SESSION INITIATION PROTOCOL SERVLET APPLICATION PROGRAMMING INTERFACE

Technical Field

The present invention relates generally to telecommunications systems and more particularly to a Session Initiation Protocol servlet Application Programming Interface.

Background of the Invention

The Session Initiation Protocol (SIP) is an application layer protocol for creating, modifying and terminating communication sessions among computing systems or other suitable devices. SIP is defined formally in *Handley et al*, "SIP: Session Initiation Protocol," Internet Engineering Task Force, RFC 2543 (August 1999). Multiple participants may participate in each session. Examples of sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution sessions. The participants in a session may communicate via multicast, via a mesh of unicast relations or via a combination of multicast and unicast relations.

Conventional systems do not provide a convenient mechanism for developing application programs that comply with SIP. A developer must custom develop applications and ensure that the applications conform with the behavior mandated by SIP protocol. This difficulty makes application development cumbersome and discourages the proliferation of new applications that comply with SIP. Given the custom nature of such applications, maintenance of such applications may also be difficult.

Summary of the Invention

The present invention addresses the above-described limitations of conventional systems by facilitating the use of SIP servlets. The term "servlet," as used in this context, refers to a portion of computer codes that executes on a server to provide desired functionality. A servlet may be considered the server-side analog to an applet. In the present invention, the servlets provide telephone services logic. In one embodiment of the present invention, an Application Programming Interface (API) is defined for SIP servlets. An API is a set of methods that an application may use to request and carry out lower level services. The SIP servlet API identifies interfaces and objects that a servlet should support

to be a full fledged SIP servlet. The API identifies what functionality is needed for an SIP servlet.

In accordance with one aspect of the present invention, an SIP servlet is provided in a data processing apparatus. The SIP servlet is run to implement telephone service logic. As part of the telephone services logic, the SIP servlet may examine, manipulate or redirect an SIP message, such as an SIP request or an SIP response.

In accordance with another aspect of the present invention, servlets are provided in a computing environment. A servlet manager is provided for managing the servlets. A communication that complies with SIP is received. The servlet manager determines that a selected one of the servlets is to process the communication, and the selected servlet then processes the communication.

In accordance with a further aspect of the present invention, an electronic device includes an interface for receiving an SIP message. The device also includes a servlet for handling the SIP message to implement telephone service logic.

Brief Description of the Drawings

An illustrative embodiment of the present invention will be described below relative to the following drawings.

FIGURE 1 depicts a block diagram of a first system that is suitable for practicing the illustrative embodiment of the present invention.

FIGURE 2 depicts a block diagram of a second system that is suitable for practicing the illustrative embodiment.

FIGURE 3 is a flow chart illustrating the general steps performed to implement telephone services logic in the illustrative embodiment.

FIGURE 4 is a flow chart illustrating the steps that are performed to implement call screening in the illustrative embodiment.

FIGURE 5 is a flow chart illustrating the steps that are performed to implement call forwarding in the illustrative embodiment.

Detailed Description of the Invention

The illustrative embodiment sets forth an Application Programming Interface (API) for SIP servlets. Developers may use this API to develop SIP servlets that provide customized behavior. The SIP servlets provide implementations of the methods set forth in the SIP servlet API. The SIP servlets enable users to dynamically extend the functionality of the server on the fly. The SIP servlets can inspect messages, change values of message headers, redirect messages and generally handle messages to implement telephone services logic. For example, the SIP servlets can be used to implement call forwarding, call screening and mobility services. Those skilled in the art will appreciate the SIP servlets may also be used to implement additional telephone services logic.

The SIP servlets of the illustrative embodiment may be written in a programming language, such as the Java programming language from Sun Microsystems, Inc. Those skilled in the art will appreciate that the servlets may also be written in other appropriate programming languages.

SIP (i.e., sessions entail the passing of "calls") request messages and response messages. These are the two exclusive types of messages used in SIP. The response messages respond to request messages. For example, when a first user wishes to initiate a session with a second user, the first user sends an INVITE request to the second user. The second user receives the INVITE request and responds with a response message that identifies whether the second user wishes to participate in the session.

Figure 1 depicts a block diagram of a first system 10 that is suitable for practicing the illustrative embodiment of the present invention. A user employs a user device 12 that is capable of communicating via SIP. The user device 12 may be a computer system, a cellular phone, an intelligent pager, a personal digital assistant (PDA) or more generally, any electronic device that is capable of participating in an SIP session. The second user

also employs a user device 14, which may be any type of component that is capable of participating in an SIP session (such as listed above).

