End System Service Examples

Xiaotao Wu and Henning Schulzrinne
Department of Computer Science
Columbia University
{xiaotaow,hgs}@cs.columbia.edu

December 8, 2004

Abstract

This technical report investigates services suitable for end systems. We look into ITU Q.1211 services, AT&T 5ESS switch services, services defined in CSTA Phase III, and new services integrating other Internet services, such as presence information. We also explore how to use the Language for End System Services (LESS) to program the services.

Contents

1 Introduction

We have defined a service creation language called the Language for End System Services (LESS) [11]. To investigate its usability, we analyze the existing services defined in the PSTN and some new services specifically for Internet telephony systems, and try to use LESS to program these services. For each service, we will first analyze whether we should put the service in a network server or in end systems. For an end system service, we further distinguish whether the service should be a basic end system implementation or a programmable service. We then use LESS to handle programmable services.

Section 2 briefly introduce the Language for End System Services (LESS). Section 3 shows the services in Annex B of ITU-T recommendation Q.1211 [4]. Section 4 shows the services in 5ESS switches [2]. Section 5 shows the services defined in CSTA Phase III [3]. Section 6 shows the new services not in PSTN networks. Section 7 concludes the paper.

2 The Language for End System Services (LESS)

The Language for End System Services (LESS) [11] is designed for programming communication services in end systems and used by end users without programming experience. The goal of the language is to allow end users to program their own communication services with little training in a graphical service creation environment. To achieve the goal, the language must represent a high-level abstraction of communication behaviors. The elements in the language must have semantic meanings. The language has to be simple and safe. We have a technical report [13] detailing how to handle the simplicity and safety in LESS. We briefly introduce LESS and its design rules in this section. If we want to add new elements in LESS for handling the services in this technical report, we must follow the defined rules.

2.1 High level abstraction of LESS

Figure 1 shows the service model of LESS. A call decision making process in LESS consists of three steps: invoking triggers, branching call decisions based on switches, and performing communication actions.

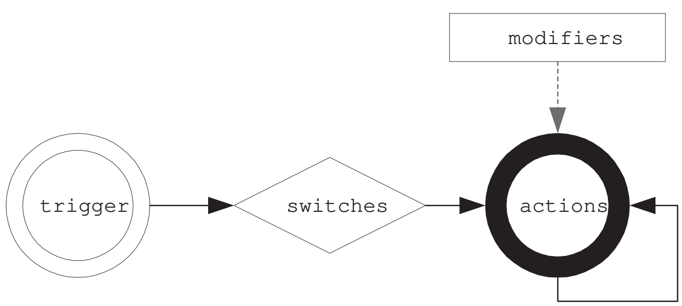

Figure 1: Call decision making process in LESS

A service starts when a trigger invoked. A trigger can be an incoming call, a timer event, or user interactions. New triggers, such as event notification, can be introduced by defining LESS extensions. In a LESS script, switches check the status of the trigger and its context. For example, an `address-switch` may check the caller and the callee's addresses, and based on the addresses to make call decisions. Call decisions are executed by performing communication actions, such as `accept`, `reject`, or `redirect` a call. The modifiers are used to provide action arguments. For example, the `location-modifier` element indicates the URI to `redirect` a call to. Additional actions may get performed based on the results of their previous actions. Multiple actions can be executed simultaneously. The abstraction simulates people's natural thinking for call decision making and is easy for users to understand and learn.

The high-level abstraction implies a tree-like structure for call decision making as shown in Figure 2. The tree-like structure makes LESS easy to analyze and safe for service programming.

Below is the script representing the decision tree in Figure 2.

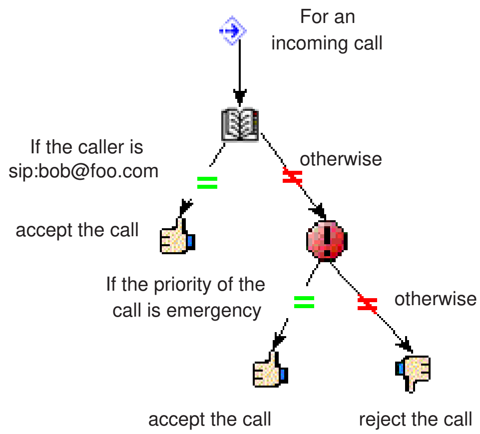

Figure 2: Tree-like structure for call decision making

```
<less>
  <incoming>
    <address-switch field="origin">
      <address is="sip:bob@foo.com">
        <accept/>
      </address>
      <otherwise>
        <priority-switch>
          <priority equal="emergency">
            <accept/>
          </priority>
          <otherwise>
            <reject status="486"/>
          </otherwise>
        </priority-switch>
      </otherwise>
    </address-switch>
  </incoming>
</less>
```

2.2 LESS design rules

To keep the language simple, we only defined four rules for LESS.

Trigger rule: A LESS script gets invoked if and only if one of its triggers get matched. A trigger must be the root of a decision tree and can appear no more than once in a script.

Switch rule: Only switches can branch call decisions. One switch have no more than two branches. A switch must be a child of a trigger or another switch in a decision tree.

Action rule: Only actions can change call status and call context and only actions can be LESS decision tree leaves. There can be subsequent actions after one action. No triggers can follow an action.

Modifier rule: A modifier can only be used as the parent element of actions to enforce the actions.

In this technical report, we should not introduce new rules except defining some complementary rules to clarify the existing rules. We must not introduce any new elements breaking these rules.

3 Q.1211 services

The technical report by Lennox et al [5] illustrated how to use SIP to handle Q.1211 services. In table 1 of that report, it defines what Q.1211 services are suitable for end systems. We only focuses on the services appropriate for end systems and tries to use LESS to program those services.

3.1 Abbreviated dialing (ABD)

"Abbreviated dialing allows the definition of short (e.g., two digit) digit sequences to represent the actual dialing digit sequence for a public or private numbering scheme."

Internet telephony end systems may simply use speed-dial buttons to handle the work. Instead of dialing a digit sequence, a single button-click may trigger a call. End systems may also use LESS service scripts to do the translation from a short sequence to an actual sequence. Below is a LESS script example for the ABD service.

```
<less>
  <outgoing>
    <address-switch field="destination">
      <address is="tel:11; phone-context=local">
        <location url="tel:+1-212-939-7054">
          <call/>
        </location>
      </address>
    </address-switch>
  </outgoing>
</less>
```

3.2 Attendant (ATT)

"This allows VPN users to access an attendant (operator) position within the VPN for providing VPN service information (e.g, VPN numbers) by dialing a special access code. An Internet telephony end system needs only to be configured with an address of an appropriate local operator to translate the special access code to the actual local address of an attendant, or some address which will resolve to that address." For LESS, this is similar to the ABD service (Section 3.1) handling.

3.3 Authentication (AUTC)

"This allows verification that a user is allowed to access certain options in the telephone network. This should be in basic end system implementation."

Comments: 'authenticate' can be an action used by end systems. End systems can perform additional actions based on the result of authentications. An end system can authenticate both caller and callee. Since multiple people may share one phone, callee authentication can ensure only the right person can take a calling call. For example,

```
<less>
  <incoming>
    <UI:authenticate user="{message.destination}">
      <success>
        <!-- If authentication succeeded, accept the call -->
        <accept/>
      </success>
      <fail>
        <!-- If authentication failed, continue alerting -->
        <UI:alert/>
      </fail>
    </UI:authenticate>
  </incoming>
</less>
```

3.4 Authorization code (AUTZ)

"This allows a user (typically in a VPN) to override the restrictions placed on the system from which calls are made. This should be in basic end system implementation." Though we can user LESS scripts to automate the authorization process, it is unsafe to save the codes in LESS scripts.

3.5 Automatic call back (ACB)

"This feature allows the called party to automatically call back the calling party of the last call directed to the called party."

This can be handled by either end systems or network servers. If handled by a network server, the network server should send a SIP REFER request to the calling party to initiate the call back (This is more like traditional Automatic Callback service).

