>
                </xs:choice>
            </xs:complexType>
        </xs:schema>
    </types>
    <message name="startAgreementInputMessage">
        <part name="parameters" element="wsag:startAgreementInput"/>
    </message>
    <message name="startAgreementOuputMessage">
        <part name="parameters" element="wsag:startAgreementResponse"/>
    </message>
    <message name="startAgreementFaultMessage">
        <part name="fault" element="wsag:continuingFault"/>
    </message>
    <message name="startNegotiationInputMessage">
        <part name="parameters" element="wsag:startNegotiationInput"/>
```

```
    </message>
    <message name="startNegotiationOuputMessage">
        <part name="parameters" element="wsag:startNegotiationResponse"/>
    </message>
    <message name="startNegotiationFaultMessage">
        <part name="fault" element="wsag:terminalFault"/>
    </message>
    <portType name="Responder" wsrp:ResourceProperties="wsag:responderProperties">
        <operation name="startAgreement">
            <input message="wsag:startAgreementInputMessage"/>
            <output message="wsag:startAgreementOuputMessage"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="terminalFault" message="wsag:startAgreementFaultMessage"/>
            <!-- or message="wsbf:baseFaultMessage " name="baseFault"
            with terminal = true-->
        </operation>
        <operation name="startNegotiation">
            <input message="wsag:startNegotiationInputMessage"/>
            <output message="wsag:startNegotiationOuputMessage"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="terminalFault" message="wsag:startNegotiationFaultMessage"/>
            <!-- or message="wsbf:baseFaultMessage " name="baseFault"
            with terminal = true-->
        </operation>
        <!-- pasting resource property accessor definitions from WSRP -->
        <operation name="GetResourceProperty">
            <input name="GetResourcePropertyRequest" message="wsrp:GetResourcePropertyRequest"/>
            <output name="GetResourcePropertyResponse" message="wsrp:GetResourcePropertyResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="GetMultipleResourceProperties">
            <input name="GetMultipleResourcePropertiesRequest"
            message="wsrp:GetMultipleResourcePropertiesRequest"/>
            <output name="GetMultipleResourcePropertiesResponse"
            message="wsrp:GetMultipleResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="QueryResourceProperties">
            <input name="QueryResourcePropertiesRequest"
            message="wsrp:QueryResourcePropertiesRequest"/>
            <output name="QueryResourcePropertiesResponse"
            message="wsrp:QueryResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
            <fault name="UnknownQueryExpressionLanguage" message="wsrp:ErrorMessage"/>
```