In Figure 1, a SIP proxy service 16 is employed. The SIP proxy server 16 is an intermediary program that acts both as a server and client for the purpose of making requests on behalf of other clients. The SIP proxy server 16 may run on a dedicated server computer system or may be run on a shared computer system. Requests are either processed by the SIP proxy server 16 or passed on, after translation, to other servers. The SIP proxy server 16 interprets and, if necessary, rewrites a request before forwarding the request. A number of servlets 22 may be resident at the SIP proxy server 16 to implement telephone services logic. A servlet manager 20 is also resident at the SIP proxy server 16. The servlet manager 20 is responsible for receiving messages and determining which of the servlets 22 are to process the messages.

The system 10 of Figure 1 also includes a user agent server 18. The user agent server 18 is a server application that contacts the user of user device 14 when an SIP request is received. In addition, the user agent server 18 returns a response on behalf of the user of user device 14. A response accepts, rejects or redirects the request. The user agent server 18 may also have associated servlet manager 24 and servlets 26.

Figure 2 shows an alternative system 28 for practicing the illustrative embodiment of the present invention. This system 28 includes user devices 30 and 32. System 28 differs from system 10 in that system 28 employs a redirect server 34 rather than a proxy server 16. The redirect server 34 is a server that accepts an SIP request, maps the address set forth in the SIP request to a new address and returns this address to a client. Unlike the SIP server 16, the redirect server 34 does not initiate its own SIP requests and, in contrast to a user agent server 18, the redirect server 34 does not accept calls. The SIP redirect server 34 has an associated servlet manager 36 and servlets 38.

Before discussing the details of the SIP servlet API provided by the illustrative embodiment, it is helpful to briefly review how the servlets may be employed. Figure 3 provides a flow chart of an overview of the steps that may be performed using the servlets. In general, a servlet is provided on one of the servers (step 40 in Figure 3). The servlet is run alone or in conjunction with other servlets to implement telephone services logic (step

42 in Figure 3). The telephone services logic may include any of a number of different types of call processing approaches that are implemented in telephone systems.

Figure 4 is a flow chart illustrating the steps that are performed to realize call screening. Initially, a server receives an INVITE request (i.e., an "invitation") for initiating a session (step 44 in Figure 4). The INVITE request contains caller information identifying the caller that wishes to initiate the session (i.e., the "call"). The caller information is noted by the servlet (step 46 in Figure 4). Based upon the caller information, the servlet determines whether to screen the call and how to screen the call (step 48 in Figure 4). The servlet may access a database or other information source that identifies the appropriate way to screen the call. The information in the database, may, for example, inform the servlet that the call is to be blocked, redirected to an operator, directed to a voicemail platform or placed in a queue.

Figure 5 is a flow chart illustrating the steps that are performed to implement call forwarding. Initially, a servlet receives an INVITE request (step 50 in Figure 5). The servlet accesses a database or another information source to obtain call processing information. In this instance, the call processing information identifies that the call is to be forwarded. As such, the servlet notes that the call has to be forwarded (step 52 in Figure 5). The INVITE request is then forwarded to the appropriate destination (step 56 in Figure 5).

When a caller wishes to make an SIP call, the caller first locates a server and sends an SIP request to the server. The most common request is an INVITE request. The SIP request may be redirected or may trigger a train of new SIP requests by proxies. Users can register their locations with SIP servers. Users are located at hosts and are identified by SIP URLs (Uniform Resource Locators). SIP URLs take the form of user@host, where the user part is a user name or telephone number and the host part is either a domain name or a numeric network address.

As mentioned above, each SIP message is either a request or a response. These messages use a generic message format as set forth in RFC 822. D. Crocker, "Standard For The Format Of ARPA Internet Text Messages," Request For Comments 822, Internet Engineering Task Force, August 1982. The generic message format includes a start line,

one or more header fields, and an empty line that indicates the end of the header fields and an optional message body.

