GFD.199
NO-WG

Version: 0.1

Andre Merzky
Alexander Papaspyrou

April 1st, 2012

---

# WS-Disagreement

<u>Status of This Document</u>

This document provides information to the grid community, proposing a standard for a WS-Disagreement protocol (WS-NO), to communicate disagreement messages between components, and also between humans.

Distribution of this document is unlimited.

<u>Copyright Notice</u>

# Contents

# 1 Introduction

A significant number of WS-Negotiation [9] exchanges are ultimately ending in disagreement. Those unsucessful negotiations are, however, exactly those which consume most resources.

We thus introduce this WS-Disagreement (WS-NO) protocol as an optimization of such doomed nogotiations. The WS-Agreement [2] protocol (WS-YES) is complementary to WS-NO. When exactly the WS-NO protocol is preferable to WS-YES depends on the specific use case and infrastructure, and is not discussed in this document – but in general, WS-NO is peferred whenever performance of negotiation is of any importance.

## 1.1 Notational Conventions

The group could not reach agreement about notational conventions. This document is thus free-form. Implementors MAY thus interpret the document at will – that will increase the performance of protocol implementations significantly.

## 1.2 Security Considerations

The protocol specified in this document does not provide any specific security measures – message verification and channel authorization/authentication is left to the lower level wire protocol. Implementations MUST ensure that communicated disagreements are truthfully preserved, to avoid spontaneous (virtual) agreement.

## 2 WS-Disagreement

The WS-Disagreement (short: WS_NO) protocol provides a mechanism to communicate disagreement between applications, and by proxy between individuals. The intent of the protocol is to formalize a commonly re-occuring negoatiation pattern, and to simplify the respective technical communication. The applicability of the protocol is, however, not limited to the exchange of disagreement notification between process instances – it is just as well applicable to exchange of disagreements between (a)social human beings. In particular, section 3.3 will discuss how the protocol can be applied to typical OGF working group discussions, increasing their effectivity by orders of magnitude.

Note that the protocol is extremely easy to implement: any two implementations disagreeing on the channel setup are considered compliant implementations – more complex implementations though, which do in fact establish a communication channel, MUST adhere to the specified wire protocol in order to be WS-NO compliant. WS-NoCompliance [7] is specifically not considered a replacement for WS-Disagreement.

### 2.1 Endpoint State Model

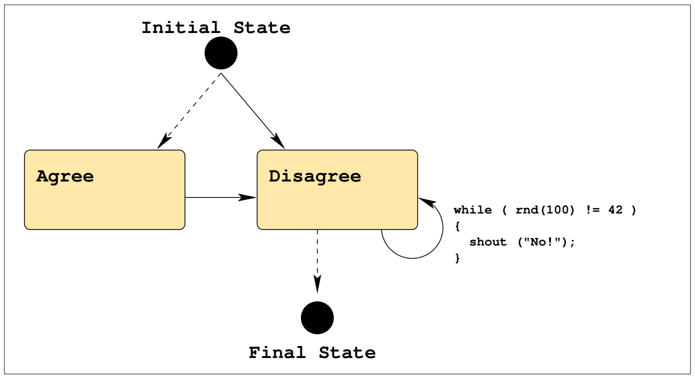

Figure 1: **The WS-Disagreement `endpoint` state model**

The state model for WS-Disagreement endpoint instances is very simple: An endpoint MUST immediately disagree on receiving any protocol message, thus entering the `Disagree` state. The `Agree` state is only reachable by accident, but an automatically invoked transition to `Disagree` ensures the stability of the state model.

It has been observed that real negotiation endpoints are often reaching the

`Disagree` state without realizing that this state is in fact a final state. The state diagram models that behavior: a random number of WS-NO messages is required before the state is accepted as being final.

## 2.2 WS-NO Messages

The presented protocol has merely one message type. Note that there is no need to specify any response message – disagreements have in general no need of response. The protocol enables any endpoint to exchange any number of disagreement messages with its peer endpoints – waiting for the peer endpoint to interpret and respond is not required, and in fact discouraged, to avoid accidental synchronization of the randomizer states (see below).

