SALUD
Internet-Draft
Intended status: Standards Track
Expires: October 20, 2016

D. Worley
Ariadne

April 18, 2016

Simpler Algorithms for Processing Alert-Info URNs
draft-worley-alert-info-fsm-00

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 an the best available signal to present to its user to indicate the characteristics of the call. This document describes a method of constructing a finite state machine (FSM) to do this selection. In many situations, the resulting FSM is simpler and faster than previously described selection algorithms. The designer must construct the FSM so that its behavior will satisfy the requirements given in the definition of the "alert" URN namespace.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 20, 2016.

Table of Contents

1. Introduction 2
2. Outline of the Algorithm 3
3. A Very Simple Example 6
4. Example 1 of RFC 7462 8
5. Example with "source" and "priority" URNs 12
6. Examples 2, 3, and 4 of RFC 7462 15
7. Normative References 18
Author's Address 18

1. Introduction

A SIP user agent server determines 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. 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 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 ringing signal. This usage has security problems and is inconvenient to implement in practice. [RFC7462] introduced an alternative practice: The 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 not expected to provide alerting signals that can render more than a small subset of the possible combinations of "alert" URNs, so the user agent is frequently required to select one 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].

Section 12 of [RFC7462] gives one possible algorithm for selecting a signal which satisfies section 11.1. This 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 user agent designers to construct an algorithm for selecting alerting signals based on the Alert-Info header fields in a SIP message. The concept of this algorithm is that the user agent processes the Alert-Info URNs left-to-right using a finite state machine (FSM), and the state of the FSM after processing the URNs determines which signal the user agent will present to the user. If the FSM is correctly constructed by the designer, the constraints of section 11.1 will be satisfied.

2. Outline of the Algorithm

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. The state of the FSM after processing the URNs determines which signal the user agent will present to the user. Each state of the FSM describes the information which has so far been extracted from the URNs.

Note that the values in an Alert-Info header field are allowed to be URIs of any schema.[RFC3261] The processing of URIs that are not "alert" URNs is not considered by this document, nor is it specified by [RFC7462]. But the algorithm designer must consider what to do with such URIs. The simplest choice is to simply 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 all Alert-Info URIs that are not "alert" URNs have been removed.

In order to reduce the infinite set of possible "alert" URNs to a finite "alphabet" of input "letters" which cause the FSM's transitions, the designer must consider which URNs the user agent will understand. For instance, if the user agent has signals to represent

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

the FSM will have transitions for inputs urn:alert:source:external and urn:alert:source:internal.

With each of these inputs, we implicitly include all URNs that extend it with with additional alert-ind-parts, such as urn:alert:source:external:far-out@example. We must do this because the user agent must ignore any trailing alert-ind-parts that it does not understand. To represent this clearly, we will write, e.g., urn:alert:source:external:* to represent the input URN urn:alert:source:external and any extensions of it by further alert-ind-parts. For any FSM state, any of these input URNs will cause the same transition.

Overall, if the user agent has a signal that represents a URN, then that URN and every URN formed from it by adding alert-ind-parts must each be contained in some letter (although in complex cases, they may be distributed across multiple letters).

Now assume that the user agent has a signal for urn:alert:source:external. Thus, it must have urn:alert:source:external:* in the FSM alphabet. Though we are not required to by the rules of [RFC7462] section 11.1, we also want to distinguish all other alert-indications in the category "source", so that earlier "source" URNs prevent any effect from contradictory "source" URNs that appear later. For example, we don't want a call with

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

or

```
Alert-Info: <urn:alert:source:special@example>,
    <urn:alert:source:external>
```

or even

```
Alert-Info: <urn:alert:source:special@example:a:b>,
    <urn:alert:source:external>
```

from being alerted as "external source", even though the user agent has no specific signals for "source:unclassified" or "source:special@example". To achieve this, we add to the FSM alphabet an additional set of URNs

```
urn:alert:source:(other)
```

which consists of all URNs starting with urn:alert:source but whose next alert-ind-part the user agent does not classify specifically.

The result of these considerations is that for any category of "alert" URN that the user agent understands, all possible URNs in that category are elements of exactly one FSM letter.