The illustrative embodiment defines an SipServlet abstract base object class that serves as the central abstraction of the SipServlet API. This abstract base class extends the GenericService object class, which implements the servlet interface. The SipServlet class defines the default method for handling all SIP requests. This method demultiplexes each request and is responsible for invoking the appropriate method on the proper instance of a servlet. The SipServlet abstract base class defines additional methods for handling particular types of SIP requests. These methods are automatically called by the service method (i.e., the servlet manager) for processing an SIP request. Objects of the SipServlet object class support two packages: javax.servlet and servlet.sip. The javax.servlet package is a package that is defined by Sun Microsystems, Inc. of Palo Alto California. A "package" groups classes of objects by functionality. A "class" is a collection of data and methods that operate on the data. Each package groups a number of interfaces. An "interface" is akin to an abstract base class and provides a signature of methods and attributes that must be implemented by an object in order to support the interface.

The servlet.sip package is defined as follows.

```
interface SipServletRequest
interface SipServletResponse
interface SipSession
interface SipBindingListener

class SipServlet
class URI
class SessionDescription
class MediaDescription
class SipUtils
```

As can be seen above, the servlet.sip package includes four interfaces: the SipServletRequest interface, the SipServletResponse interface, the SipSession interface and the SipBindingListener interface. These Interfaces will be described in more detail

below. In addition, the servlet.sip package specifies five object classes: the SipServlet object class, the URI object class, the SessionDescription object class, the MediaDescription object class and the SipUtils object class. The SipServlet object class has been described above. The URI object class is an object that encapsulates information regarding SIP uniform resource identifiers. The SessionDescription object class holds attributes and methods relating to particular sessions, and the MediaDescription object class holds attributes and methods relating to media that may be supported by servers during an SIP call. Lastly, the SipUtils object class is an object class that includes a number of utilities.

The SipServletRequest interface is more formally defined as follows.

```
public String getAuthType();
public String getCallId();
public long getDateHeader(String name);
public String getHeader(String name);
public Enumeration getHeaderNames();
public int getIntHeader (String name);
public String getMethod();
public String getPathInfo();
public String getPathTranslated();
public String getQueryString();
public String getRemoteUser();
public String getRequestedSessionID();
public String getRequestURI();
public String getServletPath();
public SipSession getSession (boolean create);
public SipSession getSession().
public boolean isRequestedSessionIdValid();
```

The SipServlet Request interface defines an object to provide client request information to a servlet. The getAuthType method gets authentification type information from a request header. SIP supports a number of different authentication options. The

getCallId method obtains the callID from the request header. The callID is a unique identifier of a call. The getDateHeader method obtains a date header field from a request. The getHeader method obtains a header that is specified as a parameter to the method. The getHeaderNames method obtains names of the headers at a given request. The getIntHeader method obtains an integer header. The getMethod method obtains a method. The getPathInfo method retrieves an SIP URI or other path information. The getPathTranslated method obtains parameters relating to path information for a path that has been translated. The getQueryString method obtains a query string, and the getRemoteUser method gets information regarding a remote user (i.e., the caller). The getRequestedSessionID method retrieves a session ID from a request. The getRequestURI obtains a request URI, and the getServletPath method retrieves a path to a servlet. The getSession method either returns an existing session or returns a value indicating that a session has not yet been created and creates a session. The isRequestedSessionIdValid method returns a boolean value indicating whether a session ID is valid or not.