The service involves two steps, one is to log received invitations, the other is to retrieve the invitations and make calls. Below is a LESS service script handling this service.

```
<less>
  <incoming>
    <status-switch>
      <status status="busy">
        <reject>
          <!-- Any action should have a 'next' for subsequent actions -->
          <!-- If one action has a sibling action, the sibling action
               will be executed in parallel with this action -->
          <next>
            <!-- A "callback" queue is defined -->
            <Queue:enqueue queue="callback"/>
          </next>
        </reject>
      </status>
    </status-switch>
  </incoming>
  <Event:notification>
    <address-switch field="origin">
      <address url="{agent.uri}">
        <Event:event-switch>
          <Event:event package="presence" status="open" activity="">
            <!-- If no 'number' parameter provided, the Queue:dequeue action
                 will sequentially retrieve all elements in the queue -->
            <Queue:dequeue queue="callback">
              <success>
                <call/>
              </success>
            </Queue:dequeue>
          </Event:event>
        </Event:event-switch>
      </address>
    </address-switch>
  </Event:notification>
</less>
```

Comments: This requires an end system to store and retrieve calls from a queue. It also requires the end system be able to monitor user's status. Comments: We use `Event:notification` to handle both local status changes and remote event notifications.

3.6 Call distribution (CD)

”This service feature allows the served user to specify the percentage of calls to be distributed among two or more destinations. Other criteria may also apply to the distribution of calls to each destination.”

This is usually done by network servers. If we extend CPL with a switch checking multiple destinations' status, we can handle the service in a proxy server by using CPL.

3.7 Call forwarding (CF)

”This service feature allows a user to have his incoming calls addressed to another number, no matter what the called party line status may be.” A simple LESS script can handle this below:

```
<less>
  <incoming>
    <location url="sip:bob-home@example.com">
      <redirect/>
    </location>
  </incoming>
</less>
```

3.8 Call forwarding on busy/don't answer (CFC)

”This service feature allows the called user to forward particular calls if the called user is busy or does not answer within a specified number of rings.”

This service requires an end system to check its status and make call decisions. The following script forwards calls on busy.

```
<less>
  <incoming>
    <status-switch>
      <status activity="on-the-phone">
        <location url="sip:bob-home@example.com">
          <redirect/>
        </location>
      </status>
    </status-switch>
  </incoming>
</less>
```

Comments: The status-switch is used to check user status

The following script is for call forwarding on no answer.

```
<less>
  <incoming>
    <alert timeout="2000">
      <timeout>
        <location url="sip:bob-home@example.com">
          <redirect/>
        </location>
      </timeout>
    </alert>
  </incoming>
</less>
```

3.9 Call gapping (GAP)

"This feature allows the service provider to restrict the number of calls to a served user to prevent congestion of the network." This service should be implemented in network servers. A similar feature, Call Limiter (LIM) (Section 3.11), will do the similar thing but for end systems.

3.10 Call hold with announcement (CHA)

"The call hold with announcement service feature allows a subscriber to place a call on hold with options to play music or customized announcements to the held party." This service requires user interaction. A LESS script that can enhance the service by customizing the announcements based on addresses is shown below:

```
<less>
  <!-- A toplevel action to handle user input -->
  <!-- For portability, the values for the parameter 'command'
       are pre-defined, each command can only be applied under a
       certain condition. 'hold' can only be used with an
       ongoing call -->
  <UI:command command="hold">
    <address-switch field="destination">
      <address is="sip:bob@example.com">
        <Media:media media="audio" input="music.au" mode="sendonly">
          <Media:mediaupdate/>
        </Media:media>
      </address>
    </address-switch>
  </UI:command>
</less>
```

Comments: There should be a toplevel action to handle user inputs, 'Media:media' modifier to set media attributes for a call, and 'Media:mediaupdate' action to change the media attributes.

3.11 Call limiter (LIM)

"This service feature allows a served user to specify the maximum number of simultaneous calls to a served user's destination. If the destination is busy, the call may be routed to an alternative destination."

Internet telephony end systems can have virtually unlimited lines. However, the resources (CPU, memory, I/O devices) are limited for an end device so the number of simultaneous calls is limited. This can be handled by a LESS script below:

```
<less>
  <incoming>
    <status-switch>
      <status active-calls="2">
        <reject status="busy"/>
      </status>
    </status-switch>
  </incoming>
</less>
```

Comments: We should define an 'active-calls' parameter in 'status' for LESS.

3.12 Call logging (LOG)

"This service feature allows for a record to be prepared each time that a call is received to a specified telephone number." The `<log>` action in LESS can handle this feature.

```
<less>
  <incoming>
    <log/>
  </incoming>
</less>
```

3.13 Call queueing (QUE)

"This service feature allows calls which would otherwise be declared busy to be placed in a queue and connected as soon as the free condition is detected. Upon entering the queue, the caller hears an initial announcement informing the caller the call will be answered when a line is available."

The call queueing service can be handled by an application server in the network. End systems can also handle it. The LESS script below handles the service.

```
<less>
  <incoming>
    <status-switch>
      <!-- For more than one active call, we use 'active-calls="1+" -->
      <status active-calls="1">
        <Media:media media="audio" input="music.au" mode="sendonly">
          <accept>
            <next>
              <Queue:enqueue queue="hold"/>
            </next>
          </accept>
        </Media:media>
      </status>
    </status-switch>
  </incoming>
  <Event:notification>
    <status-switch>
      <status active-calls="0">
        <!-- Retrieve only one call from the hold queue -->
        <Queue:dequeue queue="hold" number="1">
          <success>
            <Media:media media="audio" input="microphone" mode="sendrecv">
              <Media:mediaupdate/>
            </Media:media>
          </success>
        </Queue:dequeue>
      </status>
    </status-switch>
  </Event:notification>
</less>
```

Comments: On event notification handling: When receiving a local event notification, a user agent must update the status it is maintaining first, then invoke LESS engine for service handling. When receiving a presentity's event notification, LESS engine must maintain a temporary status variable, update the status variable to the status in the event notification, and perform event handling based on the updated status. By default, a user agent should update a presentity's status unless its LESS engine explicitly rejects the notification, for example, authentication failed.

Comments: We define a queue named 'hold' to hold all calls on hold.

3.14 Call transfer (TRA)

"The call transfer service feature allows a subscriber to place a call on hold and transfer the call to another location."

In LESS, the `<transfer>` action is used to handle the service. Call transfer is a user triggered service. A LESS script can be triggered by a user input event to perform a transfer.

```
<less>
  <!-- the ’transfer’ command only applies to the current ongoing call -->
  <UI:command command="transfer">
    <!-- Transfer all calls between 09:00 ˜ 17:00 to specialist@foo.com,
         otherwise, to technician@foo.com -->
    <time-switch>
      <time dtstart="20040716T090000Z"
        duration="PT8H" freq="weekly" byday="MO,TU,WE,TH,FR">
        <location uri="specialist@foo.com">
          <transfer/>
        </location>
      </time>
      <otherwise>
        <location uri="technician@foo.com">
          <transfer/>
        </location>
      </otherwise>
    </time-switch>
  </UI:command>
</less>
```

3.15 Call waiting (CW)

”This service feature allows a subscriber to receive a notification that another party is trying to reach his number while he is busy talking to another calling party.”

Since an Internet telephony end system can have multiple lines and a better user interface, such as a bigger LCD display, it can alert users for any new calls. No service scripts required for this service.

3.16 Closed user group (CUG)

”This service feature allows the user to be a member of a set of users who are normally authorized to make and receive calls only within the group.”

This service should be put on network servers, though end systems can handle it by using call screening services. We will introduce call screening services in Section 3.30 and 3.37.

3.17 Consultation calling (COC)

”The consultation calling service feature allows a subscriber to place a call on hold, in order to initiate a new call for consultation.”

For Internet telephony end systems, this service is similar to Call hold with announcements (CHA), which we have discussed in Section 3.10.

3.18 Customer profile management (CPM)

”This service feature allows the subscriber to real-time manage his service profile, i.e. terminating destinations, announcements to be played, call distribution, and so on.”

This is about service script management. There should be a friendly service creation and management user interface for end systems.

