SALUD
Internet-Draft
Intended status: Informational
Expires: August 24, 2017

D. Worley
Ariadne

February 20, 2017

A Simpler Method for Processing Alert-Info URNs
draft-worley-alert-info-fsm-06

Abstract

The "alert" namespace of uniform resource names (URNs) can be used in the Alert-Info header field of Session Initiation Protocol (SIP) requests and responses to inform a VoIP telephone (user agent) of the characteristics of the call that the user agent has originated or terminated. Based on the URNs in the Alert-Info header field, the user agent must select the best available signal to present to its user to indicate the characteristics of the call. This document describes a method by which a user agent's designer can, based on the user agent's signals and their meanings, construct a finite state machine (FSM) to process the URNs to select a signal in a way that obeys the restrictions given in the definition of the "alert" URN namespace. In many situations, the resulting FSM is simpler and faster than the selection algorithm described in [RFC7462].

Table of Contents

1. Introduction

When a SIP user agent server receives an incoming INVITE request, it chooses an alerting signal (the ring tone) to present to its user (the called user) by processing the Alert-Info header field(s) in the incoming INVITE request [RFC3261]. Similarly, a SIP user agent client determines an alerting signal (the ringback tone) to present to its user (the calling user) by processing the Alert-Info header field(s) in the incoming provisional response(s) to its outgoing INVITE request.

[RFC3261] envisioned that the Alert-Info header field value would be a URL that the user agent could use to retrieve a signal. This usage has security problems and is inconvenient to implement in practice. [RFC7462] introduced an alternative practice: The Alert-Info values could be URNs in the "alert" URN namespace which specify features of the call or of the signal that should be signaled to the user. [RFC7462] defined a large set of "alert" URNs and procedures for extending the set.

However, a user agent is unlikely to provide alerting signals that can explicitly render more than a small subset of the possible combinations of "alert" URNs, so the user agent is frequently required to select an alerting signal which renders only a subset of the information in the Alert-Info header field(s). The requirements for the process of selecting an alerting signal based on "alert" URNs are given in section 11.1 of [RFC7462] and can be described as follows:

> The "alert" URNs are processed from left to right. Each "alert" URN has precedence over all URNs that follow it, and its interpretation is subordinate to all URNs that precede it.
>
> As each URN is processed, one of the UA's signals is chosen which expresses that URN as far as can be done without reducing the degree to which any of the preceding URNs were expressed by the signal chosen for the preceding URN. Thus, as processing proceeds, the chosen signals become increasingly specific and contain more information, but all of the information about a particular URN that is expressed by the signal chosen for that URN is also expressed by the signals chosen for all following URNs.
>
> If the entirety of the current URN cannot be expressed by any allowed signal, then in turn, each of the trailing alert-ind-parts (the sections separated by colons) is removed until the reduced URN can be expressed by some signal which also expresses at least the same reduced versions of the preceding URNs that were expressed by the signal chosen for the preceding URN. This can be

described as "a signal that expresses as much of the current URN as possible while still expressing as much of the previous URNs as the preceding signal did."

So, for instance, consider processing

```
Alert-Info: urn:alert:category-a:part-a1:part-a2,
            urn:alert:category-b:part-b1:part-b2
```

If the UA has no signal for urn:alert:category-a:part-a1:part-a2, it removes part-a2 from the URN and checks whether it has a signal for the less-specific URN urn:alert:category-a:part-a1. If it has no signal for that URN, it gives up on the URN entirely (since urn:alert:category-a doesn't exist, and can be considered to express nothing about the call), and the chosen signal is the default signal of the UA, the signal that is used when there is no Alert-Info.

But let us suppose the UA has a signal for urn:alert:category-a:part-a1, and chooses that signal when processing the first URN. All processing after this point will be restricted to signals that express urn:alert:category-a:part-a1, or a more specific URN of the category category-a.

The UA then goes on to examine the next URN, urn:alert:category-b:part-b1:part-b2. If there is a signal that expresses both urn:alert:category-a:part-a1 and urn:alert:category-b:part-b1:part-b2, then the UA chooses that signal. If there is no such signal, the second URN is reduced to urn:alert:category-b:part-b1, and the UA checks for a signal that expresses that URN along with urn:alert:category-a:part-a1. If there is no such signal that matches that relaxed requirement, the second URN is reduced to urn:alert:category-b, which is discarded, and the chosen signal for the first URN is chosen for the second URN. In any case, all processing after this point will be restricted to signals that express urn:alert:category-a:part-a1 or a more specific URN of the category category-a, and also express the chosen part of urn:alert:category-b:part-b1:part-b2.

This process is continued until the last "alert" URN is processed; the signal chosen for the last URN is the signal that the UA uses.

Section 12 of [RFC7462] gives one possible algorithm for selecting a signal which satisfies the requirements of section 11.1. That algorithm can be used regardless of the set of alerting signals that the user agent provides and their specified meanings. This demonstrates that the rules can always be satisfied. However, the algorithm is complex and slow.

The purpose of this document is to describe an easier method for selecting signals that conforms to section 11.1: Once the user agent designer has chosen a set of signals and the URNs that they express, a finite state machine is constructed that selects alerting signals based on the URNs in the Alert-Info header field(s) in a SIP message.

- The designer selects the set of signals that the user agent produces, matching each signal to the "alert" URN or the combination of "alert" URNs which are the meaning carried by the signal.

- Based on the user agent's signals and their meanings, the designer constructs an "alphabet" containing a finite number of symbols; each possible "alert" URN is mapped into one particular symbol.

- The designer constructs a finite state machine (FSM) whose input is the alphabet of symbols and whose states describe information extracted from the Alert-Info URNs.

- Each state of the FSM has an attached signal. Processing the Alert-Info URNs will leave the FSM in some particular state; the UA renders the signal that is attached to that final state.

To select a ring tone or ringback tone based on a SIP message, the user agent processes the "alert" URNs in the Alert-Info header field from left to right. Initially the FSM is in a designated initial state. The user agent maps each successive URN into the corresponding symbol, and then executes the state transition of the FSM specified by the symbol. The state of the FSM after processing the URNs determines which signal the user agent will render to the user.

Note that the user agent generally has two FSMs, because a user agent usually wants to signal different information in ring tones than it signals in ringback tones. One FSM is used to select the ring tone to render for an incoming INVITE request. The other FSM is used to select the ringback tone to render based on an incoming provisional response to an outgoing INVITE request. Both FSMs are constructed in the same way, but the constructions are based on different lists of signals and corresponding URNs.

All of the steps of the method after the designer has selected the signals and their URNs are algorithmic, and the algorithm assures that the operation of the FSM will satisfy the constraints of section 11.1 of [RFC7462]. An implementation of the algorithmic steps is provided in [code].

In simple situations, a suitable FSM or equivalent ad-hoc code can be constructed by hand using ad-hoc analysis. Generally, this is only practical in situations where a small number of alert categories and alert indications are signaled and the categories interact in a simple, uniform way. E.g., the examples in Section 5 and Section 6 could be constructed by ad-hoc analysis. But automatic processing is valuable if the situation is too complicated to construct a correct FSM by ad-hoc analysis, or if the set of signals will change too frequently for human production to be economical.