If the user agent distinguishes both an "alert" URN and a URN that subsets it by adding one or more alert-ind-parts, the alphabet of sets of URNs is more complex. Consider a user agent that can signal not just "internal source" but also "internal source from a VIP", i.e., it has distinct signals for

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

In order to properly partition all "source" URNs, there is one letter for each of these sets:

urn:alert:source:(other)
: all "source" URNs that are not "internal" or "external",

urn:alert:source:external:*
: all "source:external" URNs,

urn:alert:source:internal
: the single URN urn:alert:source:internal,

urn:alert:source:internal:vip@example:*
: all "source:internal:vip@example" URNs, and

urn:alert:source:internal:(other)
: all "source:internal" URNs that have a following alert-ind-part that is not "vip@example".

Each state of the FSM describes the information which has so far been extracted from the URNs. For our convenience, we label each state by one or more URN classes that have so far been processed. The transitions between the states are largely determined by the labels: a transition leads from a start state to the final state that has the label that adds the input URN class to the label of the start state. For a given state and URN class, if there is no final state with an appropriate label, then the transition's final state is same as the start state because the user agent cannot signal the new URN in combination with signaling the earlier URNs.

Each state is also labeled by the signal that the user agent uses to indicate to the user the information recorded by that state. After processing the sequence of URNs in the Alert-Info header field, the user agent alerts using the signal labeling the final state.

3. A Very Simple Example

This section shows a minimal example, where the only Alert-Info URNs that the user agent can signal are:

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

As described in Section 2, we implicitly include with these URNs all URNs with additional alert-ind-parts, such as urn:alert:source:external:far-out@example, and we write, e.g., urn:alert:source:external:* for urn:alert:source:external and any extensions of it by further alert-ind-parts.

We need to classify all other alert-indications in the category "source". To this end, we add to our alphabet:

```
urn:alert:source:(other)
```

which includes all URNs starting with urn:alert:source but whose next alert-ind-part the user agent does not classify specifically.

This gives the following set of URN classes as the input alphabet of the FSM:

```
urn:alert:source:external:*
urn:alert:source:internal:*
urn:alert:source:(other)    [which includes
                             urn:alert:source:unclassified]
```

Each state of the FSM describes the information which has so far been extracted from the URNs. For our convenience, we label each state by one or more URN classes that have so far been processed (and which have some effect). The transitions between the states can be seen from the labels: a transition's final state has the label that adds the URN class which is causing the transition to the label of the start state. If there is no such final state, then the transition's final state is same as the start state. Each state is also labeled by the signal that the user agent uses to indicate to the user the information recorded by that state; after processing the sequence of URNs in the Alert-Info header field, the user agent alerts using the signal labeling the final state.

Because the user agent can signal only one URN, the states of the FSM are:

```
    initial
    source:external
    source:internal
    source:(other)
```

The full FSM is:

```
    State: initial
    Signal: default
    Transitions:
        urn:alert:source:external:* -> source:external
        urn:alert:source:internal:* -> source:internal
        urn:alert:source:(other) -> source:(other)
        other -> initial

    State: source:external
    Signal: external source
    Transitions:
        any -> source:external

    State: source:internal
    Signal: internal source
    Transitions:
        any -> source:internal

    State: source:(other)
    Signal: default
    Transitions:
        any -> source:(other)
```

As you can see, once the FSM receives a URN in category "source", it latches that value to determine the signal.

As an example of processing, if the user agent receives

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

then processing progresses:

```
    State: initial
        Process: 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: initial
        Process: urn:alert:source:external
    State: source:external
        Process: 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: initial
        Process: urn:alert:source:(other)
    State: source:(other)
        Process: 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: initial
        Process: urn:alert:priority:high
    State: initial
        Process: urn:alert:source:internal
    State: source:internal
    Signal: internal source
```

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.

4. Example 1 of RFC 7462

A more complicated example is in section 12.2.1 of [RFC7462], where the user agent can signal "external source", "internal source", "low priority", and "high priority" individually (but not in combination), was well as a default signal.

We want the user agent to understand the following URNs because it has signals to represent them:

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

As before, under each of these, we implicitly include all URNs with additional alert-ind-parts.