3.19 Customer recorded announcement (CRA)

”This service allows a call to be completed to a (customized) terminating announcement instead of a subscriber line. The served user may define different announcements for unsuccessful call completions due to different reasons (e.g. caller outside business hours, all lines are busy).” The switches in LESS can do the customization work. For example, a time-based customization is as below:

```
<less>
  <incoming>
    <!-- Effective from 07/16/2004, if an incoming call is after
         05:00:00 PM Friday, and before 09:00:00 AM Monday, play
         the no_office_hour.au announcement -->
    <time-switch>
      <time dtstart="20040716T170000Z"
        duration="PT16H" freq="weekly" byday="MO,TU,WE,TH,FR">
        <Media:media media="audio" source="no_office_hour.au" mode="sendonly">
          <accept/>
        </Media:media>
      </time>
    </time-switch>
  </incoming>
</less>
```

3.20 Customized ringing (CRG)

”This service feature allows the subscriber to allocate a distinctive ringing to a list of calling parties.”

This service can only be handled in an end system. The script below shows an enhanced version not only based on address but also based on priority of the call to perform customized ringing.

```
<less>
  <incoming>
    <priority-switch>
      <priority equal="emergency">
        <alert priority="emergency"/>
      </priority>
      <otherwise>
        <address-switch field="origin">
          <address is="sip:bob@example.com">
            <alert audio="bob.au"/>
          </address>
          <otherwise>
            <alert/>
          </otherwise>
        </address-switch>
      </otherwise>
    </priority-switch>
  </incoming>
</less>
```

Comments: This service requires LESS to support different alerting priorities, according to call priorities.

3.21 Destination user prompter (DUP)

”This service feature enables to prompt the called party with a specific announcement. Such an announcement may ask the called party enter an extra numbering, e.g. through DTMF, or a voice instruction that can be used by the service logic to continue to process the call.”

Though we can use LESS <Media:media> modifier to easily play an announcement, it is out of LESS's scope to handle DTMF inputs. The DTMF handling falls into VoiceXML's [10] domain. For example, `UI:prompt script="input1.vxml"/>`. More investigation is required on integrating VoiceXML into LESS.

Comments: LESS should abstract input/output operations and consider VoiceXML as a lower level functions to handle input/output

3.22 Follow-me diversion (FMD)

"With this service feature, a user may register for incoming calls to any terminal access." This service should be in network servers.

3.23 Mass calling (MAS)

"This service feature allows processing of huge numbers of incoming calls, generated by broadcasted advertisings or games." This service should be handled by network servers.

3.24 Meet-me conference (MMC)

"This service feature allows the user to reserve a conference resource for making a multi-party call. At a specified date and time, each participant in the conference has to dial a designated number in order to have access to the conference." This is a standard conferencing service. We can use LESS to perform time and caller checking, and authentication.

```
<less>
  <incoming>
    <time-switch>
      <!-- a conference on every Monday from 17:00 to 19:00 -->
      <time dtstart="20040716T170000Z"
        duration="PT2H" freq="weekly" byday="MO">
        <address-switch field="origin">
          <address is="sip:bob@example.com">
            <!-- We need to distinguish between remote user interaction
                 and local user interaction -->
            <RUI:authenticate script="auth.vxml">
              <success>
                <accept/>
              </success>
            </RUI:authenticate>
          </address>
        </address-switch>
      </time>
  </incoming>
</less>
```

Comments: Authentication can be done through external interface, such as a VoiceXML script.

3.25 Multi-way calling (MWC)

"This service feature allows the user to establish multiple, simultaneous telephone calls with other parties."

An Internet telephony end system can initiate as many simultaneous calls as it wishes as long as the network bandwidth and the CPU processing speed permit. A changed version of the service will be to put a limit on the number of simultaneous calls. The changed version can be handled by a LESS script below:

```
<less>
  <outgoing>
    <status-switch>
      <status active-calls="2">
```

```
        <terminate>
          <next>
            <alert message="Only 2 active calls allowed"/>
          </next>
        </terminate>
      </status>
    </status-switch>
  </outgoing>
</less>
```

3.26 Off-net access (OFA)

"This service feature allows a VPN user to access his or her VPN from any non-VPN station in the PSTN by using a personal identification number (PIN)." This service should not be handled by using LESS scripts.

3.27 Off-net calling (ONC)

"This service feature allows the user to call outside the VPN network."

This service is just a standard firewall traversal process, no service scripts required.

3.28 One number (ONE)

"This feature allows a subscriber with two or more terminating lines in any number of locations to have a single telephone number. This allows businesses to advertise just one telephone number throughout their market area and to maintain their operations in different locations to maximize efficiency. The subscriber can specify which calls are to be terminated on which terminating lines based on the area the calls originate."

This is a standard SIP proxy service, not handled by end systems.

3.29 Origin dependent routing (ODR)

"This service feature enables the subscriber to accept or reject a call, and in case of acceptance, to route this call, according to the calling party geographical location. This service feature allows the served user to specify the destination installations according to the geographical area from which the call was originated."

This service requires `<LOC:where-switch>` to handle location-based call routing. A service script below can handle the service. For end systems, we use `<redirect/>` action to route calls. A proxy server is a more appropriate place to host the service, and use `<proxy/>` action to route calls.

```
<less>
  <incoming>
    <address-switch field="origin">
      <address is="sip:bob@example.com">
        <LOC:where-switch type="civil" principle="sip:bob@example.com">
          <LOC:where country="USA" A1="New York"
            A3="New York" A6="West 120th" HNO="450" LOC="Room 563">
            <location url="sip:bob-office@example.com">
              <redirect/>
            </location>
          </LOC:where>
          <otherwise>
            <location url="sip:bob-mobile@example.com">
              <redirect/>
            </location>
          </otherwise>
        </LOC:where-switch>
      </address>
```

```
    </address-switch>
  </incoming>
</less>
```

3.30 Originating call screening (OCS)

”This service feature allows the served user to bar calls from certain areas based on the District code of the area from which the call is originated.” This service can be handled by using `address-switch`.

```
<less>
  <incoming>
    <address-switch field="destination" subfield="host">
      <address is="example.com">
        <reject status="reject"/>
      </address>
    </address-switch>
  </incoming>
</less>
```

3.31 Originating user prompter (OUP)

”This service feature allows a served user to provide an announcement which will request the caller to enter a digit or series of digits via a DTMF phone or generator. The collected digits will provide additional information that can be used for direct routing or as a security check during call processing.”

This service can be in a network server or a user agent. To enable a user agent to support this service, it requires the user agent to handle the interaction with remote users.

Comments: This requires integrating VoiceXML into LESS.

3.32 Personal numbering (PN)

”This service feature supports a UPT number that uniquely identifies each UPT user and is used by the caller to reach that UPT user.” Reaching a user through a personal address is accomplished simply by a proxy or redirection server which locates the user.

3.33 Premium charging (PRMC)

”This service feature allows for the pay back of part of the cost of a call to the called party.” This service is out of the scope of LESS.

3.34 Private numbering plan (PNP)

”This service feature allows the subscriber to maintain a numbering plan within his private network, which is separate from the public numbering plan.” This service should usually be put in outbound proxy servers. If we provide such service in end systems, it would be similar to the Abbreviated dialing (ABD) service we introduced in Section 3.1.

3.35 Reverse charging (REVC)

”This service feature allows the service subscriber (e.g. freephone) to accept to receive calls at its expense and be charged for the entire cost of the call.”

This service and the next service ”Split charging” require a detailed VoIP charging schema and have the charging information available to users. Currently, LESS cannot handle this kind of service. However, if call signalling messages can provide charging information, LESS scripts can perform call decisions, such as automatically rejecting a reverse charging calls unless it is from certain users, based on the charging information.

3.36 Split charging (SPLC)

"This service feature allows for the separation of charges for a specific call, the calling and called party each being charged for one part of the call." This service is similar to the previous service, LESS cannot handle the service.