2. Selecting the Signals and Their Corresponding "alert" URNs

The designer must select signals that the UA will generate and define the meanings that the signals will have to the user. Based on this, the designer determines for each signal the "alert" URN or combination of "alert" URNs that indicate that meaning in SIP messages, and consequently should elicit that signal from the UA.

For example, suppose the UA has a particular ring tone for calls from an external source. A call from an exteral source is marked with the URN urn:alert:source:external (specified in section 9 of [RFC7462]). Thus, the table of signals includes:

Signal	URN(s)
external source	urn:alert:source:external

Similarly, if the UA has a particular ring tone for calls from an internal source, the table includes:

Signal	URN(s)
internal source	urn:alert:source:internal

If the UA has ring tones for calls that are marked as having higher or lower priority, then the table includes:

Signal	URN(s)
high priority	urn:alert:priority:high
low priority	urn:alert:priority:low

Note that the UA must be able to signal for a message that has no "alert" URNs in the Alert-Info header field, which means that there must always be a default signal which has zero corresponding URNs:

Signal	URN(s)
default	(none)

A signal can be defined to indicate a combination of conditions. For instance, a signal that is used only for high-priority, internal-source calls expresses two URNs, and will only be used when both URNs are present in Alert-Info:

Signal	URN(s)
high priority, internal source	urn:alert:priority:high, urn:alert:source:internal

A signal can be defined to cover a number of related conditions by specifying a URN that is the common prefix of the URNs for the various conditions. For instance, the URNs for "recall due to callback", "recall due to call hold", and "recall due to transfer" all start with urn:alert:service:recall, and so one signal can be provided for all of them by:

Signal	URN(s)
recall	urn:alert:service:recall

But if a specific signal is also provided for "recall due to callback" by this entry:

Signal	URN(s)
recall generally	urn:alert:service:recall
recall due to callback	urn:alert:service:recall:callback

then if the message contains urn:alert:service:recall:callback, the "recall due to callback" signal will be chosen instead of "recall generally" because the UA chooses the signal that most completely expresses the information in the Alert-Info header field.

The designer may wish to define extension URNs that provide more specific information about a call than the standard "alert" URNs do. One method is to add additional components to standard URNs. For instance, an extra-high priority could be indicated by the URN urn:alert:priority:high:extra-high@example. The final "extra-high@example" is an "alert-ind-part" that is a private extension. (See sections 7 and 10.2 of [RFC7462] for a discussion of private extensions.) In any case, adding an alert-ind-part to a URN makes its meaning more specific, in that any call to which the longer URN can be applied can also have the shorter URN applied. In this case,

"extra-high-priority calls" are considered a subset of "high-priority calls".

Signal	URN(s)
high priority	urn:alert:priority:high
extra high priority	urn:alert:priority:high:extra@example.com

Of course, for this extension to be useful, the senders of SIP messages (e.g., other UAs) must generate the extension URN in suitable circumstances.

In some circumstances, the designer may want to create an entirely new category of "alert" URNs to indicate a type of information that is not indicated by any standard category of URNs. In that case, the designer uses a private extension as the alert-category (the third component of the URN), combined with whatever alert-ind-part (fourth component) values are desired. For example, a simplified version of the U.S. military security designations could be:

Signal	URN(s)
unclassified	urn:alert:security@example:unclassified
confidential	urn:alert:security@example:confidential
secret	urn:alert:security@example:secret
top-secret	urn:alert:security@example:top-secret

The designer should ensure that the new alert-category is orthogonal to all defined standard alert-categories, in that any combination of one of the new URNs with one of the standard URNs is meaningful in that there could be a message carrying both URNs.

In addition, the set of alert-ind-parts for the new alert-category should be comprehensive and disjoint, in that every message can be described by exactly one of them.

3. General Considerations for Processing Alert-Info

In this section, we will discuss various considerations which arise when processing Alert-Info. These have to be taken care of properly in order to conform to the standards, as well as to endure a good user experience. But since they are largely independent of the generated finite state machine and its processing, they are gathered here in a seperate section.

The UA may have a number of different finite state machines (FSMs) for processing URNs. Generally, there will be different FSMs for processing Alert-Info in incoming INVITE requests and for incoming

provisional responses to outgoing INVITE requests. But any situation that changes the set of signals that the UA is willing to generate specifies a different set of signals and corresponding URNs, and thus generates a different FSM. For example, if a call is active on the UA, all audible signals may become unavailable, or audible signals may be available only if urn:alert:priority:high is specified.

Similarly, if the set of signals is customized by user action or local policy, the generated FSM must be updated. This can be done by regenerating it according to the method described here, or by generating a "generic" FSM and instantiating it based on the available signals. (See Section 12 for a discussion of this.)

Note that the values in an Alert-Info header field are allowed to be URIs of any scheme, and within the "urn" scheme, are allowed to have any namespace [RFC3261]. The processing of URIs that are not "alert" URNs is not considered by this document, nor is that processing specified by [RFC7462]. But the algorithm designer must consider what to do with such URIs if they are encountered. The simplest choice is to ignore them. Alternatively, the algorithm may examine the URI to determine if it names an alerting signal or describes how to retrieve an alerting signal, and if so, choose to render that signal, rather than processing the "alert" URNs to select a signal. In any case, the remainder of this document assumes that the signal is to be chosen based on the "alert" URNs in Alert-Info, and that all Alert-Info URIs that are not "alert" URNs have been removed.

The UA may also receive "alert" URNs that are semantically invalid in various ways. E.g., the URN may have only three components, despite that all valid "alert" URNs have at least one alert-ind-part, and thus four components. The only useful strategy is to ignore such URNs (and possibly log them for analysis).

The method described here is robust in its handling of categories and alert-ind-parts which are unknown to the UA, and as a consequence, it is also robust if they are not valid standardized URNs. Thus, these error conditions need not be handled specially.

4. Constructing the Finite State Machine for a Very Simple Example

Constructing the FSM involves:

1. Listing the URNs which are expressed by the various signals of the user agent.

2. From the expressed URNs, constructing the finite alphabet of symbols into which input URNs are mapped and which drive the state transitions of the FSM.

3. Constructing the states of the FSM and the transitions between them.

4. Selecting a signal to be associated with each FSM state.

We will explain the process using a very simple example in which there are two signals, one expressing "internal source" and one expressing "external source", along with a default signal (for when there is no source information to signal). The "internal source" signal expresses urn:alert:source:internal, and the "external source" signal expresses urn:alert:source:external.

4.1. Listing the Expressed URNs

The first step is to establish for each of the user agent's signals what call characteristics it represents, which is to say, the set of "alert" URNs which are its information content.

Signal	URN(s)
default	(none)
internal source	urn:alert:source:internal
external source	urn:alert:source:external

From the totality of these expressed URNs, the designer can then determine which sets of URNs must be distinguished from each other. In our simple example, the expressed URNs are:

urn:alert:source:external
urn:alert:source:internal

4.2. Constructing the Alphabet

In order to reduce the infinite set of possible "alert" URNs to a finite alphabet of input symbols which cause the FSM's transitions, the designer must partition the "alert" URNs into a finite set of categories.