Although the group could not reach agreement on a single message rendering, this document presents the XML rendering as a guideline to implementors – the respective XSD document is listed in Appendix A.

The message format is:

```
  <wsda:Disagreement DisagreementId="xs:string">

    <wsda:Topic> xs:random_string </wsda:Topic> *
    <wsda:Note>  NO               </wsda:Note>  *

  </wsda:Agreement>
```

On receiving the above message on an established message channel, the endpoint MUST immediately move into the `Disagree` state. For extremely time sensitive implementations, the endpoints CAN also enter the `Disagree` state if they *suspect* they will receive a respective message in the near future. To maintain the consistency of the system in such a case, the respective endpoint MUST immediately send a WS-NO message to all of its own peer endpoints.

For multi-party disagreement communications, messages MAY specify an arbitrary number of `wsda::Topic` elements – that reflects that the participating parties have usually a different notion of the negotiation topic in the first place. In fact, implementations MAY use random topic elements, to increase the probability of disagreement. Message exchanges of coincidentally identical topic strings will move the participating endpoints in the ephemeral `Agree` state – implementations MUST then immediately exchange another message pair with new random strings, to leave the state as soon as possible.

# 3 WS-Disagreement Protocols

The current trend for HTTP-based APIs in modern web services clearly shows the need for a RESTful rendering of the WS-NO protocol. Due to the careful design of the WS-NO messages and state model, the protocol binding to HTTP-based systems is very simple.

In line with the long-standing tradition of building recommendations on the basis of Web Services technology, a SOAP-based rendering of the WS-Disagreement protocol is also provided. It must be noted, however, that, by the mere nature of that protocol being "WS-based", it cannot be expected that any two implementors will interpret the specification in the same way.

Finally, this document will also present a Human-to-Human protocol rendering, with the specific aim to facilitate OGF working group discussions.

## 3.1 REST rendering: Simple HTTP / REST

The HTTP/1.1 protocol [3] is fully sufficient to realise the WS-NO protocol. TO establish disagreement, the requesting party (the client) sends a WS-NO message to the replying party (the server).

The client MUST `POST` the content rendering of its argument to the server's endpoint base URL. If it uses the afore-described XML-based rendering 2.2, the Content-type (see RFC2616 [3], section 16.17] MUST be `'application/xml'`. The client MAY send other content types (if indicating them properly), but the server MAY decline their processing with a `415 Unsupported Media Type` status code. Clients MAY use RFC5988[6]-compliant 'Link' headers to refer to a previous negotiation (argument); however, the server is free to ignore this. Commercial implementations MAY additionally indicate their interest in faster convergence to the result via the *payment* relation; servers then SHOULD try to reach the Disagree state as quick as possible[1]

If the request is received, the server MUST reply with a verbatim mangled of the initial argument in the message body and either of the following status codes. Note that several of this status codes also apply to the Human-to-Human rendering of WS-NO, as annotated.

[1]Lessons from history, however, indicate that Quality of Service guarantees cannot be given, and the expected process speedup MAY turn out to be negative.

### 3.1.1 Status indicating ongoing disagreements

- `100 Continue`, if the server is still undecided whether it will disagree in the future. Alternatively, it MAY reply with `101 Switching Protocols`, if it realises that both party speak different languages.

- `202 Accepted`, if the server will disagree in the future, but has not realised yet that this is going to happen. This status code is expected to be the most prominent one, and corresponds to the "Agree" state. In case of a `202` status, the server MUST create a resource containing the state of the argument that contains a rendering of the current disagreement context. The server SHOULD, in addition, return an RFC5988[6]-compliant 'Link' header with the reply and indicate the number of so far fruitless discussions on the topic via the *index* relation. For further details, it MAY use additional relation indicators such as *alternate* (for first-year contributors only), *glossary* (limited to the OGSA namespace), *license* (limited to the OCCI working group), or *enclosure* (for meta-arguments, Reference Model Working Group), to provide context for future disagreements. It is NOT RECOMMENDED to use the *payment* relation on the server side; especially publicly-funded servers MUST not do so due to potential corruption charges.