3.37 Terminating call screening (TCS)

"This service feature allows the user to screen calls based on the terminating telephone number dialed." Below is a LESS script example handling this service.

```
<less>
  <incoming>
    <address-switch field="destination" subfield="host">
      <address is="example.com">
        <reject status="reject"/>
      </address>
    </address-switch>
  </incoming>
</less>
```

3.38 Time dependent routing (TDR)

"This services feature allows the served user to apply different call treatments based on the time of day, day of week, day of year, holiday, etc." Usually, routing services should be put in network servers, but we can perform similar services for time dependent redirecting. This can be handled by LESS `time-switch`.

```
<less>
  <incoming>
    <time-switch>
      <time dtstart="20040716T170000Z"
        duration="PT8H" freq="weekly" byday="MO">
        <location url="sip:bob@voicemail.com>
          <redirect/>
        </location>
      </time>
    </time-switch>
  </incoming>
</less>
```

3.39 Summary

This section summarizes new elements we need to define in LESS for Q.1211 service handling.

`authenticate` action: The action should be able to authenticate both callers and callees. There is one parameter 'user' indicating the party get authenticated. The output of the action can be `success`, and `failed`.

`Queue` package: This package contains two actions, `enqueue` and `dequeue`, for queue handling. There are two parameters for the actions. `queue` parameter provides the name of a queue. We defined some commonly used queuenames, such as `callback`, and `hold`. Users can also define their own queue names for queueing actions. `number` parameter in `dequeue` defines how many queue elements should be retrieved, if not provided, all queue elements get retrieved sequentially.

`Event:notification` handling: We use it to handle both location status changes and remote event notifications. When receiving a local event notification, a user agent must update the status it is maintaining first, then invoke LESS engine for service handling. When receiving a presentity's event notification, LESS engine must maintain a temporary status variable, set the value of the status variable as the status value in the event notification, and perform event handling based on the updated status. By default, a user agent should always change a presentity's status when receiving an event notification, unless its LESS engine explicitly rejects the notification, for example, authentication failed.

`status-switch`: We use the switch to check a user agent's own status, such as online/offline, activities, active-calls, and user's mood.

`UI` package: User interactions trigger several different topleval actions, such as `<UI:command command="call">`, `<UI:command command="hold">`, and `<UI:command command="transfer">`. From signalling protocol point of view, these actions all send outgoing signalling messages. We distinguish different purposes of the outgoing messages, for example, `<UI:command command="call">` is only for regular outgoing call initiation. `<UI:command command="hold">` represents holding the current call, `<UI:command command="transferring">` represents transferring the current call. End systems differ from network servers for outgoing message handling. There can also be user defined command names. In CPL [6], `outgoing` top-level action is usually used in an outbound proxy to proxy outgoing messages. The proxy server does not generate outgoing messages by itself, so the `outgoing` top-level action is in fact triggered by incoming messages from UAs in the server's correspondent domain. However, in end systems, an end system generates outgoing messages by itself, outgoing top-level actions are usually triggered by user interactions, for example, a user presses the 'hold' button will trigger the `<UI:command command="hold">` top-level action, a user presses the 'call' button will trigger the `<UI:command command="call">` top-level action. We replace `outgoing` as `<UI:command command="call">` as a user interaction event so we can better handle the `call` action in LESS scripts. `call` action itself will not trigger `<UI:command command="call">` top-level action. The same for `hold` and `transfer` actions, they will not trigger `<UI:command command="hold">` and `<UI:command command="transfer">` top-level actions, accordingly. Making this clear can help to avoid unexpected loops.

`Media` package: We define the `Media:media` modifier to specify the media content of a call. We define the `Media:mediaupdate` action to modify the media content of a call.

`alert` action: We define four parameters for the `alert` action. `priority` parameter defines alerting priority, it can be 'emergency', 'urgent', 'normal', 'non-urgent'. Users can also specify `audio` parameter to play an audio file, `message` to display a text message, and `style` for playing sound, viberating, or flashing. `<alert>` can also have a `timeout` parameter and a `timeout` output. If within the time of `timeout`, a user did not perform any action to stop alerting, the actions in `timeout` output get executed.

`next` output: We should allow multiple actions performed sequentially or in parallel. We use `next` for sequentially performed actions. If we put multiple actions as sibling to each other, these actions should be executed in parallel.

4 5ESS services

In this section, we investigate the services introduced in "AT&T 5ESS Switch The Premier Solution, Feature Handbook" [2]. 5ESS services including the following categories: BRCS (Business and Residence Custom Services) features, ISDN features (message and MBKS), Public Service features (emergency), Defense Switched Network Features, Toll and Tandem features, Interoffice Signaling and Control features, Operator Service Position System features, OA&M features. Among these categories, we are most interested in the BRCS features because many of BRCS features can be handled in end systems. For ISDN features, we will discuss the Message Services and the MBKS (Multibutton Key Telephone System). For Public Service features, we will discuss the group alerting service. We will not discuss the other service categories. We only list the services suitable for end systems.

4.1 Business and residence custom services (BRCS)

4.1.1 Attendant call transfer (also known as Call splitting)

"The service allows the attendant to flash and dial a code before dialing the third leg of a 3-way call. This inhibits the automatic connecting of all the parties to allow private consultation between the attendant and the third leg of the call. If the code is not dialed, the automatic 3-way is formed."

Since an Internet telephony end system usually can have multiple lines, it is easy to put one line on hold and initiate another call. However, to enable this service, an end system should be able to merge an active call and a held call into a 3-way call.

This service is not a programmable service, a user will manually use a second line to initiate the second call, and manually instruct his user agent to merge two calls.

Comments: In some situations, for example, automatic 3-way calling, we do need to have an action `merge` *to merge multiple calls.*

4.1.2 Attendant camp-on

"This service allows incoming calls that the attendant attempts to complete to a busy station to be held waiting until the busy station becomes idle. The busy station receives a tone (indication of camp-on) each time the attendant attempts a completion. The call being transferred receives audible ringing, special tone followed by audible ringing, or optional silence, while waiting for the busy station to answer the call." The following script can handle the service.

```
<less>
  <incoming>
    <status-switch>
      <!-- We use 1+ to represent 1 or more active-calls -->
      <status active-calls="1+">
        <Media:media source="wait.au" mode="sendonly">
          <accept>
            <next>
              <Queue:enqueue queue="hold"/>
            </next>
          </accept>
        </Media:media>
      </status>
    </status-switch>
  </incoming>
  <Event:notification>
    <status-switch>
      <status active-calls="0">
        <Queue:dequeue queue="hold" number="1">
          <success>
            <Media:media media="audio" input="microphone" mode="sendrecv">
              <Media:mediaupdate/>
            </Media:media>
          </success>
        </Queue:dequeue>
      </status>
    </status-switch>
  </Event:notification>
</less>
```

4.1.3 Attendant conference

"This service enables an attendant to initiate a conference call involving up to six parties (including attendant). The selection of a special conference attendant can be done from any station within the same customer group by dialing a particular access code."

There are two ways to handle conferencing services in an end system, one is to have all conference participants to connect to a conference server (by REFER or third-party call control), the other is to use end system mixing. The first way is more scalable, but requires an external conference server. The second way requires an end system supporting mixing. The following scripts show both ways for a time-based conference initiation.

```
<less>
```

```
  <!-- Use SIP REFER -->
  <timer dtstart="20040716T170000Z"
    duration="PT2H" freq="weekly" byday="MO">
    <lookup source="sip:bob@example.com
            sip:tom@example.com sip:alice@abc.com">
      <success>
        <call>
          <success>
            <location url="sip:conf@example.com">
              <transfer/>
            </location>
          </success>
        </call>
      </success>
    </lookup>
  </timer>
</less>
```

The script will automatically send a call invitation to each URL in the list, once a call established, the script will transfer the call to the conference server at `sip:conf@example.com`.

Comments: If during the process, a call comes in and get accepted, the `<transfer/>` action should not be applied to the incoming active call because it is defined as a subsequent action of the `<call/>` action and should only handle the successful call of the `<call/>` action.