Once we've listed all the expressed URNs, we can list all of the alert-categories that are relevant to the user agent's signaling; "alert" URNs in any other alert-category cannot affect the signaling and can be ignored. (The easiest method to achieve this is to skip over them during Alert-Info processing. A more formal method is to map all of these URNs into one "Other" symbol, and then for each state of the FSM, have the Other symbol transition to that same state.)

Within each relevant alert-category, we now define a distinct symbol for every expressed URN and for all of their "ancestor" URNs (those that can be created by removing one or more trailing alert-ind-parts). In order to name the symbols in a way that distinguishes them from the corresponding URNs, we remove the initial "urn:alert:" and capitalize each alert-ind-part. Thus in our example, we get these symbols:

```
Source
Source:External
Source:Internal
```

Note that there is a "Source" symbol even though there is no corresponding URN. (urn:alert:source is not a valid URN -- see [RFC7462] section 7 -- although the processing algorithm must be prepared to screen out such a purported URN if it appears in the Alert-Info header field.) However, its existance as a symbol will be useful later when we construct the FSM.

For each of these symbols, we add a symbol that classifies URNs that extend the symbol's corresponding URN with alert-ind-parts that cannot be expressed:

```
Source:Other
Source:External:Other
Source:Internal:Other
```

The latter two classify URNs like urn:alert:source:external:foo@example, which extend URNs that we already have symbols for. The first is for classifying URNs, such as urn:alert:source:bar@example, which have first alert-ind-parts that contradict all the "source" URNs that the user agent can signal.

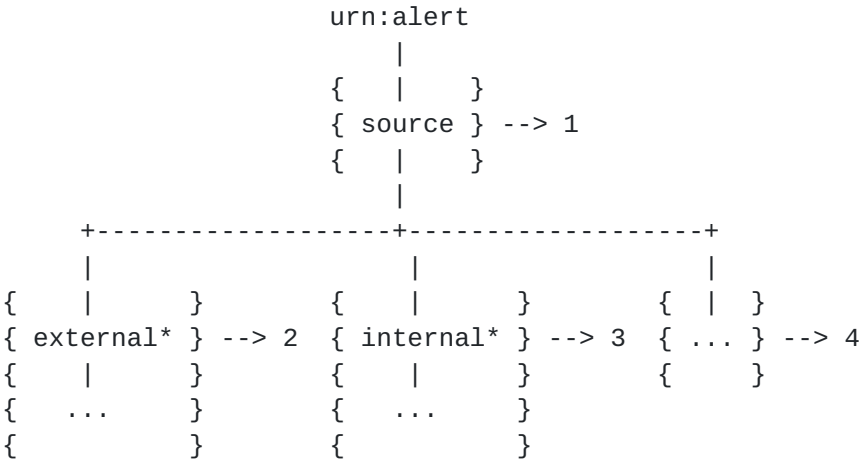
We can then simplify the set of symbols by removing the ones like Source:External:Other and Source:Internal:Other that consist of adding "Other" to a symbol which corresponds to an expressed URN which is not ancestral to any other expressed URN.

This leaves the following symbols for the "source" category:

```
Source
Source:External
Source:Internal
Source:Other
```

These can be visually summarized by showing the infinite tree of possible source "alert" URNs and how it is partitioned into subtrees that map to each of these symbols. We also mark with "*" the

expressed URNs, and by implication, the symbols that correspond to them.



1 = Source
2 = Source:External
3 = Source:Internal
4 = Source:Other

4.3. Constructing the States and Transitions

The user agent processes the Alert-Info URNs left-to-right using a finite state machine (FSM), with each successive URN causing the FSM to transition to a new state. Each state of the FSM records the information which has so far been extracted from the URNs. The state of the FSM after processing all the URNs determines which signal the user agent will render to the user.

We label each state with a set of symbols, one from each relevant category, which describe the information that's been extracted from all of the URNs that have so far been processed. The initial state is labeled with the "null" symbols that are just the category names, because no information has yet been recorded. In our simple example, the initial state is labeled "Source", since that's the only relevant category.

State: Source (initial state)

Each state has a corresponding alerting signal, which is the signal that the user agent will produce when URN processing leaves the FSM in that state. The signal is the one that best expresses the information that has been extracted from the URNs. Usually the choice of signal is obvious to the designer, but there are certain constraints that the choice must satisfy. The main constraint is that the signal's expressed URNs must be semantic supersets of (i.e.,

identical to or a prefix of) the URNs corresponding to the symbols in the state's label. In particular, if the expressed URN of the signal in a certain category is shorter than the state's label, we show that in the state's name by putting parentheses around the trailing part of the symbol that is not expressed by the signal. For instance, if the symbol in the label is "Source:External" but the signal only expresses "Source" (i.e., no "source" URN at all), then the symbol in the label is modified to be "Source:(External)".

The reason for this unintuitive construction is that in some states, the FSM has recorded information that the chosen signal cannot express.

Note that the parentheses are part of the state name, so in some circumstances there may be two or more distinct states labeled with the same symbols, but with different placement of parentheses within the symbols. These similar state names are relevant when the FSM can record information from multiple "alert" URNs but cannot express all of them -- depending on the order in which the URNs appear, the UA may have to render different signals, so it needs states that record the same information but render different subsets of that information.

The initial state's label is the string of null symbols for the relevant categories, so the only allowed signal is the default signal, which expresses no URNs:

```
State: Source (initial state)
Signal: default
```

From each state, we must construct the transition for each possible input symbol. For a particular state and symbol, we construct the label of the destination state by combining the input symbol with the symbol in the start state's label for the same category. If one of the symbols is a prefix of the other, we select the longer one; if not, we select the symbol in the start state's label.

Thus, in our simple example, the initial state has the following transitions:

```
State: Source (initial state)
Signal: default
Transitions:
    Source:External -> Source:External
    Source:Internal -> Source:Internal
    Source:Other -> Source:Other
```

In all of these transitions, the input symbol is compatible with the matching label of the state, "Source", so the destination state's label is the full input symbol.

However, there is a further constraint on the destination state: Its signal must express URNs that at least contain the expressed URNs of the signal of the start state. Within that constraint, and being compatible with the destination state's label, for the category of the input URN, the destination state's signal must express the longest URN that can be expressed by any signal.

In our example, this means that the destination Source:External state has the "external source" signal, which expresses urn:alert:source:external. Since that signal expresses all of the state's label, it is the chosen state. Similarly, the destination Source:Internal state has the "internal source" signal. But for the transition on input Source:Other, the "Source:Other" state must have the default signal, as there is no signal that expresses urn:alert:source:[some-unknown-alert-ind-part]. So the destination state is "Source:(Other)", where the parentheses record that the "Other" part of the label is not expressed by the state's signal.