To these, we add to the alphabet:

```
urn:alert:source:(other)
urn:alert:priority:(other)
```

The alphabet of the FSM is the union of the alphabets of the two component FSMs:

```
urn:alert:source:external:*
urn:alert:source:internal:*
urn:alert:source:(other)    [which includes
                             urn:alert:source:unclassified]
urn:alert:priority:low:*
urn:alert:priority:high:*
urn:alert:priority:(other)  [which includes
                             urn:alert:priority:normal]
```

In this example, the FSM is:

```
State: initial
Signal: default
Transitions:
    urn:alert:source:external:* -> source:external
    urn:alert:source:internal:* -> source:internal
    urn:alert:source:(other) -> source:(other)
    urn:alert:priority:low:* -> priority:low
    urn:alert:priority:high:* -> priority:high
    urn:alert:priority:(other) -> priority:(other)
    other -> initial
```

Since the user agent is not capable of signaling any additional information when it signals "external source", from state source:external, all URNs transition to itself:

```
    State: source:external
    Signal: external source
    Transitions:
        any -> source:external
```

Similarly:

```
    State: source:internal
    Signal: internal source
    Transitions:
        any -> source:internal
```

The state source:(other) records that an un-signalable "source" URN has been seen. But if a "priority" URN is then processed, the user agent may be able to signal that. So "priority" URNs can cause transitions to other states:

```
    State: source:(other)
    Signal: default
    Transitions:
        urn:alert:source:external:* -> source:(other)
        urn:alert:source:internal:* -> source:(other)
        urn:alert:source:(other) -> source:(other)
        urn:alert:priority:low:* -> priority:low
        urn:alert:priority:high:* -> priority:high
        urn:alert:priority:(other) -> source:(other)/priority:(other)
        other -> source:(other)
```

The "priority" URNs generate a similar set of states:

```
    State: priority:low
    Signal: low priority
    Transitions:
        any -> priority:low

    State: priority:high
    Signal: high priority
    Transitions:
        any -> priority:high
```

```
    State: priority:(other)
    Signal: default
    Transitions:
        urn:alert:source:external:* -> source:external
        urn:alert:source:internal:* -> source:internal
        urn:alert:source:(other) -> source:(other)/priority(other)
        urn:alert:priority:low:* -> priority:(other)
        urn:alert:priority:high:* -> priority:(other)
        urn:alert:priority:(other) -> priority:(other)
        other -> priority:(other)
```

And there is one further state that records that both an un-signalable source has been specified and an un-signalable priority, a state from which no URN can cause a non-default signal:

```
    State: source:(other)/priority:(other)
    Signal: default
    Transitions:
        any -> source:(other)/priority:(other)
```

As an example of processing, if the user agent receives

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

then processing progresses:

```
    State: initial
        Process: urn:alert:source:internal
    State: 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: initial
        Process: urn:alert:source:unclassified
    State: source:(other)
        Process: urn:alert:source:internal
    State: source:(other)
        Process: urn:alert:priority:high
    State: priority:high
    Signal: high priority
```

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

Let us add to the set of signals in the preceding example ones that express combinations like "internal source, high priority". This gives a total of 9 signals. If we have signals to express all combinations, the FSM is the product of two FSMs, each one like the FSM in Section 3: one handling the "source" URNs and one handling the "priority" URNs. The 16 states of the product FSM correspond to pairs of states, one from the simple "source" FSM and one from the simple "priority" FSM.

The following alphabet of URN classes is recognized:

```
urn:alert:source:external:*
urn:alert:source:internal:*
urn:alert:source:(other)    [which includes
                             urn:alert:source:unclassified]
urn:alert:priority:low:*
urn:alert:priority:high:*
urn:alert:priority:(other)  [which includes
                             urn:alert:priority:normal]
```

