An Informal Analysis of SOAP mustUnderstand, message paths, message patterns, etc.

Noah Mendelsohn
Lotus Development
Original: May 11, 2001
(This revision: 5/22/01)

The original version of this note was circulated on 5/11/01, and it has generated a lot of discussion. Unfortunately, there were some glaring errors relating to syntax in the original. I have not done a total rewrite, but this version attempts to correct those errors. I have, for the most part, not updated this to include useful insights from the discussions we have had. I do suggest that we consider doing at least the following:

- *Using this note and the related email trail as guidance, generate a list of potential clarifications (as opposed to design changes) to the SOAP V 1.1 specification.*
- *Rewrite and update this note to reflect whatever agreements we reach regarding the explanations given here about how SOAP works. I think it would be a good idea to do the same in other areas where more clarity is needed.*
- *There is a proposal for a "dependsOn" facility at the end of this note. We need to decide whether it is on the right track, and how it relates to other proposals such as those intended to provide for routing among explicit actors (my proposal provides for ordering among header entries, independent of actor).*
- *Consider a new and related question, raised originally by Glen Daniels: "if you use mustUnderstand to introduce a new feature (dependsOn is an example of such a feature),and if several header entries are target at the same actor, can you be sure the mustUnderstand entry will be noticed early enough to ensure that the new feature is safely acted upon. In other words, how do you avoid a situation where you have already done unsafe processing by the time you notice that a mustUnderstand header to your actor was not understood (note that SOAP does* not *order processing of header entries to a given actor.) I intend to send a note to dist-App starting discussion on this, unless someone else has already done so.*

Noah Mendelsohn
5/22/01

On this week's phone call, I was asked to set down some of the issues relating to routing, bodies and headers, etc. that I believe may be related to a good resolution of the mustUnderstand question. None of this is intended as a comprehensive proposal. I do not think the details are well considered, and I am not sure that everything I say is a good idea. I do hope this will help clarify the discussion or move it ahead.

I have broken this analysis into three major pieces:

- A summary of some of the outstanding questions, and the potential goals of any features in the message path/mustUnderstand area
- A clarification of the way SOAP works today... if I have this right, I believe that at the very least the specification should be clarified accordingly. This section is quite long, and some readers may wish to skip it.
- Some speculations on what might be possible/desireable/undesireable regarding more comprehensive notions of message path and related improvements to the base SOAP function.

Goals and Questions

Note that, in subsequent sections, I attempt to provide answers to these questions both for SOAP as it exists today, and for a possibly enhanced version. Among the things I think we should get out of the discussion are the following:

Clarification of Header/Body implications

SOAP implies that Body is just syntactic sugar for a header entry with no actor, but there is some implication that Body is the preferred way of carrying whichever request represents the primary function of the message. Some clarification may be useful.

Clarify message path

SOAP does refer to a message path, which appears to be post facto the ordered locations and processing (there is vagueness here) that actually were visited in processing the message. There is at least an attempt to associate special behavior with the "end" of the path, where header entries with no actor are processed. The details of all this need to be stated more precisely.

Clarify relationship of fault processing to message path, message pattern and transport bindings

SOAP very clearly indicates situations in which a fault must be generated, but only sometimes is it clear whether and how and to whom to deliver the fault. Some of that looseness is intentional, but more clarity is needed in the explanation. I take a crack at this below. Furthermore, as we begin to apply header entries to richer functions which may themselves fail, we need to be sure that fault handling is suitable and carefully documented.

Consider and clarify facilities for ensuring that essential processing is done, and in the right order

Clearly, successful handling of the message depends in many cases not only on having particular header entries understood by associated software, but on ensuring that all essential header entries are indeed picked up by some such software (not skipped), and in a suitable order. A core question that we face is how much of the infrastructure to ensure this should be the business of the core SOAP specification, and how much can be in a separate specification.