```
            <fault name="InvalidQueryExpression" message="wsrp:ErrorMessage"/>
            <fault name="QueryEvaluationError" message="wsrp:ErrorMessage"/>
        </operation>
    </portType>
</definitions>
```

G.2 wsag:NegotiationPortType

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsag="http://ws-agreement"
    xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    targetNamespace="http://ws-agreement">
    <import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    location="http://www.ibm.com/developerworks/library/ws-resource/WS-ResourceProperties.wsdl"/>
    <import namespace="http://ws-agreement" location="agreement_types.xsd"/>
    <types>
        <xs:schema targetNamespace="http://ws-agreement"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
        xmlns:wsag="http://ws-agreement">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <!--Resource property element declarations-->
            <xs:element name="context" type="wsag:AgreementContextType"/>
            <xs:element name="terms" type="wsag:AgreementType"/>
            <xs:element name="agreementServiceEPR" type="wsa:EndpointReferenceType"/>
            <!-- TBD: specific agreement terms type? -->
            <!--Resource property document declaration-->
            <xs:element name="negotiationProperties"
            type="wsag:NegotiationPropertiesType"/>
            <xs:complexType name="NegotiationPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:context"/>
                    <xs:element ref="wsag:terms"/>
                    <!--TBD: if commit state part of terms structure or not-->
                    <xs:element ref="wsag:agreementServiceEPR" minOccurs="0"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema targetNamespace="http://ws-agreement"
            xmlns:wsag="http://ws-agreement"
            xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
            xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <xs:element name="negotiateInput" type="wsag:NegotiateInputType"/>
            <xs:element name="negotiateResponse" type="wsag:NegotiateOutputType"/>
            <xs:complexType name="NegotiateInputType">
```

```
                <xs:sequence>
                    <xs:element name="newAgreementEndpointReference"
                    type="wsa:EndpointReferencetType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="NegotiateOutputType">
                <xs:sequence>
                    <xs:element name="createdAgreementEndpointReference"
                    type="wsa:EndpointReferenceType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"
                    nillable="true"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </types>
    <message name="negotiateInputMessage">
        <part name="parameters" element="wsag:negotiateInput"/>
    </message>
    <message name="negotiateOuputMessage">
        <part name="parameters" element="wsag:negotiateResponse"/>
    </message>
    <message name="negotiateTerminalFaultMessage">
        <part name="fault" element="wsag:terminalFault"/>
    </message>
    <message name="negotiateContinuingFaultMessage">
        <part name="fault" element="wsag:continuingFault"/>
    </message>
    <portType name="Negotiation" wsrp:ResourceProperties="wsag:negotiationProperties">
        <operation name="negotiate">
            <input message="wsag:negotiateInputMessage"/>
            <output message="wsag:negotiateOuputMessage"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="terminalFault" message="wsag:negotiateTerminalFaultMessage"/>
            <fault name="continuingFault" message="wsag:negotiateContinuingFaultMessage"/>
            <!-- or message="wsbf:baseFaultMessage " name="baseFault" with terminal = true-->
        </operation>
        <!-- pasting resource property accessor definitions from WSRP -->
        <operation name="GetResourceProperty">
            <input name="GetResourcePropertyRequest"
            message="wsrp:GetResourcePropertyRequest"/>
            <output name="GetResourcePropertyResponse"
            message="wsrp:GetResourcePropertyResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="GetMultipleResourceProperties">
```

```
            <input name="GetMultipleResourcePropertiesRequest"
            message="wsrp:GetMultipleResourcePropertiesRequest"/>
            <output name="GetMultipleResourcePropertiesResponse"
            message="wsrp:GetMultipleResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="QueryResourceProperties">
            <input name="QueryResourcePropertiesRequest"
            message="wsrp:QueryResourcePropertiesRequest"/>
            <output name="QueryResourcePropertiesResponse"
            message="wsrp:QueryResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
            <fault name="UnknownQueryExpressionLanguage" message="wsrp:ErrorMessage"/>
            <fault name="InvalidQueryExpression" message="wsrp:ErrorMessage"/>
            <fault name="QueryEvaluationError" message="wsrp:ErrorMessage"/>
        </operation>
    </portType>
</definitions>
```

G.3 wsag:AgreementPortType

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsag="http://ws-agreement"
    xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    xmlns:ns="http://schemas.xmlsoap.org/ws/2003/03/addressing"
    xmlns:ns1="http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup"
    targetNamespace="http://ws-agreement">
    <import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    location="http://www.ibm.com/developerworks/library/ws-resource/WS-ResourceProperties.wsdl"/>
    <import namespace="http://ws-agreement" location="agreement_types.xsd"/>
    <types>
        <xs:schema targetNamespace="http://ws-agreement"
            xmlns:wsag="http://ws-agreement"
            xmlns:wssg="http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup">
            <xs:import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ServiceGroup"
            schemaLocation="ws-servicegroup.xsd"/>
            <!-- no published WS-ServiceGroup schema yet? -->
            <!--Resource property element declarations-->
            <xs:element name="context" type="wsag:AgreementContextType"/>
            <xs:element name="terms" type="wsag:AgreementType"/>
            <!-- TBD: specific agreement terms type? -->
            <!--======================-->
            <!--WS-ServiceGroup properties-->
```

```
            <xs:element name="membershipContentRules"
            type="wssg:MembershipContentRulesType">
                <xs:annotation>Contains at least one element membershipContentRule
                such that membershipContentRule/@memberInterface="wsag:Renegotiation"
                </xs:annotation>
                <!--or can we define restriction of MembershipContentRuleType?-->
            </xs:element>
            <xs:element name="entry" type="wssg:EntryType">
                <xs:annotation>This element represents a renegotiation service
                created by the agreement.</xs:annotation>
            </xs:element>
            <!--Resource property document declaration-->
            <xs:element name="agreementProperties" type="wsag:AgreementPropertiesType"/>
            <xs:complexType name="AgreementPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:context"/>
                    <xs:element ref="wsag:terms"/>
                    <xs:element ref="wsag:membershipContentRules"/>
                    <xs:element ref="wsag:entry" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema targetNamespace="http://ws-agreement"
            xmlns:wsag="http://ws-agreement"
            xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
            xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <xs:element name="renegotiateNowInput" type="wsag:renegotiateNowInputType"/>
            <xs:element name="renegotiateNowResponse" type="wsag:renegotiateNowOutputType"/>
            <xs:element name="startRenegotiationInput"
            type="wsag:startRenegotiationInputType"/>
            <xs:element name="startRenegotiationResponse"
            type="wsag:startRenegotiationOutputType"/>
            <xs:complexType name="renegotiateNowInputType">
                <xs:sequence>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="renegotiateNowOutputType">
                <xs:sequence>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="startRenegotiationInputType">
                <xs:sequence>
                    <xs:element name="newRenegotiationEndpointReference"
                    type="wsa:EndpointReferenceType" minOccurs="0"/>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
```