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

```
State: initial/initial
Signal: default
Transitions:
    urn:alert:source:external:* -> source:external/initial
    urn:alert:source:internal:* -> source:internal/initial
    urn:alert:source:(other) -> source:(other)/initial
    urn:alert:priority:high:* -> initial/priority:high
    urn:alert:priority:low:* -> initial/priority:low
    urn:alert:priority:(other):* -> initial/priority:(other)
    other -> initial/initial

State: source:external/initial
Signal: external source
Transitions:
    urn:alert:priority:high:* -> source:external/priority:high
    urn:alert:priority:low:* -> source:external/priority:low
    urn:alert:priority:(other):* -> source:external/priority:(other)
    other -> source:external/initial
```

```
    State: source:internal/initial
    Signal: internal source
    Transitions:
        urn:alert:priority:high:* -> source:internal/priority:high
        urn:alert:priority:low:* -> source:internal/priority:low
        urn:alert:priority:(other):* -> source:internal/priority:(other)
        other -> source:internal/initial

     State: source:(other)/initial
     Signal: default
     Transitions:
         urn:alert:priority:high:* -> source:(other)/priority:high
         urn:alert:priority:low:* -> source:(other)/priority:low
         urn:alert:priority:(other):* -> source:(other)/priority:(other)
         other -> source:(other)/initial

       State: initial/priority:high
       Signal: high priority
       Transitions:
           urn:alert:source:external:* -> source:external/priority:high
           urn:alert:source:internal:* -> source:internal/priority:high
           urn:alert:source:(other) -> source:(other)/priority:high
           other -> initial/priority:high

       State: source:external/priority:high
       Signal: external source/high priority
       Transitions:
           any -> source:external/priority:high

       State: source:internal/priority:high
       Signal: internal source/high priority
       Transitions:
           any -> source:internal/priority:high

       State: source:(other)/priority:high
       Signal: high priority
       Transitions:
           any -> source:(other)/priority:high

       State: initial/priority:low
       Signal: low priority
       Transitions:
           urn:alert:source:external:* -> source:external/priority:low
           urn:alert:source:internal:* -> source:internal/priority:low
           urn:alert:source:(other) -> source:(other)/priority:low
           other -> initial/priority:low
```

```
      State: source:external/priority:low
      Signal: external source/low priority
      Transitions:
          any -> source:external/priority:low

      State: source:internal/priority:low
      Signal: internal source/low priority
      Transitions:
          any -> source:internal/priority:low

      State: source:(other)/priority:low
      Signal: low priority
      Transitions:
          any -> source:(other)/priority:low

      State: initial/priority:(other)
      Signal: default
      Transitions:
          any -> initial/priority:(other)

      State: source:external/priority:(other)
      Signal: external source
      Transitions:
          any -> source:external/priority:(other)

      State: source:internal/priority:(other)
      Signal: internal source
      Transitions:
          any -> source:internal/priority:(other)

      State: source:(other)/priority:(other)
      Signal: default
      Transitions:
          any -> source:(other)/priority:(other)
```

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: initial/initial
    Process: urn:alert:source:internal
State: source:internal/initial
    Process: urn:alert:source:unclassified
State: source:internal/initial
    Process: urn:alert:priority:high
State: source:internal/priority:high
Signal: internal source/high priority
```

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

Example 2 of [RFC7462] is similar to the example in Section 5, but it does not include a signal for the combination "internal source, low priority" so as to set up some tricky resolution examples.

The FSM for this example has the same alphabet as the FSM of the preceding example. Most of the states of this FSM are the same as the states of the preceding example, 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 we label "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 we label "[source:internal]/priority:low"; it records the reverse sequence of events.

The changes in the FSM are:

```
State: source:internal/initial
Signal: internal source
Transitions:
    urn:alert:priority:low:* -> source:internal/[priority:low]
    (other transitions unchanged)

State: initial/priority:low
Signal: low priority
Transitions:
    urn:alert:source:internal:* -> [source:internal]/priority:low
    (other transitions unchanged)
```

```
    State: source:internal/[priority:low]
    Signal: internal source
    Transitions:
        any -> source:internal/[priority:low]

    State: [source:internal]/priority:low
    Signal: low priority
    Transitions:
        any -> [source:internal]/priority:low
```

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: initial/initial
        Process: urn:alert:source:internal
    State: source:internal/initial
        Process: urn:alert:source:unclassified
    State: source:internal/initial
        Process: urn:alert:priority:high
    State: source:internal/priority:high
    Signal: internal source/high priority
```

If the user agent receives

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

    State: initial/initial
        Process: urn:alert:source:internal
    State: source:internal/initial
    Signal: internal source
```

If the user agent receives

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

    State: initial/initial
        Process: urn:alert:source:external
    State: source:external/initial
        Process: urn:alert:priority:low
    State: source:external/priority:low
    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: initial/initial
        Process: urn:alert:source:internal
    State: source:internal/initial
        Process: urn:alert:priority:low
    State: source:internal/[priority:low]
    Signal: internal source
```

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

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

    State: initial/initial
        Process: urn:alert:priority:low
    State: initial/priority:low
        Process: urn:alert:source:internal
    State: [source:internal]/priority:low
    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:

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

    State: initial/initial
        Process: urn:alert:priority:low
    State: initial/priority:low
        Process: urn:alert:source:internal
    State: [source:internal]/priority:low
        Process: urn:alert:source:external
    State: [source:internal]/priority:low
    Signal: low priority
```

whereas if the second transition had been to the state "initial/ priority:low" (on the basis that there is no proper state "source:internal/priority:low"), then the third transition would have been to the state "source:external/priority:low", and the signal would have been "external source/low priority".

7. 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>.

Author's Address

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

Email: worley@ariadne.com