Insofar as necessary for the above, consider the use of modules and new mustUnderstand headers

The proposal has been made that the existing mustUnderstand facility in SOAP is indeed a building block which allows us to layer support for ordered delivery, and for checking the order of processing, as a separate module on top of the core XML protocol. On the other hand, mustUnderstand applies to new header entries (I.e. element children of the <Header>, but not to enhancements modeled as attributes on existing header entries or on the <Header> element itself. The question of whether we can do the new facilities as a clean module therefore boils down, in part, to the question of whether the new facilities can be effectively and conveniently modeled as new header entries in the messages exchanged by applications.

How SOAP works today

I know the following is not exactly what the specification says: that's the point. I am trying to read between the lines and make more precise what the specification implies. If I don't have it right, please help me out. Also, as noted above, this section is very long as I have tried to capture all the details I could. Skip down to the "A Proposal" section if you don't feel like reading this clarification of current SOAP. I do think this is useful background in understanding the proposal I

make later.

Clarification of Header/Body implications

A SOAP message consists of an optional <Header>, containing a lexically ordered set of header entries (element children), followed by a <Body> which is itself a distinguished form of header entry. Unless otherwise stated, all the characteristics of header entries apply to bodies as well.

Although the SOAP specification says [1]: ": A body entry is semantically equivalent to a header entry intended for the default actor and with a SOAP mustUnderstand attribute with a value of "1"." I believe this is somewhat misleading and should be corrected. I believe that a more accurate explanation would be:

"Each message has exactly one distinguished header which is represented by the Body element in the envelope. The body is a special case of a header entry intended for the default actor and with a SOAP mustUnderstand attribute with a value of "1".

The Body is somewhat special in that, in the common case where a message carries one principle request, that SHOULD BE encoded in the Body, with other header entires intended to provide supporting or extension function. For example, the RPC specification (chapter 7) mandates that method calls and responses be in Bodies (not other anonymous header entries), and the fault specification (section 4.4) mandates that Faults be carried as body elements. Certain SOAP processing software may be optimized for the common case in which the Body carries the principle payload or request for the message."

Clarify message path

SOAP implements a model in which various headers entries may or may not be processed at various points along the message path, but no explicitly named point (I.e. explicit actor) may be further along the path than the point at which entries handled by the default actor are processed. In this respect, SOAP implements a model in which the ordering of processing of explicitly targeted messages is indeterminate with respect to each other, and all headers with the default actor are known to be processed after all explicitly targeted messages. Furthermore, SOAP makes no statement one way or the other regarding the possible equivalence of two or more seemingly distinct actors. As an example, one may infer that http://schemas.xmlsoap.org/soap/actor/next is effectively equivalent to the actor next reached by the message. No statement is made regarding the order of processing for entries destined to two or more such “equivalent” actors.

There may be more than one header entry for the default actor; the Body is surely one such, but there maybe others. SOAP provides no explicit ordering of the processing done by the several entries destined to the default actor, and it is in particular possible that some such entries will be processed before or after the Body.
For each header entry, two principal pieces of information are used to determine where it will be processed, and what function is to be performed. The two pieces of information are: (1) a URI specified as the value of an actor attribute and (2) the namespace qualified element name of the immediate child of the header. In the typical case, the intention is that the actor will be used primarily to determine where processing occurs, and the child will determine the type of processing. As Glen Daniels has pointed out, we cannot in fact force those who invent header vocabularies or those who build processing software to observe this distinction. For example, in certain environments, processing software may choose to redirect certain sorts of request to particular servers based on the nature of the request as well as or instead of the value in the actor. Still, we can anticipate that the common case is to observe the distinction, yet be aware of the fact that the distinction does not always apply.

Header entries may be inserted into messages at any point along the path except the end. To

more precisely define the lifecycle and implications of a header entry, let us informally define several terms which do not occur in the SOAP specification itself: we say a header entry is "inserted" at the time it is lexically included in the envelope, and we say we say a header entry is "processed" when the operation encoded in the header is processed at a location specified by the entry (as noted above, location is typically determined by the actor, nature of processing by the child element).
SOAP requires that header entries be removed from the message at the point of processing, but identical entries can be reinserted. This phraseology is intended to discourage a view of individual headers as being multicasts to more than one downstream actor, but instead to imply a hop by hop contract (whether this is a good or bad distinction to attempt is not the point; current versions of the SOAP specification read this way). Although each header entry is "processed" at, at most, one point in the path, we may say that entries can be "read" or "removed prior to processing" at any point at which they are found in a message. So, if a SOAP message passes through a cache node, the processor there can "read" the entire message for caching, not just the header that calls for caching. Similarly, if a header calls for a message to be encrypted, other headers can be "read" and "removed prior to processing", and new headers carrying the encrypted forms of the data can be inserted.