```
            </xs:sequence>
        </xs:complexType>
        <xs:complexType name="startRenegotiationOutputType">
            <xs:sequence>
                <xs:element name="createdRenegotiationEndpointReference"
                type="wsa:EndpointReferenceType" minOccurs="0"/>
                <xs:element name="agreement" type="wsag:AgreementType"
                nillable="true"/>
            </xs:sequence>
        </xs:complexType>
    </xs:schema>
</types>
<message name="renegotiateNowInputMessage">
    <part name="parameters" element="wsag:renegotiateNowInput"/>
</message>
<message name="renegotiateNowOuputMessage">
    <part name="parameters" element="wsag:renegotiateNowResponse"/>
</message>
<message name="renegotiateNowFaultMessage">
    <part name="fault" element="wsag:terminalFault"/>
</message>
<message name="startRenegotiationInputMessage">
    <part name="parameters" element="wsag:startRenegotiationInput"/>
</message>
<message name="startRenegotiationOuputMessage">
    <part name="parameters" element="wsag:startRenegotiationResponse"/>
</message>
<message name="startRenegotiationFaultMessage">
    <part name="fault" element="wsag:terminalFault"/>
</message>
<portType name="Agreement" wsrp:ResourceProperties="wsag:agreementProperties">
    <operation name="renegotiateNow">
        <input message="wsag:renegotiateNowInputMessage"/>
        <output message="wsag:renegotiateNowOuputMessage"/>
        <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
        <fault name="terminalFault" message="wsag:renegotiateNowFaultMessage"/>
        <!-- or message="wsbf:baseFaultMessage " name="baseFault"
        with terminal = true-->
    </operation>
    <operation name="startRenegotiation">
        <input message="wsag:startRenegotiationInputMessage"/>
        <output message="wsag:startRenegotiationOuputMessage"/>
        <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
        <fault name="terminalFault"
        message="wsag:startRenegotiationFaultMessage"/>
        <!-- or message="wsbf:baseFaultMessage " name="baseFault"
        with terminal = true-->
```