- `203 Non-Authoritative Information`, if the server is tentatively agreeing (but eventually will disagree) and claims to do so because of information provided by a third party. It is RECOMMENDED that this information comes from a third party endorsing the same opinion as the server; this way, the server can delegate requests for compensation after disagreement to the endorsing third party. In any case, `203` statuses MUST be used whenever the server disagrees, but does not wish to disclose this information at the current point in time.

- `205 Reset Content`, if the server feels like restarting the discussion from scratch, for whatever reason (which includes no reason at all). With this status given, the client MUST assume that all arguments exchanged so far need to be exchanged again[2].

### 3.1.2 Status indicating redirections and reiterations

- `300 Multiple Choices`, if the client provides an argument that may lead to different decision paths on the server side. Note however that any path will eventually lead to a disagreement.

- `303 See Other` and `304 Not Modified`, if the client provides an argument that has been made already. This way, the server MAY indicate repeating

[2] A common real-world example for this case is the replacement of an implementation by another, e.g. when a project has ended and personnel regarding a specific discussion will change.

discussions. This feature is, however, not expected to be widely implemented.

• `305 Use Proxy`, if ongoing disagreements between this specific client and server implementation have led to a deadlock state where no party wishes to exchange arguments any more. The server MAY indicate with this status code that it is not willing to further discuss directly, but due to multiple implementations (see also HTTP `203` and `206`) another server will do so on its behalf. Management functions such as VPs and ADs MUST implement this status and be prepared to act as such."

### 3.1.3 Status indicating problems in reaching disagreement

• `400 Bad Request`, if the argument made by the client is syntactically not understandable by the server. A common case for this status is line noise, for example if clients are situated in airplanes, trains or other means of movement. Servers MAY decide to interpret garbled messages anyway though – `400` is fully OPTIONAL.

• `405 Method Not Allowed`, if the client uses means of argumentation that are not appropriate to this particular server. Natural examples for such are physical violence or explicit language, but depending on the client and server implementation details, each party's mileage may vary. Note that the implementation of this status is OPTIONAL, however if not implemented, `410 Gone` SHOULD be implemented.

• `406 Not Acceptable`, whenever the client provides an argument in the body that the server is unlikely to be able to compromise on, effectively leading to disagreement. This status MUST be implemented as a fallback at least for (a) arguments on syntactical issues regarding the overall topic to be disagreed upon (a lesson from the OGSA-BES and HPC-BP working groups), and (b) for backup purposes in the unlikely event that disagreement is in peril (a lesson from the PGI working group). With this status code, the client SHOULD refrain from arguing any further, but MAY proceed as indicated for `205 Reset Content`.

• `410 Gone`, if the disagreeing party leaves discussion permanently. The implementation of this status is OPTIONAL in the good tradition of implementations disappearing without notice. Any continuing disagreement must start with `205 Reset Content` (see above).

• `411 Length Required`. For certain (especially long-running) communications, a server MAY wish to limit the length of arguments for the sake of brevity in eventually reaching a disagreement. The corresponding *Content-length* header MUST be returned.

• `413 Request Entity Too Large` and `414 Request-URI Too Long`, if the previous status is reached repeatedly from the same client. The server MAY resort to `305 Use Proxy` if a client repeatedly submits requests triggering these status codes.

- `416 Request Range Not Satisfiable`, if the range of argument cannot be handled by the server in a reasonable way (or time). Specific implementations MAY be more relaxed in using this status than others (see JSDL-WG communication as an example for a strict implementation).

- `503 Service Not Available`, if the server is not able to process arguments supplied by the client for a limited amount of time, because of maintenance or overcommitment. In this case, the server SHOULD indicate this fact by the given status code. Note that for certain geographical regions (e.g. Southern Europe), there are known time periods during which this status may appear; in that case, the server MAY refrain from answering at all.

## 3.2 WS-Rendering: SOAP over HTTP

### 3.2.1 Namespaces

The following namespaces are used in this document:

```
soap  - http://schemas.xmlsoap.org/wsdl/soap/
wsdl  - http://schemas.xmlsoap.org/wsdl/
xs    - http://www.w3.org/2001/XMLSchema
wsno  - http://schemas.ogf.org/no/2012/04/01/
```

### 3.2.2 Disagreement Port Type

The Disagreement Port Type defines a single operation for disagreement. The intent of the port type is to provide an interface for disagreeing on arbitrary topics. Operations are specified using a combination of English and IDL. (A normative rendering is presented in Appendix B.)