Thus, the initial state and the states it can transition to are:

```
    State: Source (initial state)
    Signal: default
    Transitions:
        Source:External -> Source:External
        Source:Internal -> Source:Internal
        Source:Other -> Source:(Other)

    State: Source:External
    Signal: external source (urn:alert:source:external)

    State: Source:Internal
    Signal: internal source (urn:alert:source:internal)

    State: Source:(Other)
    Signal: default
```

Looking at the state Source:External, we see that it is incompatible with all input symbols other than Source:External, and thus all of its transitions are to itself:

```
State: Source:External
Signal: external source (urn:alert:source:external)
Transitions:
    Source:External -> Source:External
    Source:Internal -> Source:External
    Source:Other -> Source:External
```

and similarly:

```
State: Source:Internal
Signal: internal source (urn:alert:source:internal)
Transitions:
    Source:External -> Source:Internal
    Source:Internal -> Source:Internal
    Source:Other -> Source:Internal

State: Source:(Other)
Signal: default
Transitions:
    Source:External -> Source:(Other)
    Source:Internal -> Source:(Other)
    Source:Other -> Source:(Other)
```

4.4. Summary

The FSM can be constructed by processing the file "very-simple.txt" with the program "alert-info-fsm.py" in [code]. The program's output shows the stages of the construction, which are:

1. The signals have the meanings:

Signal	URN(s)
default	(none)
internal source	urn:alert:source:internal
external source	urn:alert:source:external.

2. The expressed URNs are

```
urn:alert:source:external
urn:alert:source:internal
```

3. The relevant categories of "alert" URNs are only:

```
source
```

4. Thus, the infinite universe of possible "alert" URNs can be reduced to these symbols, which are the categories of URNs that

are different in ways that are significant to the resolution process:

```
Source
Source:External
Source:Internal
Source:Other
```

5. The FSM is:

```
State: Source (initial state)
Signal: default
Transitions:
    Source:External -> Source:External
    Source:Internal -> Source:Internal
    Source:Other -> Source:(Other)

State: Source:External
Signal: external source (urn:alert:source:external)
Transitions:
    Source:External -> Source:External
    Source:Internal -> Source:External
    Source:Other -> Source:External

State: Source:Internal
Signal: internal source (urn:alert:source:internal)
Transitions:
    Source:External -> Source:Internal
    Source:Internal -> Source:Internal
    Source:Other -> Source:Internal

State: Source:(Other)
Signal: default
Transitions:
    Source:External -> Source:(Other)
    Source:Internal -> Source:(Other)
    Source:Other -> Source:(Other)
```

* Each state is labeled by a set of symbols which describe the information which has been extracted from the URNs so far.

* Each state has a signal which is a semantic superset of the state's label, i.e., the signal's expressed URNs match the initial portion of the label symbols. If Alert-Info processing finishes with the FSM in a state, the user agent will render the state's signal to the user.

* The state's label is marked to show what subset of the symbols are expressed by the state's signal. Two states can have the same label but different signals.

* If a transition's input symbol is compatible with (is a semantic subset) of the start state's label for that category, the destination state's label is updated with the input symbol. If not, the destination state is the start state state. This is how the state's label records what information has been accumulated while processing the Alert-Info URNs.

* A transition's destination state has a signal which semantically subsets the start state's signal as much as possible in the category of the input symbol. (In most cases, the choice of signal is unique. In rare cases there may be more than one signal that meets this criterion, so the designer may have some flexibility.)

4.5. Examples of Processing Alert-Info URNs

In the trivial case where the user agent receives no Alert-Info URNs, then processing begins and ends with the FSM in the initial state and selects the default signal.

If the user agent receives

```
Alert-Info: <urn:alert:source:internal>
```

then processing progresses:

```
State: Source
    Process: Source:Internal (urn:alert:source:internal)
State: Source:Internal
Signal: internal source
```

If the user agent receives

```
Alert-Info: <urn:alert:source:external>,
    <urn:alert:source:internal>
```

then processing progresses:

```
State: Source
    Process: Source:External (urn:alert:source:external)
State: Source:External
    Process: Source:Internal (urn:alert:source:internal)
State: Source:External
Signal: external source
```

If the user agent receives

```
Alert-Info: <urn:alert:source:unclassified>,
    <urn:alert:source:internal>
```

then processing progresses:

```
State: Source
    Process: Source:Other (urn:alert:source:unclassified)
State: Source:(Other)
    Process: Source:Internal (urn:alert:source:internal)
State: Source:(Other)
Signal: default
```

If the user agent receives

```
Alert-Info: <urn:alert:priority:high>,
    <urn:alert:source:internal>
```

then processing progresses:

```
State: Source
    Ignore: urn:alert:priority:high
State: Source
    Process: Source:Internal (urn:alert:source:internal)
State: Source:Internal
Signal: internal source
```

5. Example with "source" and "priority" URNs

Now consider an example where the user agent can signal "external source", "internal source", "low priority", and "high priority" individually or in any combination of source and priority, along with a default signal. This example is essentially the cartesian product of two copies of the example in Section 4, one dealing with the call's source and one dealing with the call's priority. So there is a total of 9 signals:

Signal	URN(s)
default	(none)
external source	urn:alert:source:external
internal source	urn:alert:source:internal
low priority	urn:alert:priority:low
low priority/external source	urn:alert:priority:low, urn:alert:source:external
low priority/internal source	urn:alert:priority:low, urn:alert:source:internal
high priority	urn:alert:priority:high
high priority/external source	urn:alert:priority:high, urn:alert:source:external
high priority/internal source	urn:alert:priority:high, urn:alert:source:internal

The expressed URNs are:

```
urn:alert:source:external
urn:alert:source:internal
urn:alert:priority:low
urn:alert:priority:high
```

The relevant categories of "alert" URNs are only:

```
source
priority
```

The alphabet of symbols is:

```
Source
Source:External
Source:Internal
Source:Other
Priority
Priority:Low
Priority:High
Priority:Other
```

The 16 states are as follows, where 10 states have a simple structure because from them, no further information can be recorded.

```
State: Priority/Source
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source
    Priority:High -> Priority:High/Source
    Priority:Low -> Priority:Low/Source
    Source:Other -> Priority/Source:(Other)
    Source:External -> Priority/Source:External
    Source:Internal -> Priority/Source:Internal

State: Priority:(Other)/Source
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source
    Priority:High -> Priority:(Other)/Source
    Priority:Low -> Priority:(Other)/Source
    Source:Other -> Priority:(Other)/Source:(Other)
    Source:External -> Priority:(Other)/Source:External
    Source:Internal -> Priority:(Other)/Source:Internal

State: Priority:(Other)/Source:(Other)
Signal: default
Transitions:
    any -> Priority:(Other)/Source:(Other)

State: Priority:(Other)/Source:External
Signal: external source
Transitions:
    any -> Priority:(Other)/Source:External

State: Priority:(Other)/Source:Internal
Signal: internal source
Transitions:
    any -> Priority:(Other)/Source:Internal

State: Priority:High/Source
Signal: high priority
Transitions:
    Priority:Other -> Priority:High/Source
    Priority:High -> Priority:High/Source
    Priority:Low -> Priority:High/Source
    Source:Other -> Priority:High/Source:(Other)
    Source:External -> Priority:High/Source:External
    Source:Internal -> Priority:High/Source:Internal
```

```
State: Priority:High/Source:(Other)
Signal: high priority
Transitions:
    any -> Priority:High/Source:(Other)

State: Priority:High/Source:External
Signal: high priority/external source
Transitions:
    any -> Priority:High/Source:External

State: Priority:High/Source:Internal
Signal: high priority/internal source
Transitions:
    any -> Priority:High/Source:Internal

State: Priority:Low/Source
Signal: low priority
Transitions:
    Priority:Other -> Priority:Low/Source
    Priority:High -> Priority:Low/Source
    Priority:Low -> Priority:Low/Source
    Source:Other -> Priority:Low/Source:(Other)
    Source:External -> Priority:Low/Source:External
    Source:Internal -> Priority:Low/Source:Internal

State: Priority:Low/Source:(Other)
Signal: low priority
Transitions:
    any -> Priority:Low/Source:(Other)

State: Priority:Low/Source:External
Signal: low priority/external source
Transitions:
    any -> Priority:Low/Source:External

State: Priority:Low/Source:Internal
Signal: low priority/internal source
Transitions:
    any -> Priority:Low/Source:Internal
```

```
State: Priority/Source:(Other)
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source:(Other)
    Priority:High -> Priority:High/Source:(Other)
    Priority:Low -> Priority:Low/Source:(Other)
    Source:Other -> Priority/Source:(Other)
    Source:External -> Priority/Source:(Other)
    Source:Internal -> Priority/Source:(Other)

State: Priority/Source:External
Signal: external source
Transitions:
    Priority:Other -> Priority:(Other)/Source:External
    Priority:High -> Priority:High/Source:External
    Priority:Low -> Priority:Low/Source:External
    Source:Other -> Priority/Source:External
    Source:External -> Priority/Source:External
    Source:Internal -> Priority/Source:External

State: Priority/Source:Internal
Signal: internal source
Transitions:
    Priority:Other -> Priority:(Other)/Source:Internal
    Priority:High -> Priority:High/Source:Internal
    Priority:Low -> Priority:Low/Source:Internal
    Source:Other -> Priority/Source:Internal
    Source:External -> Priority/Source:Internal
    Source:Internal -> Priority/Source:Internal
```