The SipServletResponse interface extends the ServletResponse interface of the java x.servlet package. The SipServletResponse Interface is defined as follows.

```
public static final int SC_TRYING;
public static final int SC_RINGING;
public static final int SC_CALL_BEGIN_FORWARDED;
public static final int SC_CALL_QUEUED;
public static final int SC_OK;
public static final int SC_MULTIPLE_CHOICES;
public static final int SC_MOVED_PERMANENTLY;
public static final int SC_MOVED_TEMPORARILY;
public static final int SC_SEE_OTHER;
public static final int SC_USE_PROXY;
public static final int SC_ALTERNATIVE_SERVICE;

public static final int SC_BAD_REQUEST;
public static final int SC_UNAUTHORIZED;
public static final int SC_PAYMENT_REQUIRED;
public static final int SC_BAD_FORBIDDEN;
```

```
public static final int SC_NOT_FOUND;
public static final int SC_METHOD_NOT_ALLOWED;
public static final int SC_PROXY_AUTHENTICATION_REQUIRED;
public static final int SC_REQUEST_TIMEOUT;
public static final int SC_CONFLICT;
public static final int SC_GONE;
public static final int SC_LENGTH_REQUIRED;
public static final int SC_REQUEST_URI_TOO_LARGE;
public static final int SC_REQUEST_ENTITY_TOO_LONG;
public static final int SC_UNSUPPORTED_MEDIA_TYPE;
public static final int SC_BAD_EXTENSION;
public static final int SC_TEMPORARLY_UNABAILABLE;
.public static final int SC_CALL_LEG_DNE;
public static final int SC_LOOP_DETECTED;
public static final int SC_TOO_MANY_HOPS;
public static final int SC_ADDRESS_INCOMPLETE;
public static final int SC_AMBIGUOUS;
public static final int SC_BUSY_HERE;
public static final int SC_SERVER_INTERNAL_ERROR;
public static final int SC_NOT_IMPLEMENTED;
public static final int SC_BAD_GATEWAY;
public static final int SC_SERVICE_UNAVAILABLE;
public static final int SC_GATEWAY_TIMEOUT;
public static final int SC_VERSION_NOT_SUPPORTED;
public static final int SC_BUSY_EVERYWHERE;
public static final int SC_DECLINE;
public static final int SC_DOES_NOT_EXIT_ANYWHERE;
public static final int SC_NOT_ACCEPTABLE;

public boolean containsHeader(String name);
public void sendError(int sc, String mesg) throws IOException;
public void sendError(int sc) throws IOException;
public void sendRedirect(String location) throws IOException;
public void setDateHeader(String name, long date);
```

```
public void setHeader(String name, String value);
public void setIntHeader(String name, int value);
public void setStatus (int sc);
public void setStatus (int sc, String sm);
```

As listed above, interface includes the definition of a number of constants (i.e., static final variable). These constants correspond to the status codes defined within the SIP specification.

The SipServletResponse interface also contains a number of methods. The containsHeader method returns a boolean value specifying whether the response contains a header or not. Multiple sendError methods are defined to generate error alarms. The input parameters may include just a status code or may include a status code and a message in String format. The sendRedirect method causes an exception to and specifies where a response is to be redirected. The setDateHeader method establishes a value for date header, and the setHeader method establishes a value for a particular named header. The setIntHeader method sets a value for an integer header. The setStatus method sets a status code and may also include the additional parameter of a String that specifies a status message.

The SipSession interface contains the following methods.

```
public long getCreationTime();
public String getId();
public long getLastAccessedTime();
public int getMaxInactiveInterval();
public Object getValue (String name);
public String [] getValueNames();
public void invalidate();
public boolean isNew();
public void putValue(String name, Object value);
public void removeValue(String name);
public void setMaxInactiveInterval(int interval);
```

The getCreationTime method returns the creation time for a session. The creation time information is retrieved from a session description object. The getId method returns a session ID. The getLastAccessedTime method returns information regarding the last time a session was accessed. The getMaxInactiveInterval method returns the largest period of time for which the session has been inactive. The getValue method returns a value for a given named attribute. The getValue names method returns the names of values in a session description object. The invalidate method invalidates the session, and the isNew method returns a boolean value specifying whether the session has been newly created or not. The putValue method establishes a value for a given attribute. The removeValue method removes a value, and the setMaxInactiveInterval method establishes a value for the maximum inactive interval attribute.

The SipSessionBindingListener interface is formally defined as follows.

```
public void valueBound (SipSessionBindingEvent event);
public void valueUnbound(SipSessionBindingEvent event.
```

The SipSessionBindingListener interface extends the EventListener interface of the javax.servlet package. This interface primarily causes an object to be notified when it is bound or unbound from a session.

As mentioned above, the servlet SIP package includes a number of objects. The SIP servlet object is more formally defined as follows.

```
public SipServlet ();
protected void doInvite(SipServletRequest req, SipServletResponse resp)
  throws ServletException, IOException;
protected long getLastModified(SipServletRequest req);
protected void doAck(SipServletRequest req, SipServletResponse resp)
  throws ServletException, IOException;
protected void doBye(SipServletRequest req, SipServletResponse resp)
  throws ServletException, IOException;
protected void doCancel(SipServletRequest req, SipServletResponse resp)
```

```
    throws ServletException, IOException;
  protected void doRegister(SipServletRequest req, SipServletResponse resp)
    throws ServletException, IOException;
  protected void doOptions(SipServletRequest req, SipServletResponse resp)
    throws ServletException, IOException;
  protected void service(SipServletRequest req, SipServletResponse resp)
    throws ServletException, IOException;
  public void service(ServletRequest req, ServletResponse resp)
    throws ServletException, IOException;
```

This object has a number of methods for handling respective types of requests. For example, the doInvite method is used for handling INVITE requests. Similar methods are also provided for the ACK, BYE, CANCEL, REGISTER and OPTIONS request. The service request provides a default service as has been described above.