Comments: We allow a URL list to be in the `source` parameter of a `<lookup>` action. It looks cumbersome. An alternative way is to allow users to define constant variables in a script. For example, we can introduce a `<define>` tag and have
`<define name="IRT group" values="sip:bob@ex.com sip:tom@ex.com"/>`.
There are two drawbacks for this solution. First, we have a introduce a new rule for variable definition and how to use user-defined variables. Second, when performing multiple script merging, naming conflict may confuse users. Thus, we decide not to allow user-defined constant variables. However, a service creation environment can help users to maintain a global defined name-value mapping.

```
<less>
  <!-- Use local mixing -->
  <timer dtstart="20040716T170000Z"
    duration="PT2H" freq="weekly" byday="MO">
    <sub ref="IRTdef"/>
    <lookup source="sip:bob@ex.com sip:tom@ex.com">
      <success>
        <call subject="Group meeting">
          <success>
            <merge subject="Group meeting"/>
          </success>
        </call>
      </success>
    </lookup>
  </timer>
</less>
```

Comments: `merge` action is required to merge a call to an existing active call. We need to define how to perform the `merge` action. If the parameter `subject` is not defined, the `merge` action will merge all active calls into one multi-party call. If `subject` parameter is provided, all calls with the same subject will get merged.

4.1.4 Authorization code

”The codes allow the station user to input an assigned code to change the restrictions associated with the originating station to those associated with the assigned authorization code. Thus, unauthorized use of facilities is avoid.”

```
<less>
  <outgoing>
    <address-switch field="destination">
      <address is="sip:conf123@example.com">
        <call>
          <success>
            <Msg:sendmsg type="dtmf" digits="123456"/>
          </success>
        </call>
      </address>
    </address-switch>
  </outgoing>
</less>
```

Comments: This service requires "Msg:sendmsg" action can send different type of message out, message type can be "message", "dtmf", "audio", "image", "file".

Comments: For an `<outgoing>` toplevel action, the `<call/>` action is in fact continue the signalling process. The script requires the support of the `status` parameter in the `<failed>` output of a `<call/>` action.

4.1.5 Automatic recall

"This service lets the customer automatically call the LOCDN (last outgoing call directory number) currently associated with the customer's station when both stations become idle. It is different from Automaticall Callback, which automatically places a call on LICDN (last incoming call directory number)." The service script below shows how to program the service in LESS.

```
<less>
  <outgoing>
    <call>
      <failed status="486,603">
        <Queue:enqueue queue="callback">
          <next>
            <Event:subscribe/>
          </next>
        </Queue:enqueue>
      </failed>
    </call>
  </outgoing>
  <Event:notification>
    <Event:event-switch>
      <Event:event activity="normal">
        <!-- User {message.origin} to represent the sender of the notification-->
        <Queue:dequeue queue="callback" match="{message.origin}">
          <success>
            <call/>
          </success>
        </Queue:dequeue>
      </Event:event>
    </Event:event-switch>
  </Event:notification>
</less>
```

Comments: This service requires LESS to handle the result of an outgoing call and allow multiple actions in a decision branch. The LESS script use a queue to connect two kinds of toplevel action handling.

Comments: The ORIGIN variable represents the sender of the signalling message that triggers the `Event:notification` toplevel action. There is no user-defined variables in LESS, however, LESS has several pre-defined variables to represent toplevel action parameters and local status, and has constant variables to represent user groups.

4.1.6 Call forwarding busy line (CFBL)

"This service permits calls attempting to terminate to a busy line to be redirected to another customer-specified line." The following script can handle the service.

```
<less>
  <incoming>
    <status-switch>
      <status active-calls="1+">
        <location url="sip:phone2@example.com">
          <redirect/>
        </location>
      </status>
    </status-switch>
  </incoming>
</less>
```

4.1.7 Call forwarding busy line–incoming only

"This service provides that only incoming DID calls are forwarded to the specified business group line on busy. Intragroup call attempts and attempts from private facilities to terminate to the busy line receive busy treatment." This is in fact address-based call forwarding services.

```
<less>
  <incoming>
    <status-switch>
      <status active-calls="1+">
        <address-switch field="origin">
          <address isnot="sip:*@example.com">
            <location url="sip:phone2@example.com">
              <redirect/>
            </location>
          </address>
        </address-switch>
      </status>
    </status-switch>
  </incoming>
</less>
```

4.1.8 Call forwarding–don't answer

"This service forwards incoming calls to a station when the called station is not answered after a customer-specified number of ringing cycles." The script below can handle the service.

```
<less>
  <incoming>
    <alert timeout="2000">
      <timeout>
        <location url="sip:phone2@example.com">
          <redirect/>
        </location>
      </timeout>
    </alert>
  </incoming>
</less>
```

Comments: This service requires a timeout parameter in 'alert' action

4.1.9 Unconditional call forwarding

"This service forwards all incoming calls to another telephone." We can simply handle the service by using `<redirect/>` action.

```
<less>
  <incoming>
    <location url="sip:bob@room123.example.com">
      <redirect/>
    </location>
  </incoming>
</less>
```

4.1.10 Call hold

"This service allows a station user to 'hold' a call in progress by flashing and then dialing the call hold code. This frees the line for originating another call, answering a waiting call, or returning to a held call." In LESS, we use the action `<Media:mediaupdate/>` to change the media source for "hold" and "unhold" service.

```
  ...
  <!-- hold -->
  <Media:media media="audio" input="music.au" mode="sendonly">
    <Media:mediaupdate/>
  </Media:media>
  ...
  <!-- unhold -->
  <Media:media media="audio" input="microphone" mode="sendrecv">
    <Media:mediaupdate/>
  </Media:media>
```

4.1.11 Call transfer–individual–all calls

"This service allows a station user to transfer any established call to another station within or outside the PBX or business group without the assistance of the attendant. This is accomplished by flashing while on a stable 2-party call, dialing the desired party, and hanging up the telephone." The `<transfer/>` action can handle this service.

```
<less>
  <incoming>
    <media media="audio" input="transfer.au" mode="sendonly">
      <accept>
        <next>
          <location url="sip:bob@room123.example.com">
            <transfer>
              <failed>
                <media media="audio" input="failed.au" mode="sendonly">
                  <mediaupdate/>
                </media>
              </failed>
            </transfer>
          </location>
        </next>
      </accept>
    </media>
  </incoming>
</less>
```

4.1.12 Call waiting and cancel call waiting

Since an Internet telephony end system usually have multiple lines, users do not have to use flash button to switch calls. However, users should be able to control how many active calls an end device can handle. The following script can help to configure the maximum number of calls.

```
<less>
  <incoming>
    <status-switch>
      <status active-calls="3+">
        <reject reason="busy"/>
      </status>
    </status-switch>
  </incoming>
</less>
```

Comments: We need to carefully define what is an active-call. An instant messaging session should not be considered as an active-call. Instant messaging is handled by the `Msg:messaging` and `<UI:command command="sendmsg"` toplevel actions.

4.1.13 Circle hunting

"This service allows all lines in a multiline hunt group to be tested for busy, regardless of the point of entry into the group. When a call is made to a line in an MLHG, a regular hunt is performed starting at the terminal associated with the dialed number. It continues to the last terminal in the MLHG, then proceeds to the first terminal in the group and continues to hunt sequentially through the remaining lines in the group. Busy tone is returnedif the called terminal is reached without finding one that is idle." This service is more suitable for a proxy server to handle. However, we can also handle it in an end system by using `<transfer/>` action.

```
<less>
  <incoming>
    <media type="audio" input="wait.au" mode="sendonly">
      <accept>
        <next>
          <lookup source="sip:bob@ex.com sip:tom@ex.com"
                  order="sequential">
            <success>
              <transfer>
                <succeed>
                  <stop/>
                </succeed>
              </transfer>
            </success>
          </lookup>
        </next>
      </accept>
    </media>
  </incoming>
</less>
```

Comments: We should define a parameter `order` to have the addresses of a `<lookup>` result to be handled sequentially. The `<stop>` action is to stop script processing.

4.1.14 Conference calling

"The service allows a station user to establish a conference call involving up to five other parties without attendant assistance." This service is almost the same as the attendant conferencing service we discussed before, it requires an end system to support mixing, and to merge an incoming call to an existing conference call. We use `<merge/>` action to handle the merge.