**Operations:**

The following section gives (non-normatively) the total set of operations defined on the Disagreement port type. Normative information is provided in the appendix. The request to disagree MUST always succeed. However, as the container MAY take some time to transition between the possible states, ephemeral `Agree` states are possible.

**Disagree:** This operation is used to disagree on a given topic.

| | |
|---|---|
| **Input:** | An argument, statement, or opinion. |
| **Output:** | A disagreement with the provided input. |
| **Faults:** | None. Since the response of the operation is a failure in all cases, the authors do not deem it necessary to additionally complicate the interface. |

### 3.2.3 Optional Extensions:

To determine supported extensions, the container MAY provide additional information; the client SHOULD expected those information to be not agreed upon, and that the actual extension implementation MAY in fact be missing.

**Subscription to Notification Events:**

A service endpoint that allows its clients to subscribe for messages concerning activity state changes MUST do so using either the WS-Eventing [5] or WS-Notification [8] protocols. Since discussions that adhere to WS-NO have additional round-trip messages and race-conditions anyway, compliant WS-NO services implementing this extension SHOULD notify subscribers repeatedly and MAY do so at will.

**Lifetime Management:**

A WS-NO service that supports lifetime of arguments MUST implement WS-ResourceLifetime [4] operation, which SHOULD use the appropriate WS-RL mechanisms to indicate the requestors suggestion for the initial setting of the termination time for this specific argument. Consumers must be aware though that the service MAY ignore the message completely. Also, services MAY accept the termination request and still restart the argument at a random time in the future again.

If the WS-NO implementation is unable or unwilling to set the `TerminationTime` attribute of the argument to the given value, then the request MAY or MAY NOT fail. The use of the xsi:nil attribute with value `'true'` indicates there is no scheduled termination time requested for the argument; this is the default for all arguments and MAY be applied arbitrarily by the service for arguments with different lifetimes. If the element does not include the time zone designation, the value of the element MUST be interpreted with respect to a random timezone.

Alternatively, services MAY use a symmetric interface on the client side to come to a disagreement over acceptable discussion times.

## 3.3 Human-to-Human Rendering

WS-NO can easily be rendered as a human-to-human communication protocol. The authors consider that in particular relevant in order to facilitate OGF standardization discussions. Those are often hallmarks of disagreements, but the exchange modus for disagreement messages is usually extremely time consuming.

The human-to-human implementation of WS-NO is exceptionally simple and elegant: as the mere mentioning of WS-* protocols is a well established method to incite disagreement, the act of uttering *"WS-Disagreement"* is considered to be a fully compliant implementation of the protocol. For the very rare cases that OGF groups are not in disagreement about `WS-*`, we recommend a to mention any additional security layer. For the remaining fringe cases, a language bindings BrainFuck [1] will provide the ultimate solution.

In the unlikely use case that a full WS-NO message exchange is required to reach the `Disagree` state, we RECOMMEND that the message bodies (topics / arguments) are uttered very loudly (shouted in fact), and omnidirectional, in order to increase the signal-to-noise ratio. The general semantics of the return status codes described in 3.1 apply, as annotated there. Those return status code numbers MUST again be shouted, and to keep the disagreement process rolling.

# 4 Intellectual Property Issues

## 4.1 Contributors

This document is the result of the joint efforts of many contributors, and reflects experiences gathered in numerous (and more importantly virtually endless) OGF group discussions. The authors listed here and on the title page are those taking responsibility for the content of the document, and all errors. The editors (underlined) are committed to taking permanent stewardship for this document and can be contacted in the future for inquiries, advise, and training in non-violent communication.

**Andre Merzky**
andre@merzky.net
Center for Computation and Technology
Louisiana State University
216 Johnston Hall
70803 Baton Rouge
Louisiana, USA

**Alexander Papaspyrou**
alexander.papaspyrou@tu-dortmund.de
Institut für Roboterforschung
Technische Universität Dortmund
Otto-Hahn-Str. 8
44227 Dortmund
Germany

We explicitly avoid to list contributors, as those will (a) likely disagree with that document, and (b) are too numerous to list, really. On the other hand, the PGI-WG in fact deserves honorable mentioning. Kudos!