An example of processing that involves multiple "source" URNs and one "priority" URN:

```
Alert-Info: <urn:alert:source:internal>,
    <urn:alert:source:unclassified>,
    <urn:alert:priority:high>

State: Source/Priority
    Process: Source:Internal (urn:alert:source:internal)
State: Source:Internal/Priority
    Process: Source:(Other) (urn:alert:source:unclassified)
State: Source:Internal/Priority
    Process: Priority:High (urn:alert:priority:high)
State: Source:Internal/Priority:High
Signal: internal source/high priority
```

6. Example 1 of RFC 7462

A more complicated example is in section 12.2.1 of [RFC7462]. It is like the example in Section 5 of this document, except that the user agent can only signal "external source", "internal source", "low priority", and "high priority" individually but not in combination, as well as a default signal:

Signal	URN(s)
default	(none)
internal source	urn:alert:source:external
external source	urn:alert:source:internal
high low	urn:alert:priority:low
high priority	urn:alert:priority:high

The signals can express the following URNs:

```
urn:alert:source:external
urn:alert:source:internal
urn:alert:priority:low
urn:alert:priority:high
```

The relevant categories of "alert" URNs are:

```
source
priority
```

The alphabet of symbols is:

```
Source
Source:External
Source:Internal
Source:Other
Priority
Priority:Low
Priority:High
Priority:Other
```

In this example, the FSM has 20 states because both "source" and "priority" URNs are recorded, but the order in which the two appear affects the signal:

```
State: Priority/Source
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source
    Priority:High -> Priority:High/Source
    Priority:Low -> Priority:Low/Source
    Source:Other -> Priority/Source:(Other)
    Source:External -> Priority/Source:External
    Source:Internal -> Priority/Source:Internal
```

State Priority:(Other)/Source can transition to states that can signal source, because the recorded priority can't be signaled and thus does not block the signalling of the source:

```
State: Priority:(Other)/Source
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source
    Priority:High -> Priority:(Other)/Source
    Priority:Low -> Priority:(Other)/Source
    Source:Other -> Priority:(Other)/Source:(Other)
    Source:External -> Priority:(Other)/Source:External
    Source:Internal -> Priority:(Other)/Source:Internal

State: Priority:(Other)/Source:(Other)
Signal: default
Transitions:
    any -> Priority:(Other)/Source:(Other)

State: Priority:(Other)/Source:External
Signal: external source
Transitions:
    any -> Priority:(Other)/Source:External

State: Priority:(Other)/Source:Internal
Signal: internal source
Transitions:
    any -> Priority:(Other)/Source:Internal
```

Because there are no signals for combinations of "source" and "priority" URNs, processing a "source" URN from the state Priority:High/Source leads to a state that records the priority information, but does not signal it:

```
State: Priority:High/Source
Signal: high priority
Transitions:
    Priority:Other -> Priority:High/Source
    Priority:High -> Priority:High/Source
    Priority:Low -> Priority:High/Source
    Source:Other -> Priority:High/Source:(Other)
    Source:External -> Priority:High/Source:(External)
    Source:Internal -> Priority:High/Source:(Internal)

State: Priority:High/Source:(Other)
Signal: high priority
Transitions:
    any -> Priority:High/Source:(Other)
```

From the state Priority:High/Source, "source" URNs transition to states that record both source and priority but signal only priority, one of which is Priority:High/Source:(External). But from Priority/Source:External, the symbol Priority:High transitions to the state Priority:(High)/Source:External, which records the same information but signals the source, not the priority. -- One state is reached by processing a "priority" URN and then a "source" URN, whereas the other is reached by processing a "source" URN and then a "priority" URN.

```
State: Priority:High/Source:(External)
Signal: high priority
Transitions:
    any -> Priority:High/Source:(External)

State: Priority:High/Source:(Internal)
Signal: high priority
Transitions:
    any -> Priority:High/Source:(Internal)
```

And similarly for Priority:Low/Source:

```
State: Priority:Low/Source
Signal: low priority
Transitions:
    Priority:Other -> Priority:Low/Source
    Priority:High -> Priority:Low/Source
    Priority:Low -> Priority:Low/Source
    Source:Other -> Priority:Low/Source:(Other)
    Source:External -> Priority:Low/Source:(External)
    Source:Internal -> Priority:Low/Source:(Internal)
```

```
State: Priority:Low/Source:(Other)
Signal: low priority
Transitions:
    any -> Priority:Low/Source:(Other)

State: Priority:Low/Source:(External)
Signal: low priority
Transitions:
    any -> Priority:Low/Source:(External)

State: Priority:Low/Source:(Internal)
Signal: low priority
Transitions:
    any -> Priority:Low/Source:(Internal)

State: Priority/Source:(Other)
Signal: default
Transitions:
    Priority:Other -> Priority:(Other)/Source:(Other)
    Priority:High -> Priority:High/Source:(Other)
    Priority:Low -> Priority:Low/Source:(Other)
    Source:Other -> Priority/Source:(Other)
    Source:External -> Priority/Source:(Other)
    Source:Internal -> Priority/Source:(Other)

State: Priority/Source:External
Signal: external source
Transitions:
    Priority:Other -> Priority:(Other)/Source:External
    Priority:High -> Priority:(High)/Source:External
    Priority:Low -> Priority:(Low)/Source:External
    Source:Other -> Priority/Source:External
    Source:External -> Priority/Source:External
    Source:Internal -> Priority/Source:External

State: Priority:(High)/Source:External
Signal: external source
Transitions:
    any -> Priority:(High)/Source:External

State: Priority:(Low)/Source:External
Signal: external source
Transitions:
    any -> Priority:(Low)/Source:External
```

```
State: Priority/Source:Internal
Signal: internal source
Transitions:
    Priority:Other -> Priority:(Other)/Source:Internal
    Priority:High -> Priority:(High)/Source:Internal
    Priority:Low -> Priority:(Low)/Source:Internal
    Source:Other -> Priority/Source:Internal
    Source:External -> Priority/Source:Internal
    Source:Internal -> Priority/Source:Internal

State: Priority:(High)/Source:Internal
Signal: internal source
Transitions:
    any -> Priority:(High)/Source:Internal

State: Priority:(Low)/Source:Internal
Signal: internal source
Transitions:
    any -> Priority:(Low)/Source:Internal
```