4.1.15 Customer-changeable speed calling

”This service allows subscribers to assign their own Speed Calling codes directly and immediately from their own telephone by dialing a change Speed Calling list access code, an abbreviated code and a new telephone number. It is available for 1- and or 2-digit Speed Calling list owners.” In an end system, speed dialing modification should be done by editing service scripts from a local GUI. Below is a script for speed dialing handling.

```
<less>
  <outgoing>
    <address-switch field="destination" subfield="user">
      <address is="1">
        <location url="sip:bob@example.com">
          <call/>
        </location>
      </address>
    </address-switch>
  </outgoing>
</less>
```

4.1.16 Direct connect

”This service automatically places a call to a preselected called number when a station goes off-hook. This feature can be used for introffice and interoffice calls and does not affect termination to a line.” This service requires a specially configured UA. The service should not be handled by a service script.

4.1.17 Distinctive ringing

This service applies a distinctive ringing to determine the source of an incoming call. The following script handles the service.

```
<less>
  <incoming>
    <address-switch field="origin">
      <address is="sip:bob@example.com">
        <alert audio="bob.au"/>
      </address>
    </address-switch>
  </incoming>
</less>
```

4.1.18 Four- and eight-party

”This service provides POTS for up to four or eight customers sharing the same line. It provides fully selective ringing or semiselective ringing for up to four customers and semiselective ringing for five to eight customers.” The following script plays distinctive ringing for different user sharing the same phone.

```
<less>
  <incoming>
    <address-switch field="destination">
      <address is="sip:bob@example.com">
        <alert audio="bob.au"/>
      </address>
      <otherwise>
```

```
        <address-switch field="destination">
          <address is="sip:tom@example.com">
            <alert audio="tom.au"/>
          </address>
          <otherwise>
            <address-switch field="destination">
              <address is="sip:mary@example.com">
                <alert audio="mary.au"/>
              </address>
            </address-switch>
          </otherwise>
        </address-switch>
      </otherwise>
    </address-switch>
  </incoming>
</less>
```

4.1.19 Recorded telephone dictation

"This service permits access to and control of customer-owned dictating equipment from a station in the customer group." This service requires an end system to be able to record and playback a phone call.

```
<less>
  <UI:command command="accept">
    <address-switch field="origin">
      <address is="sip:bob@example.com">
        <log record="true"/>
      </address>
    </address-switch>
  </UI:command>
</less>
```

Comments: Accepting an incoming call should be considered as a user interaction event and should get handled. We should enhance the `<log>` *action to handle recording instead of creating a new action* `<record>`*. Enhancing* `<log>` *can easily associate a call log to a recorded audio file.*

4.1.20 Time-of-Day features

"This kind of features can perform automatic actions based on time-of-day. For example, Slumber Service (also known as Do Not Disturb) can temporarily prohibit an individual customer station or a functional group of individual stations from receiving calls. It is used in hospitals to restrict incoming calls to patients during the night or any designated period." We can use LESS `time-switch` to handle the service.

```
<less>
  <incoming>
    <time-switch>
      <time dtstart="20040716T230000Z"
        duration="PT8H" freq="daily">
        <reject reason="Do not disturb"/>
      </time>
    </time-switch>
  </incoming>
</less>
```

4.2 Integrated services digital network features

4.2.1 Message services

ISDN Message Services provides centralized and personalized call coverage of message answering capabilities. For Basic Message Service, calls to a message service client are redirected via one of the following features.

Call forward – busy line

Call forward – don't answer

Call forward – variable

To provide the personalized call coverage, message service attendants can be equipped with special ISDN station sets that display call information on calls directed to the message service center. The following information is available for display: Call type, Originating party DN, Calling party DN. This service requires an end system can identify a message service, e.g., a voicemail server. The following script forward a call to a voicemail server when the user is busy.

```
<less>
  <incoming>
    <status-switch>
      <status active-calls="2+">
        <location url="sip:bob@voicemail.example.com">
          <redirect/>
        </location>
      </status>
    </status-switch>
  </incoming>
</less>
```

There can be additional features for message services, for example, message waiting indicator, which can inform a client that he/she has message(s) waiting. The Message Waiting Indication Event Package for SIP [7] can provide required message information. An end system should be able to handle the event for the message waiting indicator service.

```
<less>
  <Event:notification>
    <Event:event-switch>
      <Event:event package="message-summary" message-waiting="yes">
        <alert message="There are new messages for you" audio="mwi.au"/>
      </Event:event>
    </Event:event-switch>
  </Event:notification>
</less>
```

4.2.2 Multibutton key telephone system (MBKS)

"Feature Function Buttons on the MBKS set can be assigned to activate certain features. For example, a user can press the function button assigned to Automatic Callback on Busy when a busy number is dialed." We can bind a script to a specific button for this kind of services.

4.3 Public service features

"This part describes features designed to be used directly by Public Service officials (police, fire) or by telephone company personnel in response to requests from these officials." There are four services defined in 5ESS as public service features, namely "Basic Emergency Service (911)", "Call Tracing", "Emergency Ringback", and "Group Alerting". Emergency services are usually not programmable. Call tracing should be handled by network routers. We discuss the "Group Alerting" service below.

4.3.1 Group alerting

”This service permits the controller (or alerting party), upon dialing a code, to signal a preselected number of telephones simultaneously. One-way communication is established from the controller to all members of the group that responded to the signal.” The script below can handle the service.

```
<less>
  <outgoing>
    <!-- define name="Alerting group"
         values="sip:security@abc.com sip:fire@abc.com" -->
    <address-switch field="destination">
      <address is="sip:alert@example.com">
        <lookup source="sip:security@abc.com sip:fire@abc.com">
          <media mode="sendonly">
            <call subject="Alert">
              <success>
                <merge subject="Alert"/>
              </success>
            </call>
          </media>
        </lookup>
      </address>
    </address-switch>
  </outgoing>
</less>
```

4.4 Summary

`<merge>` action: We define `<merge>` action to merge multiple calls into a mixer in an end system so to create an end system based conference. By default, `<merge>` action will merge the call in its toplevel action to the current existing active call. If the `subject` parameter specified, the `<merge>` action will merge calls with the same subject.

Use a URL list as `source` of `<lookup>`: We allow a URL list in the `source` parameter of a `<lookup>` action. It looks cumbersome, however, it can avoid introducing user-defined variables so no new rules required and no potential naming conflicts.

Sequentially handle the result of `<lookup>`: By default, the URLs of the result of `<lookup>` are handled in parallel. To allow sequential handling, we add a new parameter `order`. `order="sequential"` makes the action in `<lookup>` use the URLs of the `<lookup>` result sequentially.

Variables: LESS does not have user-defined variables, however, it allows several pre-defined variables to retrieve system information, user information, toplevel action information, and response information. Below is some examples,

System information: For example, {`system.bandwidth`} for the bandwidth of the system.

User information: For example, {`user.activity`} for script owner’s current activity; {`user.mood`} for script owner’s current mood.

Agent information: For example, {`agent.numcalls`} represents the number of existing calls.

Message information: For example, {`message.origin`} for the originator of a toplevel action’s signalling message.

In may cases, system information and user information variables are used in a descriptive message, such as the `reason` parameter of a reject message. We use `status-switch` to handle call decision based on system and user information.

Mid-session action handling: The most important thing for mid-session actions is to define which session to handle. The rule is, if the mid-session action has a parent action (the mid-session action is a subsequent action of the parent action), and the parent action has an established session, the mid-session action handles its parent action's session. Otherwise, the mid-session action handles its toplevel action's session if the session has been established, otherwise, the mid-session action handles the current ongoing session.

Enhancing `<log>` for recording: We should enhance the `<log>` action to handle recording instead of creating a new action `<record>`. Enhancing `<log>` can easily associate a call log to a recorded audio file.

`<stop>` action to stop executing a script: This is useful when sequentially handling `<lookup>` results, or handling a repeated `<timer>` topleval action.

Msg:sendmsg action: The `Msg:sendmsg` action can send different type of message out, message type can be "message", "dtmf", "audio", "image", "file".

Continue the action triggered by user interaction: User interactions in fact trigger actions. For example, `<outgoing>` action in fact trigger a `<call>` action. If we use a `<call>` action inside an `<outgoing>` without providing any modifiers or parameters, the `<call>` action will continue the action triggered by the `<outgoing>`. We can perform additional actions based on the output of the `<call>` action.