As noted above, header entries in a SOAP envelope are lexically ordered. In other words, they appear one after the other in the XML envelope document. Although the SOAP specification does not specifically say so, some implementers of SOAP processors have proposed that processing can be much more efficient if both the message path (actors to be visited) and perhaps the processing order of header entries destined for a given actor can be strictly top to bottom. Since entries are removed when processed, there is the possibility of merely peeling off header entries in order and processing them. (Question raised since this note was first published: can header entries be reordered prior to retransmission from intermediaries? If so, are there any rules or limitations regarding such reordering?)

One situation in which "read" access occurs is in situations where a single data graph encoded according to SOAP chapter 5 spans multiple header entries. In these cases, it is highly likely that the processor for one entry will need read access to multi-ref structures in others. This tends to cause some tension with the proposal to process header entries strictly in order (comment: multirefs across header entries seems like excess complexity...I'm not sure we should have had it in SOAP v1.1...just my opinion.)

Clarify relationship of fault processing to message path, message pattern and transport bindings

SOAP messages are fundamentally one-way. There is in general no notion of maintaining a record of the path by which a message was delivered, or of returning responses or fault information to any fixed point.

Although SOAP messages are fundamentally one-way, the HTTP binding serves as an example of the possibility for transport bindings to introduce higher level message patterns such as request/response. Presumably, other bindings could also support a request response model, but the SOAP specification does not mandate any particular relation between request/responses implemented by HTTP for example and SMTP (I think it should).

Interestingly, the request/response model is explicitly exploited in a transport independent way by the RPC specification in chapter 7. The specification does not state, but strongly implies that RPC responses will indeed be returned to the originator of the request, and that the transport binding will be responsible for doing the necessary addressing. Certainly the HTTP binding does this.

The relationship of faults to message patterns is even more subtle, and perhaps will need improvement or clarification. Specifically, section 4.4 of the SOAP specification gives information about what must be in a fault message. However, since messages are fundamentally one-way,

and since there is no general-purpose notion of a request/response or other higher level pattern, nothing is stated in general as to where if at all faults are delivered. In this respect, the specification is nearly vacuous: it mandates that a fault message be constructed in certain manner, but it allows it to be dropped on the floor before anyone sees it.

That's what the spec says. Reading between the lines, here's what I think it is really trying to say:

> "Although SOAP messages are fundamentally one-way, SOAP supports the construction of higher level message patterns on top of that. In addition to one-way, this specification includes a well architected notion of request/response, which certain applications and SOAP processors may choose to use when sending SOAP messages. SOAP transport bindings are responsible for supporting the request/response abstraction (always? optionally?). Transport bindings that support request/response are responsible for providing the ability for an intermediary or SOAP end point to return response or fault messages back to the originator of a request. Response messages can be assumed to take the exact reverse path of requests; so response and fault messages can be inspected if desired by the same actors the processed the outbound request {not sure about that..maybe we should make request/resp apply only to the endpoints?}. SOAP RPC uses the request/response message pattern, and is therefore usable only on with transport bindings that support request/response.
>
> The delivery rules for Fault messages are message pattern specific. The rules for the two message patterns supported by this version of the SOAP specification are:
>
> * For one way messages, no mandatory behavior is given for generating or routing fault messages, except that if delivered they must be in the form of section 4.4.
>
> * Faults generated during the transmission and processing of a request message in a request/response pattern are delivered on a best-effort basis to the originator of the entire message sequence (which may or may not be the insertion point of the header that caused the fault), passing as for all responses along the reverse of the same path as the request. Transport bindings that implement the request/response pattern are responsible for either eventually delivering the response or fault, or else eventually reporting a broken connection to the requester. No specific time limit for such reporting is given. The faults generated by problems with the response message itself are treated in the same manner as faults in a one-way message: no mandatory behavior is given for generating or routing fault messages generated by such failed response. "

I think that's about what the experts on the current SOAP specification has said about how it really works. I may not have all the details right, but I do think we would do well to get them nailed down to this level of specificity, and to clarify the specification accordingly. I believe this is necessary even if we decided to make no actual changes to the intended behavior of the system.