As an example of processing, if the user agent receives

```
Alert-Info: <urn:alert:source:internal>
```

then processing progresses:

```
State: Priority/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority/Source:Internal
Signal: internal source
```

A more complicated example involves multiple "source" URNs which do not select a non-default signal and one "priority" URN which can be signaled:

```
Alert-Info: <urn:alert:source:unclassified>,
    <urn:alert:source:internal>,
    <urn:alert:priority:high>

State: Priority/Source
    Process: Source:Other (urn:alert:source:unclassified)
State: Priority/Source:(Other)
    Process: Source:Internal (urn:alert:source:internal)
State: Priority/Source:(Other)
    Process: Priority:High (urn:alert:priority:high)
State: Priority:High/Source:(Other)
Signal: high priority
```

Since the only characteristic of a state that affects the output of the FSM is the state's signal, several groups of states in this FSM can be merged using standard FSM optimization algorithms:

```
states with signal "high priority":
    Priority:High/Source
    Priority:High/Source:(Other)
    Priority:High/Source:(External)
    Priority:High/Source:(Internal)

states with signal "low priority":
    Priority:Low/Source
    Priority:Low/Source:(Other)
    Priority:Low/Source:(External)
    Priority:Low/Source:(Internal)

states with signal "external source":
    Priority/Source:External
    Priority:(High)/Source:External
    Priority:(Low)/Source:External
    Priority:(Other)/Source:External

states with signal "internal source":
    Priority/Source:Internal
    Priority:(High)/Source:Internal
    Priority:(Low)/Source:Internal
    Priority:(Other)/Source:Internal
```

This reduces the FSM to 7 states.

7. Examples 2, 3, and 4 of RFC 7462

Examples 2, 3, and 4 of [RFC7462] are similar to the example in Section 5, but they do not include a signal for the combination "internal source, low priority" to make resolution examples work asymmetrically.

The FSM for this example has the same alphabet as the FSM of Section 5. Most of the states of this FSM are the same as the states of the FSM of Section 5, but the state Source:Internal/Priority:Low is missing because there is no signal for that combination. It is replaced by two states: One state is Source:Internal/Priority:(Low); it records that Source:Internal was specified first (and is to be signaled) and that Priority:Low was specified later (and can not be signaled -- but it still prevents any further "priority" URN from having an effect). The other state is Source:(Internal)/Priority:Low; it records the reverse sequence of events.

The changes in the FSM are:

```
State: Priority:Low/Source
Signal: low priority
Transitions:
    Source:Internal -> Priority:Low/Source:(Internal)
    (other transitions unchanged)

State: Priority:Low/Source:(Internal)
Signal: low priority
Transitions:
    any -> Priority:Low/Source:(Internal)

State: Priority/Source:Internal
Signal: internal source
Transitions:
    Priority:Low -> Priority:(Low)/Source:Internal
    (other transitions unchanged)

State: Priority:(Low)/Source:Internal
Signal: internal source
Transitions:
    any -> Priority:(Low)/Source:Internal
```

An example of processing that involves multiple "source" URNs and one "priority" URN:

```
Alert-Info: <urn:alert:source:internal>,
    <urn:alert:source:unclassified>,
    <urn:alert:priority:high>

State: Priority/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority/Source:Internal
    Process: Source:Other (urn:alert:source:unclassified)
State: Priority/Source:Internal
    Process: Priority:High (urn:alert:priority:high)
State: Priority:High/Source:Internal
Signal: internal source/high priority
```

If the user agent receives

```
Alert-Info: <urn:alert:source:internal>

State: Priority/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority/Source:Internal
Signal: internal source
```

If the user agent receives

```
Alert-Info: <urn:alert:source:external>,
    <urn:alert:priority:low>

State: Priority/Source
    Process: Source:External (urn:alert:source:external)
State: Priority/Source:External
    Process: Priority:Low (urn:alert:priority:low)
State: Priority:Low/Source:External
Signal: external source/low priority
```

Suppose the same user agent receives

```
Alert-Info: <urn:alert:source:internal>,
    <urn:alert:priority:low>
```

Note that there is no signal that corresponds to this combination. In that case, the processing is:

```
State: Priority/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority/Source:Internal
    Process: Priority:Low (urn:alert:priority:low)
State: Priority:(Low)/Source:Internal
Signal: internal source
```

If the order of the URNs is reversed, what is signaled is the the meaning of now-different first URN:

```
Alert-Info: <urn:alert:priority:low>,
    <urn:alert:source:internal>

State: Priority/Source
    Process: Priority:Low (urn:alert:priority:low)
State: Priority:Low/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority:Low/Source:(Internal)
Signal: low priority
```

Notice that the existence of the new states prevents later URNs of a category from overriding earlier URNs of that category, even if the earlier one was not itself signalable and the later one would be signalable in the absence of the earlier one:

```
Alert-Info: <urn:alert:priority:low>,
    <urn:alert:source:internal>,
    <urn:alert:source:external>

State: Priority/Source
    Process: Priority:Low (urn:alert:priority:low)
State: Priority:Low/Source
    Process: Source:Internal (urn:alert:source:internal)
State: Priority:Low/Source:(Internal)
    Process: Source:External (urn:alert:source:external)
State: Priority:Low/Source:(Internal)
Signal: low priority
```

This situation shows the necessity of states whose labels contain parentheses. If the second transition had been to the state Priority:Low/Source (on the basis that there is no proper state Priority:Low/Source:Internal), then the third transition would have been to the state Priority:Low/Source:External, and the signal would have been "external source/low priority".

8. An Example that Subsets Internal Sources

In the the example of Section 4, there are signals for "external source" and "internal source". Let us add to that example a signal for "source internal from a VIP". That last signal expresses the private extension URN urn:alert:source:internal:vip@example, which is a subset of urn:alert:source:internal, which is expressed by the "source internal" signal. There is a total of 3 expressed URNs, one of which is a subset of another:

```
urn:alert:source:internal
urn:alert:source:internal:vip@example
urn:alert:source:external
```

This generates the following alphabet of symbols, which includes two Other symbols for the category source:

```
Source
Source:Internal
Source:Internal:Vip@example
Source:Internal:Other
Source:Other
```

9. An Example of "alert:service" URNs

In this example there are signals for "service forward" (the call has been forwarded) and "source recall callback" (a recall due to a callback). This gives 2 expressed URNs:

urn:alert:service:forward
urn:alert:service:recall:callback

This generates the following alphabet of symbols. Note that there are two "Other" symbols, because the "alert:service" URNs have an additional level of qualification.