```
        </operation>
        <!-- pasting resource property accessor definitions from WSRP -->
        <operation name="GetResourceProperty">
            <input name="GetResourcePropertyRequest"
            message="wsrp:GetResourcePropertyRequest"/>
            <output name="GetResourcePropertyResponse"
            message="wsrp:GetResourcePropertyResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName"
            message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="GetMultipleResourceProperties">
            <input name="GetMultipleResourcePropertiesRequest"
            message="wsrp:GetMultipleResourcePropertiesRequest"/>
            <output name="GetMultipleResourcePropertiesResponse"
            message="wsrp:GetMultipleResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="QueryResourceProperties">
            <input name="QueryResourcePropertiesRequest"
            message="wsrp:QueryResourcePropertiesRequest"/>
            <output name="QueryResourcePropertiesResponse"
            message="wsrp:QueryResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
            <fault name="UnknownQueryExpressionLanguage" message="wsrp:ErrorMessage"/>
            <fault name="InvalidQueryExpression" message="wsrp:ErrorMessage"/>
            <fault name="QueryEvaluationError" message="wsrp:ErrorMessage"/>
        </operation>
    </portType>
</definitions>
```

G.4 wsag:RenegotiationPortType

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsag="http://ws-agreement"
    xmlns:wsrp="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    targetNamespace="http://ws-agreement">
    <import namespace="http://www.ibm.com/xmlns/stdwip/web-services/WS-ResourceProperties"
    location="http://www.ibm.com/developerworks/library/ws-resource/WS-ResourceProperties.wsdl"/>
    <import namespace="http://ws-agreement" location="agreement_types.xsd"/>
    <types>
        <xs:schema
            targetNamespace="http://ws-agreement"
```

```
            xmlns:wsag="http://ws-agreement"
            xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <!--Resource property element declarations-->
            <xs:element name="context" type="wsag:AgreementContextType"/>
            <xs:element name="terms" type="wsag:AgreementType"/>
            <!-- TBD: specific agreement terms type? -->
            <xs:element name="agreementServiceEPR" type="wsa:EndpointReferenceType"/>
            <!--Resource property document declaration-->
            <xs:element name="renegotiationProperties"
            type="wsag:RenegotiationPropertiesType"/>
            <xs:complexType name="RenegotiationPropertiesType">
                <xs:sequence>
                    <xs:element ref="wsag:context"/>
                    <xs:element ref="wsag:terms"/>
                    <!--TBD: if commit state part of terms structure or not-->
                    <xs:element ref="wsag:agreementServiceEPR"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
        <xs:schema
            targetNamespace="http://ws-agreement"
            xmlns:wsag="http://ws-agreement"
            xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
            xmlns:wsbf="http://www.ibm.com/xmlns/stdwip/web-services/WS-BaseFaults">
            <xs:import namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"/>
            <xs:element name="renegotiateInput" type="wsag:renegotiateInputType"/>
            <xs:element name="renegotiateResponse" type="wsag:renegotiateOutputType"/>
            <xs:complexType name="renegotiateInputType">
                <xs:sequence>
                    <xs:element name="agreement" type="wsag:AgreementType"/>
                </xs:sequence>
            </xs:complexType>
            <xs:complexType name="renegotiateOutputType">
                <xs:sequence>
                    <xs:element name="agreement" type="wsag:AgreementType"
                    nillable="true"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
    </types>
    <message name="renegotiateInputMessage">
        <part name="parameters" element="wsag:renegotiateInput"/>
    </message>
    <message name="renegotiateOuputMessage">
        <part name="parameters" element="wsag:renegotiateResponse"/>
    </message>
    <message name="renegotiateTerminalFaultMessage">
```

```
        <part name="fault" element="wsag:terminalFault"/>
    </message>
    <message name="renegotiateContinuingFaultMessage">
        <part name="fault" element="wsag:continuingFault"/>
    </message>
    <portType name="Renegotiation">
        <operation name="renegotiate">
            <input message="wsag:renegotiateInputMessage"/>
            <output message="wsag:renegotiateOuputMessage"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="terminalFault"
            message="wsag:renegotiateTerminalFaultMessage"/>
            <fault name="continuingFault"
            message="wsag:renegotiateContinuingFaultMessage"/>
            <!-- or message="wsbf:baseFaultMessage " name="baseFault"
            with terminal = true-->
        </operation>
        <!-- pasting resource property accessor definitions from WSRP -->
        <operation name="GetResourceProperty">
            <input name="GetResourcePropertyRequest"
            message="wsrp:GetResourcePropertyRequest"/>
            <output name="GetResourcePropertyResponse"
            message="wsrp:GetResourcePropertyResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName"
            message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="GetMultipleResourceProperties">
            <input name="GetMultipleResourcePropertiesRequest"
            message="wsrp:GetMultipleResourcePropertiesRequest"/>
            <output name="GetMultipleResourcePropertiesResponse"
            message="wsrp:GetMultipleResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName"
            message="wsrp:ErrorMessage"/>
        </operation>
        <operation name="QueryResourceProperties">
            <input name="QueryResourcePropertiesRequest"
            message="wsrp:QueryResourcePropertiesRequest"/>
            <output name="QueryResourcePropertiesResponse"
            message="wsrp:QueryResourcePropertiesResponse"/>
            <fault name="UnknownResource" message="wsrp:ErrorMessage"/>
            <fault name="InvalidResourcePropertyQName" message="wsrp:ErrorMessage"/>
            <fault name="UnknownQueryExpressionLanguage" message="wsrp:ErrorMessage"/>
            <fault name="InvalidQueryExpression" message="wsrp:ErrorMessage"/>
            <fault name="QueryEvaluationError" message="wsrp:ErrorMessage"/>
        </operation>
```

```
    </portType>
</definitions>
```