Consider and clarify facilities for ensuring that essential processing is done, and in the right order

In the introduction above, I stated:

"Clearly, successful handling of the message depends in many cases not only on having particular header entries understood by associated software, but on ensuring that all essential headers are indeed picked up by some such software (not skipped), and in a suitable order. A core question that we face is how much of the infrastructure to ensure this should be the business of the core SOAP specification, and how much can be delegated to a separate specification."

SOAP today provides a mustUnderstand attribute for use on header entries. Presuming that messages are routed to the appropriate set of intended actors, mustUnderstand allows one to detect situations in which (a) successful processing of the header is essential to the overall safe

handling of the message and (b) the targeted software recognizes that it is indeed the intended actor, but that it is in fact incapable of doing the essential processing. SOAP today does not attempt to provide protection for the situation in which essential processing is missed due to erroneous routing of a message, mislabeling of actors on a header, incorrect configuration of an actor's identity in the processing software, etc. all of which can result in an essential actor being skipped.

Insofar as necessary for the above, consider the use of modules and new mustUnderstand header entries

Overall, this note is being written in the context of a larger discussion aimed at exploring the questions of (1) whether additional specification is in fact needed to provide facilities which will meet the needs of users for robust message exchange and (2) whether such facilities can be cleanly layered in a separate specification. So, we must first consider what users need, and then see how much can be done by layering on the existing SOAP specification. I think this mostly means: can any new features be conveniently and effectively encoded in new mustUnderstand header entries?" See below for discussion of some of the option. I think the answer is: yes, most new facilities can indeed be added in the form of new header entries, but the results are sometimes cumbersome or inconvenient when compared to adding new attributes (which are not subject to mustUnderstand). We will have to weigh the trade-offs.