Service
Service:Forward
Service:Recall
Service:Recall:Callback
Service:Recall:Other
Service:Other

10. An Example Using Country Codes

In this example, we consider how a UA generates ringback signals when the UA wishes to reproduce the traditional behavior that the caller hears the ringback signals defined by the telephone service in the callee's country, rather than the ringback singals defined by the service in the caller's country. In the Alert-Info header field of the 180 provisional response, we assume that the called UA provides an "alert:country" URN containing the ISO 3166-1 alpha-2 country code.

The UA has a default signal and a "non-country" signal for urn:alert:service:call-waiting. For the example country with code "XA", the UA has a default signal and signals for urn:alert:service:call-waiting and urn:alert:service:forward. For the example country with code "XB", the UA has a default signal and a signal for urn:alert:service:forward. These inconsistencies between the non-country signals and the country signals are chosen to demonstrate the flexibility of the construction method, showing that three systems of signals can be combined correctly even when the systems were established without coordination between them.

The signals are:

Signal	URN(s)
default	(none)
call-waiting	urn:alert:service:call-waiting
XA default	urn:alert:country:xa
XA call-waiting	urn:alert:country:xa, urn:alert:service:call-waiting
XA forward	urn:alert:country:xa, urn:alert:service:forward
XB default	urn:alert:country:xb
XB forward	urn:alert:country:xb, urn:alert:service:forward

The expressed URNs are:

urn:alert:country:xa
urn:alert:country:xb
urn:alert:service:call-waiting
urn:alert:service:forward

The relevant categories of "alert" URNs are only:

country
service

The alphabet of symbols is:

Country
Country:[other]
Country:Xa
Country:Xb
Service
Service:[other]
Service:Call-waiting
Service:Forward

The 15 states are as follows:

```
       State: 0 Country/Service
       Signal: default
       Transitions:
           Country:[other] -> 1 Country:([other])/Service
           Country:Xa -> 5 Country:Xa/Service
           Country:Xb -> 9 Country:Xb/Service
           Service:[other] -> 13 Country/Service:([other])
           Service:Call-waiting -> 14 Country/Service:Call-waiting
           Service:Forward -> 16 Country/Service:(Forward)

    State: 1 Country:([other])/Service
    Signal: default
    Transitions:
        Country:[other] -> 1 Country:([other])/Service
        Country:Xa -> 1 Country:([other])/Service
        Country:Xb -> 1 Country:([other])/Service
        Service:[other] -> 2 Country:([other])/Service:([other])
        Service:Call-waiting -> 3 Country:([other])/Service:Call-waiting
        Service:Forward -> 4 Country:([other])/Service:(Forward)

       State: 2 Country:([other])/Service:([other])
       Signal: default
       Transitions:
           any -> 2 Country:([other])/Service:([other])

       State: 3 Country:([other])/Service:Call-waiting
       Signal: call-waiting
       Transitions:
           any -> 3 Country:([other])/Service:Call-waiting

       State: 4 Country:([other])/Service:(Forward)
       Signal: default
       Transitions:
           any -> 4 Country:([other])/Service:(Forward)

       State: 5 Country:Xa/Service
       Signal: XA default
       Transitions:
           Country:[other] -> 5 Country:Xa/Service
           Country:Xa -> 5 Country:Xa/Service
           Country:Xb -> 5 Country:Xa/Service
           Service:[other] -> 6 Country:Xa/Service:([other])
           Service:Call-waiting -> 7 Country:Xa/Service:Call-waiting
           Service:Forward -> 8 Country:Xa/Service:Forward
```

```
State: 6 Country:Xa/Service:([other])
Signal: XA default
Transitions:
    any -> 6 Country:Xa/Service:([other])

State: 7 Country:Xa/Service:Call-waiting
Signal: XA call-waiting
Transitions:
    any -> 7 Country:Xa/Service:Call-waiting

State: 8 Country:Xa/Service:Forward
Signal: XA forward
Transitions:
    any -> 8 Country:Xa/Service:Forward

State: 9 Country:Xb/Service
Signal: XB default
Transitions:
    Country:[other] -> 9 Country:Xb/Service
    Country:Xa -> 9 Country:Xb/Service
    Country:Xb -> 9 Country:Xb/Service
    Service:[other] -> 10 Country:Xb/Service:([other])
    Service:Call-waiting -> 11 Country:Xb/Service:(Call-waiting)
    Service:Forward -> 12 Country:Xb/Service:Forward

State: 10 Country:Xb/Service:([other])
Signal: XB default
Transitions:
    any -> 10 Country:Xb/Service:([other])

State: 11 Country:Xb/Service:(Call-waiting)
Signal: XB default
Transitions:
    any -> 11 Country:Xb/Service:(Call-waiting)

State: 12 Country:Xb/Service:Forward
Signal: XB forward
Transitions:
    any -> 12 Country:Xb/Service:Forward
```

```
State: 13 Country/Service:([other])
Signal: default
Transitions:
    Country:[other] -> 2 Country:([other])/Service:([other])
    Country:Xa -> 6 Country:Xa/Service:([other])
    Country:Xb -> 10 Country:Xb/Service:([other])
    Service:[other] -> 13 Country/Service:([other])
    Service:Call-waiting -> 13 Country/Service:([other])
    Service:Forward -> 13 Country/Service:([other])

State: 14 Country/Service:Call-waiting
Signal: call-waiting
Transitions:
    Country:[other] -> 3 Country:([other])/Service:Call-waiting
    Country:Xa -> 7 Country:Xa/Service:Call-waiting
    Country:Xb -> 15 Country:(Xb)/Service:Call-waiting
    Service:[other] -> 14 Country/Service:Call-waiting
    Service:Call-waiting -> 14 Country/Service:Call-waiting
    Service:Forward -> 14 Country/Service:Call-waiting

State: 15 Country:(Xb)/Service:Call-waiting
Signal: call-waiting
Transitions:
    any -> 15 Country:(Xb)/Service:Call-waiting

State: 16 Country/Service:(Forward)
Signal: default
Transitions:
    Country:[other] -> 4 Country:([other])/Service:(Forward)
    Country:Xa -> 8 Country:Xa/Service:Forward
    Country:Xb -> 12 Country:Xb/Service:Forward
    Service:[other] -> 16 Country/Service:(Forward)
    Service:Call-waiting -> 16 Country/Service:(Forward)
    Service:Forward -> 16 Country/Service:(Forward)
```