Carefully define active-call: Many services are based on the availability of devices. We use the number of active calls to define a device's availability. In LESS, an active call specifically refers to an audio-enabled ongoing session.

5 Services defined in CSTA Phase III

In this section, we investigate the services in ECMA standard ECMA-269, "Services for Computer Supported Telecommunications Applications (CSTA) Phase III" [3]. The call model of CSTA services is to use protocol messages to control end systems. Many CSTA services are in fact simply actions in LESS. For example, `Accept Call` is a CSTA control service, but just an action, `<accept/>`, in LESS.

CSTA Phase III categorizes services into System Services, Monitoring Services, Snapshot Services, Call Control Services and Events, Call Associated Features, Media Attachment Services and Events, Routing Services, Physical Device Features, Logical Device Features, Device Maintenance Events, I/O Services, Data Collection Services, Voice Services and Events, and Call Detail Record (CDR) Services. Among these services, System Services are used to handle the relationship between switch functions and CSTA service functions and are out of the scope of LESS, Monitoring Services are like to define the DPs (Detection Points) in Intelligent Network (IN) and are also out of the scope of LESS. Snapshot Services are used to send device information to switch functions, not suitable for LESS. The Routing Services are not end system services and are also not suitable for LESS. The Device Maintenance Events, I/O Services, and Data Collection Services are too low-level for end user oriented service programming, thus they are not suitable for LESS. We will discuss the other services below.

5.1 Call control services and events

5.1.1 Accept call

This service can be mapped to `<alert/>` action in LESS.

5.1.2 Alternate call

This service places an existing active call on hold and then retrieve a previously held call. This service in fact involves several actions. We can use the `<Media:mediaupdate/>` action to put a call held/unheld and put held calls into a queue by `enqueue` action.

5.1.3 Answer call

This service can be mapped to `<accept/>` action in LESS.

5.1.4 Call back call-related

The Call Back Call-Related service allows a computing function to request that the calling device retry the call to the called device when the called device is in an appropriate state to accept the call. This is the same as the ACB service described in Section 4.1.5.

5.1.5 Call back message call-related

The Call Back Message Call-Related service allows a computing function to request that the switching function leave a pre-defined message requesting that the called device call the calling device. For example, the called device may have been busy when called.

```
<less>
  <incoming>
    <status-switch>
      <status activity="on-the-phone">
        <alert message="Please call back {message.origin}">
          <next>
            <Queue:enqueue queue="callback"/>
          </next>
        </alert>
      </status>
    </status-switch>
  </incoming>
</less>
```

5.1.6 Camp on call

The Camp On Call service allows the computing function to queue a call for a device (that typically is busy) until that device becomes available. The script below can handle the service.

```
<less>
  <incoming>
    <status-switch>
      <status active-call="1+">
        <media type="audio" input="wait.au" mode="sendonly">
          <accept>
            <next>
              <Queue:enqueue queue="callback"/>
            </next>
          </accept>
        </media>
      </status>
    </status-switch>
  </incoming>
</less>
```

5.1.7 Clear call

The Clear Call service releases all devices from an existing call. In the case of a conference call, this results in all devices in the conference call being released from the call. This service can be mapped to the `<terminate/>` action.

Comments: For an end system hosted multi-way calling, the `<terminate/>` action will disconnect all merged calls. This action can be used for a conference with a specific ending time.

5.1.8 Clear connection

The Clear Connection service releases a specific device from a call. In the case of a two-party call, this may result in the call being torn down. In the case of a conference call, this results in the specific party being removed from the conference. For a two-party call, it is the same as Clear Call and can be represented by the `<terminate/>` action. For a conference call, we need to provide a parameter to the `<terminate/>` action to indicate which call leg should be disconnected. We can use a SIP url to identify the call leg.

5.1.9 Conference call

The Conference Call service provides a conference of an existing held call and another active call at a conferencing device. This is the same as the 3-way calling service. It requires LESS to have a `<merge/>` command to merge multiple two-party calls into a conference call.

5.1.10 Consultation call

The Consultation Call service places an existing active call at a device on hold and initiates a new call from the same device. This is in fact two actions in LESS, using `<Media:mediaupdate/>` to put a call on hold and using `<call/>` to initiate a new call.

5.1.11 Deflect call

The Deflect Call service allows the computing function to divert a call to another destination that may be inside or outside the switching sub-domain. This can be mapped to LESS `<transfer/>` action.

5.1.12 Dial digits

The Dial Digits service allows the computing function to perform a dialling sequence that is associated with a call that has already been initiated. This service is used to send message out after a call invitation and we use `<Msg:sendmsg>` to handle the action.

5.1.13 Directed pickup call

The Directed Pickup Call service moves a specified call and connects it at a new specified destination. This results in the connection being diverted to a new destination inside the switching sub-domain. This service should not be handled by end systems, but by an application server in the network.

5.1.14 Group pickup call

The Group Pickup Call service moves a call that is a member of a specified or default pickup group to a new specified destination. The same as Directed Pickup Call, this service should be handled by an application server in the network.

5.1.15 Hold call

The Hold Call service places a connected connection on hold at the same device. The `<Media:mediaupdate/>` action in LESS handles this service and `<enqueue>` action can put a call in a queue.

5.1.16 Intrude call

The Intrude Call service adds the calling device to a call at a busy called device. Depending upon the switching function, the result will be that the calling device is either actively or silently participating in the called devices existing call or consulting with the called device with a new call. This can be handled by LESS `<merge/>` action with the `<media>` modifier set as `mode="sendonly"` for silently participating.

5.1.17 Join call

The Join Call service allows a computing function to request, on behalf of a device, that the device be joined into an existing call. This can also be handled by LESS `<merge/>` action.

5.1.18 Make call

The Make Call service allows the computing function to set up a call between a calling device and a called device. This can be handled by LESS `<call/>` action.

5.1.19 Make predictive call

The Make Predictive Call service shall originate a call between two devices by first creating a connection to the called device. The service returns a positive acknowledgement that provides the connection at the called device. Subsequent actions are taken depending upon the call progress and the actions requested. This service makes additional actions based on the result of `<call/>` action. For example,

```
<less>
  <outgoing>
    <call>
      <success>
        ....
      </success>
      <redirected>
        ...
      </redirected>
      <rejected>
        ...
      </rejected>
    </call>
  </outgoing>
</less>
```

5.1.20 Park call

The Park Call service moves a specified call at a device to a specified (parked-to) destination. Two ways to handle this in an end system, the first is to transfer the call to a resource server, the second is to change the media input of the call as described in Section 3.10.

5.1.21 Reconnect call

The Reconnect Call service will clear a specified connection at the reconnecting device and retrieve a specified held connection at the same device. This service can be handled by using the action `<terminate/>` to terminate an existing call and using `<Media:mediaupdate/>` and `<dequeue>` to retrieve a held call.

5.1.22 Retrieve call

The Retrieve Call service connects a specified held connection. This can be handled by `<dequeue>` and `<Media:mediaupdate/>` actions.

5.1.23 Send message

The Send Message service allows the computing function to send a message to one or more devices. The message, composed of one or more MIME body parts, is included in the Send Message service request. This service can be used to send messages for many different types of applications (Instant Messaging, Email, Pager, and Short Message Service (SMS), etc). In LESS, we use `<Message:sendmsg/>` action to send a message, and `<Email:send/>` to send an email.

5.1.24 Single step conference call

The Single Step Conference Call joins a new device into an existing call. This service can be repeated to make n-device conference calls (subject to switching function limits). This can be handled by the action `<merge/>`.

5.1.25 Single step transfer call

The Single Step Transfer Call service transfers an existing connection at a device to another device. This transfer is performed in a single-step, that is the device doing the transfer does not have to place the existing call on hold before issuing the Single Step Transfer Call service. This can be handled by the action `<transfer/>`.