The MediaDescription object is defined as follows:

```
public class MediaDescription

  protected String name;
  protected String address;
  protected String title;
  protected String connectionInfo;
  protected String bandwidthInfo;
  protected String encriptionKey;
  protected String duration;
  protected String mediaAttributes;

  public MediaDescription ();
```

It contains the name, address and title attributes for the media. In addition, connection information and bandwidth information is contained therein. An encriptionKey may be included in the media description object, and duration information specifying the

duration of information contained in the media may also be included. MediaAttributes may be contained therein.

The SessionDescription object class is defined as follows.

```
public class SessionDescription

protected String version;
protected String owner;
protected String name;
protected String sessionInfo
protected String URI;
protected String email;
protected String phone;
protected String connectionInfo;
protected String bandwidthInfo;
protected Vector sessionAttrib;
protected String duration;
protected String schedule;

public SessionDescription();
```

The session description object holds description information regarding a particular session. The attributes include a version attribute, an owner attribute and a name attribute. The attributes may also conclude session information and a URI. Email information and phone information may also be stored as attributes. Connection information and bandwidth information may be contained as attributes. Duration and schedule information may be contained as attributes and a vector of session attributes my also be contained therein.

While the present invention has been described with reference to an illustrative embodiment thereof, those skilled in the art will appreciate that various changes in form and detail may be made without departing from the intended scope as defined in the attached claims.

Claims

1. In a data processing apparatus, a method, comprising the steps of:
providing a Session Initiation Protocol (SIP) servlet; and
running the SIP servlet to implement telephone service logic.

2. The method of claim 1, wherein the data processing apparatus is an SIP server.

3. The method of claim 1, wherein, as part of the telephone service logic, the SIP servlet examines an SIP message.

4. The method of claim 1, wherein, as part of the telephone service logic, the SIP servlet processes an SIP invitation.

5. The method of claim 1, wherein, as part of the telephone service logic, the SIP servlet processes SIP requests.

6. The method of claim 1, wherein, as part of the telephone service logic, the SIP servlet processes SIP responses.

7. The method of claim 1, wherein the data processing apparatus is part of a network that supports the Internet Protocol (IP).

8. The method of claim 1, wherein the data processing apparatus is part of the Internet.

9. The method of claim 1, wherein the telephone service logic is call forwarding logic.

10. The method of claim 1, wherein the telephone service logic is call screening logic.

11. In a computing environment, a method, comprising the steps of:
providing servlets;
providing a servlet manager for managing the servlets

receiving a communication that complies with the Session Initiation Protocol (SIP);

with the servlet manager, determining that a selected one of the servlets is to process the communication; and

processing the communication with the selected servlet.

12. The method of claim 11, wherein the step of processing the communication comprises forwarding the communication to a destination.

13. The method of clam 11, wherein the step of processing the communication comprises modifying the communication.

14. .The method of claim 11, wherein the step of processing the communication comprises handling SIP requests.

15. The method of claim 11, wherein the step of processing the communication comprises handling SIP responses.

16. A medium holding instructions for execution by a data processing apparatus to perform a method, comprising the steps of:

providing a Session Initiation Protocol (SIP) servlet; and

running the SIP servlet to implement telephone service logic.

17. The medium of claim 16, wherein, as part of the telephone service logic, the SIP servlet examines an SIP message.

18. The medium of claim 16, wherein, as part of the telephone service logic, the SIP servlet processes SIP requests.

19. The medium of claim 16, wherein the telephone service logic is call forwarding logic.

20. The medium of claim 16, wherein the telephone service logic is call screening logic.

21. An electronic device, comprising:

an interface for receiving a Session Initiation Protocol (SIP) message; and

a servlet for handing the SIP message to implement telephone service logic.

22. The device of claim 21, wherein the device is a computer system.

23. The device of claim 21, further comprising additional servlets for providing additional telephone service logic.

24. The device of claim 21, wherein the device receives additional SIP messages and wherein the device further comprises additional servlets for processing the additional SIP messages.

25. The device of claim 24, wherein the device further comprises a servlet manager for determining, for each of the additional SIP messages, which of the servlets is to process the message.