Call-waiting can be signaled in conjunction with country XA but in conjunction with country XB. Thus the ordering of urn:alert:service:call-waiting with urn:alert:country:xa does not matter, but if urn:alert:country:xb appears before urn:alert:service:call-waiting, call-waiting cannot be signaled. On the other hand, if urn:alert:service:call-waiting appears before urn:alert:country:xb, then call-waiting is signaled, but using the non-country sgnal.

```
Alert-Info: urn:alert:country:xa,
        urn:alert:service:call-waiting

State: 0 Country/Service
    Process: Country:Xa (urn:alert:country:xa)
State: 5 Country:Xa/Service
    Process: Service:Call-waiting (urn:alert:service:call-waiting)
State: 7 Country:Xa/Service:Call-waiting
Signal: XA call-waiting

Alert-Info: urn:alert:service:call-waiting,
        urn:alert:country:xa

State: 0 Country/Service
    Process: Service:Call-waiting (urn:alert:service:call-waiting)
State: 14 Country/Service:Call-waiting
    Process: Country:Xa (urn:alert:country:xa)
State: 7 Country:Xa/Service:Call-waiting
Signal: XA call-waiting

Alert-Info: urn:alert:country:xb,
        urn:alert:service:call-waiting

State: 0 Country/Service
    Process: Country:Xb (urn:alert:country:xb)
State: 9 Country:Xb/Service
    Process: Service:Call-waiting (urn:alert:service:call-waiting)
State: 11 Country:Xb/Service:(Call-waiting)
Signal: XB default

Alert-Info: urn:alert:service:call-waiting,
        urn:alert:country:xb

State: 0 Country/Service
    Process: Service:Call-waiting (urn:alert:service:call-waiting)
State: 14 Country/Service:Call-waiting
    Process: Country:Xb (urn:alert:country:xb)
State: 15 Country:(Xb)/Service:Call-waiting
Signal: call-waiting
```

11. Prioritizing Signals

The specifications in [RFC7462] are oriented toward giving the sender of Alert-Info control over which of the "alert" URNs are most important. But in some situations, the user agent may prefer to prioritize expressing one URN category over another regardless of the order their URNs appear in Alert-Info. This section describes how

that can be accommodated within the framework of [RFC7462], and presents an example FSM resulting from that approach.

This example uses the signals of Section 6, viz., "external source", "internal source", "low priority" and "high priority", but this time, we want to signal "high priority" in preference to any other signal that might be applicable.

We accommodate this within the framework of [RFC7462] by assigning the signal "high priority" for each of these combinations of URNs:

```
urn:alert:priority:high
urn:alert:priority:high, urn:alert:source:internal
urn:alert:priority:high, urn:alert:source:external
```

The result is that the "high priority" signal is the "best" signal for any combination of urn:alert:priority:high with "source" URNs.

The intermediate steps of the method produce the same results as before. The signals can express the following URNs:

```
urn:alert:source:external
urn:alert:source:internal
urn:alert:priority:low
urn:alert:priority:high
```

The relevant categories of "alert" URNs are:

```
source
priority
```

The alphabet of symbols is:

```
Source
Source:External
Source:Internal
Source:Other
Priority
Priority:Low
Priority:High
Priority:Other
```

When the FSM is constructed, it is the same as the FSM for Section 6, except that certain states are effectively renamed and merged, because any "source" is defined to be expressed if high priority is expressed:

```
Priority:(High)/Source:External and
Priority:High/Source:(External) become:
    State: Priority:High/Source:External
    Signal: high priority

Priority:(High)/Source:Internal and
Priority:High/Source:(Internal) become:
    State: Priority:High/Source:Internal
    Signal: high priority
```

This reduces the FSM to 18 states. In addition, these two new states, along with a number of other states, can be merged by FSM optimization, since all of them have the signal "high priority" and from them, there are no transitions to states outside this set. The final FSM has 10 states.

12. Dynamic Sets of Signals

This section discusses how to construct FSMs for a user agent that allows variable sets of signals, for example, if the user can configure the use of ringtones. Several approaches can be used:

- Whenever the set of ringtones is changed, re-execute the processes of Section 4.
- Whenever the set of ringtones is changed, rebuild the list of expressed URNs (Section 4.1) and reconstruct the alphabet of symbols (Section 4.2). Then use an algorithm for dynamically constructing states of the FSM as they are needed during Alert-Info processing.
- If the sets of possible URNs expressed by the ringtones is sufficiently limited, the steps of Section 4 can be carried out "generically", and the generic FSM can be specialized for the current ringtone configuration.

The remainder of this section gives an example of the third approach.

For the example, we will use a set of ringtones that express the identify of the caller. To signal this information, a private extension "alert" URN category is used, "caller@example":

```
urn:alert:caller@example:alice@example.com
urn:alert:caller@example:bob@example.com
etc.
```

which we can express by the generic pattern

```
    urn:alert:caller@example:IDENTITY
```

where "IDENTITY" is replaced in succession by the set of caller identities that have their own ringtones to generate the set of expressed URNs.

The alphabet is then:

```
  Caller@example
  Caller@example:IDENTITY
  Caller@example:Other
```

where "IDENTITY" is replaced in succession by the set of caller identities. The "Caller@example:Other" symbol includes all URNs of the category "caller@example" that are not included in any of the other symbols.

The states and transitions of the FSM are:

```
    State: Caller@example (initial state)
    Signal: default
    Transitions:
        Caller@example:IDENTITY -> Caller@example:IDENTITY
        Caller@example:Other -> Caller@example:(Other)

    State: Caller@example:IDENTITY
    Signal: signal for caller IDENTITY
    Transitions:
        any -> Caller@example:IDENTITY

    State: Caller@example:(Other)
    Signal: default
    Transitions:
        any -> Caller@example:(Other)
```

where again, the second state is replicated once for each caller identity that has a ringtone, with "IDENTITY" replaced with the caller identity.

13. Revision History

[Note to RFC Editor: Please remove this entire section upon publication as an RFC.]

13.1. Changes from draft-worley-alert-info-fsm-05 to draft-worley-alert-info-fsm-06

Editorial improvements from independent submission reviewer.

Add note at end of introduction that you can do this by hand in simple cases.

Add the country-code example.

13.2. Changes from draft-worley-alert-info-fsm-04 to draft-worley-alert-info-fsm-05

Editorial improvements.

13.3. Changes from draft-worley-alert-info-fsm-03 to draft-worley-alert-info-fsm-04

Editorial improvements.

13.4. Changes from draft-worley-alert-info-fsm-02 to draft-worley-alert-info-fsm-03

Correct indenting of some lines.

13.5. Changes from draft-worley-alert-info-fsm-01 to draft-worley-alert-info-fsm-02

Recast exposition to feature the implementation of the construction algorithm.

13.6. Changes from draft-worley-alert-info-fsm-00 to draft-worley-alert-info-fsm-01

Reorganized the text, including describing how the FSM states are constructed.

14. References

14.1. Normative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC7462] Liess, L., Ed., Jesske, R., Johnston, A., Worley, D., and P. Kyzivat, "URNs for the Alert-Info Header Field of the Session Initiation Protocol (SIP)", RFC 7462, DOI 10.17487/RFC7462, March 2015, <http://www.rfc-editor.org/info/rfc7462>.

14.2. Informative References

[code] Worley, D., "draft-worley-alert-info-fsm.aux", February 2017, <http://svn.resiprocate.org/rep/ietf-drafts/worley/draft-worley-alert-info-fsm.aux>.

Acknowledgments

Thanks to Paul Kyzivat, whose relentless identification of the weaknesses of earlier versions made the final document much, much better than it would have been, by changing it from the exposition of a concept into a practical tool. Thanks to Rifaat Shekh-Yusef for his thorough review.

Author's Address

Dale R. Worley
Ariadne Internet Services
738 Main St.
Waltham, MA 02451
US

Email: worley@ariadne.com