5.1.26 Transfer call

The Transfer Call service transfers a call held at a device to an active call at the same device. The held and active calls at the transferring device shall be merged into a new call. Also, the Connections of the held and active calls at the transferring device shall become Null and their ConnectionIDs shall be released (i.e., the transferring device is no longer involved with the call). This service is in fact merge a held call and an existing call into a 3-way calling. It should be handled by LESS `<merge>` action.

6 New services

One of the biggest advantage of Internet telephony is its ability to easily integrate other Internet services, such as presence information, email, and web. The integration can introduce many new services which are impossible for PSTN networks. These new services are not mentioned in Q.1211 or 5ESS service documents. In this section, we introduces several new toplevel actions for new service triggering, several new actions for invoking other Internet services, and two new switches, `Event:event-switch` and `LOC:where-switch`, for event and location information handling.

6.1 New actions to invoke other Internet services

There are many existing Internet services, and more new services emerging. We are not going to investigate each service detail, instead, we will try to abstract the common usage of these services so we can use LESS scripts to describe how to invoke the services.

6.1.1 Email

For email, we usually care about three things, receivers, subject, and content. We can use `<location>` or `<lookup>` modifier to specify receivers, use `subject` parameter in `<email>` action for subject information, and put content as the data of an `<email>` tag. For example,

```
<location url="mailto:bob@example.com">
  <Email:send subject="This is a test">
    We simply send this as a test for our LESS scripts.
  </Email:send>
</location>
```

There is no output of the `email` action.

6.1.2 Web

There are two parameters for a web service action, one is the URI to visit, the other is the HTTP method. The URI parameter can be handled by the `<location>` and `<lookup>` modifiers. We use different action to represent different HTTP methods. For example, `<Web:get>` represents HTTP GET method, `<Web:post>` represents HTTP POST method. Below is a service example,

```
<location uri="http://www.example.com">
  <Web:get/>
</location>
```

The output of the a web action can be `succeed`, `redirect`, or `failed`.

6.1.3 Instant messaging

People use instant messaging as a complementary communication method for voice call and email. We introduce two toplevel actions `<Msg::message-coming>` for incoming message, `<UI:command command="sendmsg">` for an outgoing message, and one action `<Msg::sendmsg>` for sending a message out. For example,

```
<less>
  <Msg:message-coming>
    <address-switch field="origin">
      <address is="sip:bob@example.com">
        <location url="sip:bob@example.com">
          <Msg:sendmsg>Hi, Bob</Msg:sendmsg>
        </location>
      </address>
    </address-switch>
  </Msg:message-coming>
</less>
```

6.1.4 Event subscription and notification

If we want to specify different event subscription and notification policies by using LESS, LESS will get very complicated. IETF has some efforts on the event handling policies. For example, the Extensible Markup Language (XML) Based Format for Event Notification Filtering [1] is used to handle event subscription policies, the Presence Authorization Rules [8] is used to handle notification policies, the A Document Format for Expressing Privacy Preferences for Location [9] is used to handle location-based event notification policies. Since LESS and all these policies are XML-based, it should be straight-forward to integrate these policies into LESS. However, to make LESS simple, we will not integrate them into basic LESS definition. Instead, we decide to use existing LESS switches for presence authorization handling, and use several parameters, such as `events`, `interval`, and `expires`, in `Event:subscribe` action for event filtering. Below is an example for address-based event subscription handling.

```
<Event:subscription>
  <address-switch field="origin">
    <address is="sip:bob@example.com">
      <Event:accept>
        <next>
          <Event:notify status="closed"/>
        </next>
      </Event:accept>
    </address>
  </address-switch>
</Event:subscription>
```

The event handling introduces several new toplevel actions and actions. The `<Event:subscription>` represents an incoming event subscriptions. The `<UI:command command="subscribe"` represents an outgoing subscription attempt. The `<Event:notification>` represents an incoming or an outgoing notification, depending on the `direction` parameter. The `<Event:accept>` accepts incoming subscriptions, `<Event:deny>` denies incoming subscriptions, and `<Event:defer>` defers the decision on incoming subscription handling. The `<Event:subscribe>` triggers an outgoing subscription request, and `<Event:notify>` triggers an outgoing notification.

The <Event:subscribe> action can use the <location> and <lookup> modifiers to specify the presentities. The action has several parameter, `events` specifies the events the user is interested in. The parameter can be mapped to the <what> element in an Event Notification Filtering [1] document, `interval` specifies how often the subscriber would like to receive a notification, and `expires` provides the valid period of the subscription.

The <Event:notify> action can contain many parameters to represent different status. For example,

```
  <Event:notify status="open" Geo:LOC="506" RPID:place-type="hotel"/>
```

6.2 Event notification and event-based services

Event notifications can trigger many new services, we introduce a new switch `event-switch` for event handling. For example,

```
<Event:notification>
  <address-switch field="origin">
    <address is="sip:bob@example.com">
      <Event:event-switch>
        <Event:event status="online">
          <location url="sip:bob@example.com">
            <call/>
          </location>
        </Event:event>
      </Event:event-switch>
    </address>
  </address-switch>
</Event:notification>
```

The presentity of the event in an event-switch is always the sender of the toplevel action <Event:notification>. <Event:event-switch> tag can only be inside the <Event:notification> and <Event:subscription> tags.

6.3 Location-based services

Location information gets more and more important in people's communications. It is not only used for tracking, but also for guarding appropriate communication behaviors, triggering communication actions, and discovering available resources in a context. We defined a new switch specifically for location information handling [12]. Below is an example showing location-based call decision making.

```
<incoming>
  <LOC:where-switch>
    <LOC:where placytype="movie-theatre">
      <alert style="vibrate"/>
    </LOC:where>
  </LOC:where-switch>
</incoming>
```

By default, the principle of the location in a `LOC:where-switch` is the owner of the script. The principal can also be other people. For example, `<LOC:where-switch principal="{message.origin}">` has the sender of an incoming message as the principal.

7 Conclusion

This paper shows that most of the existing PSTN services that suitable for end systems can be handled by LESS. In addition, we can use LESS to handle many new services with the integration of other Internet services.

References

[1] An extensible markup language (XML) based format for event notification filtering. Internet Draft draft-ietf-simple-filter-format-00, Internet Engineering Task Force, February 2004. Work in progress.

[2] AT&T. *5ESS Switch, The Premier Solution, Feature Handbook, Issue 4*. AT&T, September 1987.

[3] Ecma International. Services for computer supported telecommunications applications (csta) phase iii. Standard 269, Ecma International, June 2004.

[4] International Telecommunication Union. General recommendations on telephone switching and signaling – intelligent network: Introduction to intelligent network capability set 1. Recommendation Q.1211, International Telecommunication Union, Geneva, Switzerland, March 1993.

[5] J. Lennox, Henning Schulzrinne, and Thomas La Porta. Implementing intelligent network services with the session initiation protocol. Technical Report CUCS-002-99, Columbia University, New York, New York, January 1999.

[6] J. Lennox, X. Wu, and H. Schulzrinne. Call processing language (CPL): A language for user control of internet telephony services. RFC 3880, Internet Engineering Task Force, October 2004.

[7] R. Mahy. A message summary and message waiting indication event package for the session initiation protocol (SIP). RFC 3842, Internet Engineering Task Force, August 2004.

[8] J. Rosenberg. Presence authorization rules. Internet Draft I-D, Internet Engineering Task Force, October 2004.

[9] Henning Schulzrinne. Policy rules for disclosure and modification of geographic information. Internet Draft draft-ietf-geopriv-policy-00, Internet Engineering Task Force, November 2003. Work in progress.

[10] W3C. Voice extensible markup language (voicexml) version 2.0. http://www.w3.org/TR/2001/WD-voicexml20-20011023.

[11] Xiaotao Wu and Henning Schulzrinne. Programmable end system services using SIP. In *Conference Record of the International Conference on Communications (ICC)*, May 2003.

[12] Xiaotao Wu and Henning Schulzrinne. Location-switch for call processing language (CPL). Internet draft, Internet Engineering Task Force, February 2004. Work in progress.

[13] Xiaotao Wu and Henning Schulzrinne. The simplicity and safety of the language for end system services (LESS). Technical report, Department of Computer Science, Columbia University, July 2004.