Better yet, I just had another idea regarding attributes. What if we have a new mustUnderstand header entry that lists the global attributes you must understand:

```
<SOAP-ENV:Envelope  xmlns:a="uria" xmlns:b"urib">

  <SOAP-ENV:Header >
     <SOAP-ENV:mustUnderstandAttributes
          SOAP-ENV:mustUnderstand="true">
         <SOAP-ENV:Attribute>
                a:enhancementAttr
         </SOAP-ENV:Attribute>
         <SOAP-ENV:Attribute>
                b:anotherEnhancementAttr
         </SOAP-ENV:Attribute>
     </SOAP-ENV:mustUnderstandAttributes>
  </SOAP-ENV:Header>

  <!-- following better be understood! -->
  <SOAP-ENV:Body a:enhancementAttr="x"
                 b:enhancementAttr="y">
      ...
  </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

I think this is a bit clunky, but it might work.

A Proposal

In this section I attempt to build on the analysis above to outline some of the options available to us regarding mustUnderstand, message paths. I first quickly summarize the detailed analysis given above, then speculate on options available to us moving forward.

Quick summary of the status quo in SOAP v1.1

As described in detail above, the status quo is that SOAP:

- ...does not in general provide for specification of the order in which header entries are processed, except to indicate that those destined to the anonymous actor come last as a group. There is no other notion of message path in the current specification.
- ...has a somewhat vague notion of request/response and message pattern that we can probably flesh out in useful ways, and without (likely) breaking compatibility with the existing specification.
- ...provides a mustUnderstand attribute that is effective in detecting cases where the message reaches the intended actor, but that actor has inappropriate software. It does not attempt to handle the case where, for whatever reason, messages fail to reach the appropriate processors, or go to the right places in the wrong order. Note that the latter is a plausible failure mode, if SOAP processors make erroneous assumptions in attempting to infer delivery addresses from actor URIs.
- ...note also that mustUnderstand is useful with features that are added in the form of new header entries, but not directly with those represented in other forms such as new attributes. For example, imagine that SOAP version 1.1 had, for some reason, left out support for encodings. One could have added a SOAP-ENV:encodingStyle attribute in a SOAP 1.2, but there would be no way to apply mustUnderstand to that new attribute (except maybe for the trick I outline above).

Design proposal: Ensuring that essential processing occurs, and in an appropriate order

It seems reasonable to assume that users will want software to provide more robust facilities than SOAP currently provides. In other words, users will want to detect situations in which actors are skipped or visited in the wrong order. What is far less clear is whether those facilities should be:

- Built as enhancements to SOAP itself
- Built as separate modules (I.e. using only existing facilities of SOAP 1.1, such as Headers and mustUnderstand), but intended for ubiquitous deployment in more or less every SOAP processor
- Provided in an application-specific manner, integrated into higher level workflow specifications, or otherwise left for implementation outside of the typical SOAP processor. In this case, we should still show that SOAP provides the foundation on which such enhancements can be added.

I think that choosing among these options will take some thought. Each has advantages. To move the discussion along, here is a strawperson proposal for enhancing SOAP itself to deliver the addition function. I am not necessarily advocating this approach, just exploring the possibilities. Take it as a thought piece, and a rough one at that.

First, some assumptions:

- If we want to get into the business of ensuring that essential operations happen in an appropriate order, then we must provide some means of specifying what that order is.
- If some operations, call them A,B,C,D and E are to happen in some predetermined order, it is not sufficient to discover early failures only at the end when "E" is attempted. It may not be safe to begin "C" if "A" or "B" has failed or been missed. (This is the reason I don't think that the proposal Gudge has circulated is quite sufficient; as far as I know, it defers checking until a message has reached its ultimate destination.)
- I can't justify this one, but in general it feels to me as if each step in the process and each header entry should be treated symmetrically, with the possible exception of the Body.

Here is one possible design, consistent with the above requirements and assumptions. In general, it provides for specifying a partial order which must be observed in processing headers *whether or not those headers are destined for distinct actors.* Furthermore, it enhances the mustUnderstand capability to force an error if essential processing fails to occur at the appropriate point in the partial order. The proposal does not introduce a message path in any other sense: it does not indicate the specific URIs to which a message should be sent, although in certain cases that could be inferred from the order of the headers. One embodiment of this proposal would be:

- Any header entry or body can be identified by an ID= attribute (or we might decide to make this SOAP-ENV:ID).
- SOAP-ENV:mustUnderstand continues to have its existing semantic. SOAP-ENV:mustHappen is a new attribute which means that the corresponding header entry must be successfully understood and processed ahead of any dependent entries in the partial order. SOAP-ENV:mustHappen implies SOAP-ENV:mustUnderstand, but it is not an error to specify both (e.g. to facilitate interoperation with SOAP V1 .1 processors)
- When a header entry with SOAP-ENV:mustHappen is processed it is removed from the message (per SOAP v1.1), and is replaced with a new entry, typically empty, with the same ID and the attribute SOAP-ENV:hasHappened. Other attributes are typically dropped, but we'll have to think about this a bit.
- Any header can indicate the IDs of the other headers on which it depends. One possible syntax is:

```
<HEADER>
  <ENTRYA ID="HeaderA" SOAP-ENV:mustHappen="true">
  ...
  </ENTRYA>
  <ENTRYB ID="HeaderB" SOAP-ENV:mustHappen="true">
  ...
  </ENTRYB>
  <ENTRYC SOAP-ENV:dependsOn="HeaderA HeaderB">
  ...
  </ENTRYA>
</HEADER>
```

  Note th at, in this example, A and B can happen in any order, but both must precede the unnamed third header entry. It is an error if in any message (original or relayed) a header entry referred to from "dependsOn" fails to resolve.
- It is an error if any header entry referred to from dependsOn is not labeled either "mustHappen" or "hasHappened".
- Prior to processing any header entry carrying a "dependsOn" attribute, a check is made to ensure that each of the referenced entries "hasHappened".
- It is an error for any header entry targeted to an explicit actor to depend on the Body or on any other header targeted to the anonymous actor. This preserves the SOAP v1.1 model that the anonymous actor is at the end of the path. {…we need to think about http://schemas.xmlsoap.org/soap/actor/next and other similar actors…)
- It is possible to impose dependencies on multiple entries destined for the same actor. It is also possible to impose explicit dependencies on header entries, including the body, destined for the anonymous actor. In this manner, it becomes possible for the first time to explicitly control whether the body is indeed the last header entry processed (typical) or whether others may follow it in deterministic manner (e.g. to log a successful result, help sign a newly prepared response message, etc.)
- All errors described above are reflected as ordinary SOAP faults. Details to be provided.

- Request/response is introduced as a formal message pattern per the analysis above. Consistent with existing SOAP behavior, any errors (including but not limited to hasHappened faults, dangling ID references, etc.) are modeled as ordinary SOAP faults. Consistent with existing SOAP behavior, faults detected in one-way messages and in response messages need not be delivered to any particular point in the message path (I.e. can be dropped). Faults resulting from request messages must be sent back to the originator of the message. The message path for a (successful) request is deemed to end no earlier than the processing of the last header targeted at the anonymous actor in dependency order. In other words, that is the turnaround point for a request/response. Where multiple headers are targeted to the anonymous actor and where order is not completely determined, determination of the end point is application-specific. For the request/response pattern, response messages are deemed to have been generated at the end point of the request, as just described. In other words, the response message is not formally deemed to have been sent until (at earliest) the request message has been fully processed. Of course, software working on behalf of the request may be quietly assembling information for the response as request processing proceeds.
- It is possible, though unusual, to establish dependencies that require a message to revisit a named actor. In other words, Header3 for ActorA follows Header2 for Actor B follows Header1 for ActorA. Such constructs should be used only when the application bindings and underlying transport bindings are known to support them.

**********EXAMPLE**********

Original message:

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Header>
        <nsa:DoThisForA xmlns:nsa="http://nsa.gov/uriA"
              ID="A"
              SOAP-ENV:Actor="uriA"
              SOAP-ENV:mustUnderstand="true"
              SOAP-ENV:mustHappen="true">
        ...
        </nsansa:DoThisForA>

        <nsb:DoThisForB xmlns:nsb=”http://nsb.com/uriB”
              ID="B"
              SOAP-ENV:Actor="uriB"
              SOAP-ENV:mustUnderstand="true"
              SOAP-ENV:mustHappen="true">
        ...
        </nsb:DoThisForB>
        <nsc:DoThisForC xmlns:nsc="http://nsc.com/uriC"
              ID="C"
              SOAP-ENV:Actor="uriC"
              SOAP-ENV:dependsOn="A B">
        ...
        </nsc:DoThisForC>
  </SOAP-ENV:Header>

</SOAP-ENV:Envelope>
```

After successful processing at uriA:

```
<SOAP-ENV:Envelope>
```

```
  <SOAP-ENV:Header>
        <nsa:DoThisForA xmlns:nsa="http://nsa.gov/uriA"
              ID="A" SOAP-ENV:hasHappened="true"/>

        <nsb:DoThisForB xmlns:nsb=”http://nsb.com/uriB”
              ID="B"
              SOAP-ENV:Actor="uriB"
              SOAP-ENV:mustUnderstand="true"
              SOAP-ENV:mustHappen="true">
        ...
        </nsb:DoThisForB>
        <nsc:DoThisForC xmlns:nsc="http://nsc.com/uriC"
              ID="C"
              SOAP-ENV:Actor="uriC"
              SOAP-ENV:dependsOn="A B">
        ...
        </nsc:DoThisForC>
  </SOAP-ENV:Header>
</SOAP-ENV:Envelope>
```

...and after processing at uriB:

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Header>
        <nsa:DoThisForA xmlns:nsa="http://nsa.gov/uriA"
              ID="A" SOAP-ENV:hasHappened="true"/>

        <nsb:DoThisForB xmlns:nsb=”http://nsb.com/uriB”
              ID="B" SOAP-ENV:hasHappened="true"/>

        <nsc:DoThisForC xmlns:nsc="http://nsc.com/uriC"
              ID="C"
              SOAP-ENV:Actor="uriC"
              SOAP-ENV:dependsOn="A B">
        ...
        </nsc:DoThisForC>
  </SOAP-ENV:Header>
</SOAP-ENV:Envelope>
```

Header entry C is processed, because its dependencies can be seen to have been successfully met. The replacement "hasHappened" headers are carried only to support dependency checking, not to propagate any other information about original headers "A" or "B".

(Note: the syntax and details need some work. For example, the replacement header entries A and B are not likely to be schema valid. Maybe the syntax should be:

```
<SOAP-ENV:hasHappened ID=”A”/>
```

We can get these details right if the general idea is deemed to have merit.)

********** **END EXAMPLE**********

A few notes on this design.

- The use of new SOAP-ENV: attributes is convenient, but means that the new facilities are not clearly layered as modules on SOAP V1 .1. There is no way to directly label attributes such as SOAP-ENV:dependsOn as "mustUnderstand". One can easily imagine other representations which, though less convenient, would be encoded entirely within new header elements.
- This design intentionally avoids indicating anything new about the physical routing of messages. As with SOAP today, actor URI's will often represent actual service points to which messages can be delivered, but indirections are possible. So, for example pseudo actors such as ,http://schemas.xmlsoap.org/soap/actor/next can be used and can participate in dependency relationships. Of course, it may be possible to infer a lot about the legal physical routings from the logical dependency relationships.
- Not all header entriess need participate in dependency relationships. While utility may be limited, there are no changes to the current rules for mustUnderstand used in isolation. Use of ID= on mustUnderstand headers is optional, as is mustHappen.

Again, this is not intended as a very polished proposal. It certainly doesn't represent any formal position of Lotus (or our parent company IBM, though they represent themselves in the workgroup anyway). I do hope this is useful in at least pointing out which aspects of the problem may be interrelated.

Is SOAP V1.1 mustUnderstand the right building block?

Given the analysis above, it seems reasonable that we should ask again whether the existing mustUnderstand is indeed the right building block for reliable processing. I am not sure, but let's assume for the moment that we continue to support it, and revisit the question later if necessary. If nothing else, we should be reluctant to make changes that break compatibility with SOAP version 1.1.

Summary

Having aired a specific proposal, let me reiterate that it is only one of the directions that we should explore. Nothing above has proven that the notion of partial order is what users need. Perhaps users need to determine dependencies in more elaborate ways? If so, I don't think we should attempt anything complicated in our core specification. Furthermore, I have questioned utility of mustUnderstand in its current form.

Perhaps the right model is not to do more, but to do less. Maybe we should do nothing in the core protocol to ensure users that their own vocabularies are understood, and suggest that in future mustUnderstand will be applied primarily to enhancements to SOAP itself and to other ubiquitously deployed libraries. I can't prove it, but it's my intuition that mustUnderstand's weakness is in situations where you accidentally skip actors where very particular sorts of processing are to occur. Perhaps that is less of an issue for core features of SOAP itself, which one would expect would be "understood" at more or less every actor along the way?

In any case, I think a detailed clarification of SOAP V1 .1, such as the one I attempted in the first section would be very useful. Perhaps this manifests itself in the abstract model, but I think it is ultimately the SOAP (or XMLP) specification that should be made crisp and unambiguous. That is the document that most of our users will read.

I do apologize for having sent such a long note. I've been carrying most of these ideas around for awhile, and in my own mind they are all interrelated. Thank you for your patience in considering them.

Noah Mendelsohn

Lotus Development

[1] http://www.w3.org/TR/SOAP/#_Toc478383504
...more references would be useful, I'm sure...