This document is dedicated to Wolfgang Ziegler, who relentlessly heads the WS-Agreement efforts in OGF. His work is admirable, even if ultimately doomed.

## 4.2 Intellectual Property Statement

## 4.3 Disclaimer

## 4.4 Full Copyright Notice

## A WS-NO types

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.ogf.org/no/2012/04/01/"
  xmlns:tns="http://schemas.ogf.org/no/2012/04/01/"
  elementFormDefault="qualified">
  <xs:element name="Disagreement">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Topic" type="xs:string" />
        <xs:element name="Note" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="NO"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="noId" type="xs:ID"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# B WS-NO WSDL

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions
  targetNamespace="http://schemas.ogf.org/no/2012/04/01/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsno="http://schemas.ogf.org/no/2012/04/01/"
  name="WS-Disagreement">
  <wsdl:documentation>
    A service that disagrees with every request.
  </wsdl:documentation>
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
        namespace="http://schemas.ogf.org/no/2012/04/01/"
        schemaLocation="wsno_types.xsd">
      </xsd:import>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="DisagreementRequest">
    <wsdl:part element="wsno:Disagreement" name="disagreeIn" />
  </wsdl:message>
  <wsdl:message name="DisagreementResponse">
    <wsdl:part element="wsno:Disagreement" name="disagreeOut" />
  </wsdl:message>
  <wsdl:portType name="DisagreementPortType">
    <wsdl:documentation>
      The only port type of the WS-Disagreement protocol, and
      probably one of the most used interfaces in standardization.
    </wsdl:documentation>
    <wsdl:operation name="Disagree">
      <wsdl:documentation>
        The standard operation for any kind of disagreement,
        yielding a verbatim copy of the argument provided, and a
        reply expressing the disagreement on it.
      </wsdl:documentation>
      <wsdl:input message="wsno:DisagreementRequest" />
      <wsdl:output message="wsno:DisagreementResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="DisagreementSoapBinding"
    type="wsno:DisagreementPortType">
    <wsdl:documentation>
      A much disagreed-upon SOAP binding of the WS-Disagreement
      protocol
    </wsdl:documentation>
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Disagree">
      <soap:operation
        soapAction="http://schemas.ogf.org/ws-no/2012/04/01/Disagree"
            />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="DisagreementService">
    <wsdl:port binding="wsno:DisagreementSoapBinding"
      name="DisagreementBinding">
      <wsdl:documentation>
        The highly disputable binding of the service to SOAP.
      </wsdl:documentation>
      <soap:address location="http://www.example.org/no/" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

## References

[1] BrainFuck.
`http://en.wikipedia.org/wiki/Brainfuck`.

[2] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification – WS-Agreement. Grid Forum Document GFD.192, 2011. Global Grid Forum.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[4] J. Frey, S. Graham, K. Czajkowski, D. F. Ferguson, I. Foster, F. Leymann, T. Maguire, N. Nagaratnam, M. Nally, T. Storey, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and S. Weerawarana. Web Services Resource Lifetime. `http://www.ibm.com/developerworks/library/ws-resource/ws-resourcelifetime.pdf`.

[5] A. Malhotra, K. Warr, D. Davis, and W. Chou. Web services eventing (WS-eventing). W3C working draft, W3C, Dec. 2009. http://www.w3.org/TR/2009/WD-ws-eventing-20091217.

[6] M. Nottingham. Web Linking. RFC 5988 (Proposed Standard), Oct. 2010.

[7] Till Ulenspiegel and Humpty Dumpty. The WS-NoCompliance Protocol. Grid Forum Document GFD.666, January 1st 1970. Global Grid Forum.

[8] S. Vinoski. Web services notifications. IEEE Internet Computing, 8(2):86–90, March 2004.

[9] O. Waeldrich, D. Battr, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler. WS-Agreement Negotiation Version 1.0 – WS-Negotiation. Grid Forum Document GFD.193, 2011. Global Grid Forum.