

A Simple Service Access Framework for the Web

Scenera Research LLC

Cary, North Carolina

www.sceneraresearch.com

The Service Access Framework (SAF) locates and provides service access to files or other types of objects that a user retrieves through a Web browser. It associates a service with a Web-accessible object using a configurable matching condition, referred to as a service lock. A user's access to services can be regulated by its configuration through the browser, the user's ISP, or other service providers.

The SAF is a simple, generic system with the attributes of service-oriented architecture. It is provided as a single extension to browsers and has a simple, generic configuration that allows freedom for expansion when it is needed.

Introduction

The Service Access Framework (SAF) was originally designed to provide service access to Web pages and other URL-accessible objects so that users can easily view and add metadata to those objects wherever they reside on the Web (1). Once its design was realized, the resulting service access tool exceeded our original goals by allowing virtually any service to be accessible for the Web pages or objects that a user can access through a Web browser.

The SAF also reduces, and possibly eliminates, the need for installing service specific plug-ins and extensions. It is simple and generic and can be provided as a single extension to browsers. Ideally, it will be integrated into browsers as a base feature.

The process we went through in designing the system, the reasons for specific choices we made, and details of the system's architecture are described here. For clarity an example of how SAF might be used to access and acquire video multimedia on a mobile device is included.

Design

Because the number of Web-based user services is vast and those services can be accessed in many different ways, we considered several options while architecting the service access tool: a stand-alone service Web site, a stand-alone service Web site used in conjunction with a client-side agent application, and an application-specific browser extension. All of these service access

tool architectures have benefits, but their operation requires user knowledge that restricts the possible number of users of the application or its associated services.

We decided against using a stand-alone service Web site primarily because it would require users to know about the service site and to remember to use it. With this in mind, we decided that a service that is built into the browser would be the simplest and most automated. Furthermore, stand-alone Web site services are used only with a restricted set of objects. For example, a typical photo service site only provides services for photographs uploaded directly to it. Our system uses objects that are dispersed across the Internet.

We decided against a stand-alone service Web site used with a client-side agent application because the use of Web pages and extensions typically involves downloading code to run on a user's device. This design would require an independent application that runs on the same device as a browser, but runs in its own execution environment rather than in that of the browser. It would require the retrieval of code by browsers and client-side applications, which could cause problems related to security, performance, and reliability. Although it was not our initial goal to address these problems, the final design of the SAF has addressed most of them and offers the potential to address more.

Our primary reason for deciding against an application-specific browser extension, such as those used for browser spell-checkers, bookmarklets, and talking browsers, is that we were concerned that too few users would download and install it. A user would need to know that the extension exists and anticipate the need for the services that it would provide. We wanted the service to be available in contexts where it is useful without requiring its previous installation, or even previous knowledge of it. We also wanted the flexibility to easily increase the numbers and types of services it provides without updating an existing installation base.

We chose to use a browser extension that allows service access to many services. It runs in the browser's execution environment and does not require users to install extensions that are specific to the services that are accessed through it. Also, it is generic, meaning that its design allows access to many types of services. The service access extension can be built into or automatically added to Web browsers, it allows the use and on-going development of multiple services, and there are few, if any, limits to the number and types of services it provides. Its services are readily available to those who need them whenever they are needed.

Today's browsers use a wide variety of extensions that let users perform specific tasks without leaving the Web page. For example, Flickr allows users to annotate a page and access the annotations of other users (2) and it provides Flickrbar and FlickrFox extensions to Firefox so its users can readily access and use their accounts (3). Del.icio.us and other social bookmark

managers provide extensions that let users tag, annotate, rate, and associate Web sites (4). Other readily available Web service extensions include ones that search for particular words or phrases on a Web page, make a site's tables more presentable, convert a Web page to PDF, pay for items on a Web site through virtual credit cards and other accounts, and send messages to friends through Web page links. We wanted to develop a system with the potential to grow to similar ubiquity.

Design Features

The SAF has a base set of standard services that can be expanded to accommodate the needs of its users and hosts. It includes a service directory (S-directory) that maintains service profiles (S-profiles) that identify and allow access to multiple services.

An S-profile holds service descriptions (S-descriptions), each of which identifies a service to be presented by a browser and specifies how the service is accessed. The SAF associates a service specified in an S-profile with a URL-accessible object based on a configurable matching condition known as a service lock (S-lock), which is also included in each S-profile.

An S-lock is unlocked by a service key (S-key), which is a set of attribute values that are provided by a browser and associated with a URL-accessible object retrieved by the browser. Examples of attributes are an object's MIME-type, size, or source URL. When the S-key of the object matches the condition specified by an S-lock in an S-profile, the S-profile "opens," allowing access to its S-description.

The S-directory is accessed by a service URL (S-URL) that is provided to a browser for this purpose. An S-URL may be a part of a browser's configuration or provided with a URL-accessible object.

The User's Perspective: SAF Operation

While waiting for a flight from Raleigh to London, Vivien O'Hara uses her search engine to find a Web site that is devoted to Margaret Mitchell, the author of "Gone with the Wind." She had just finished reading the novel on her laptop when it was announced that her flight to London was delayed for five hours, and she was looking for ways to occupy her time while she waited. When the author's Web site is loaded, a home page toolbar appears and its buttons are updated. Some buttons are added, some are removed, and some remain.

The Web site is full of information about the author in the form of HTML pages with embedded images and links. The home page has objects that take Vivien to information about Ms. Mitchell's books, book reviews, biographical information, Pulitzer Prize, and movies made of

her books. As she moves her mouse across the page, object toolbars appear and disappear depending on which object in the page is underneath her mouse pointer.

She pauses over an image that advertises a movie of the book she had just finished reading, titled “Gone with the Wind.” Surprised that the novel was already made into a film, she looks at the toolbar and selects a button labeled “Movie.” A new page opens with various icons to take her to pages that describe how the movie was made, its cast members, its reviews, its Oscars, the history of the Civil War, and ways to purchase or view the movie. She moves her mouse pointer to an image of the movie’s poster, labeled “Watch the movie.” The toolbar updates with buttons for “Amazon,” “Netflix,” “View Online,” “Download,” “Raleigh Theaters,” “Raleigh Blockbuster,” and “Pay-per-View.”

Hoping to reserve or purchase tickets so she can see the movie when she returns from London, she goes to the “Raleigh Theaters” page and finds that the movie is no longer playing locally. Looking at her laptop clock, she sees she has plenty of time, so she returns to the “Movie” page and clicks the “View Online” button. This opens a site with a video stream to the film. There she pays for the movie through PayPal and watches it on her laptop while she waits for her flight.

Four hours later, as the final movie credits scroll to an end, Vivien looks at her clock and she sees she still has some time, so she decides to read more about this phenomenal author. She selects “Biography” from the toolbar, and is taken to a Wikipedia page that tells her all about the author’s life. There she is saddened to see that the toolbar has a button labeled “Obituary.” She selects that button and is taken to the NY Times archives where she reads the announcement of Ms. Mitchell death from injuries when she was hit by a car in Atlanta in 1949. Vivien’s astonishment at the year is broken when she hears her flight to London called, so she quickly uses the toolbars to purchase and download the movie from Direct2Drive, shuts down her laptop, and boards her plane. Once aloft, she opens her laptop and watches the movie again during her trip to London.

Architecture

Service Access Model

We chose to use a template-based service access model for simplicity, and to date we have used only URL templates. Variables in the templates are pre- and post-fixed with “%” characters, and variable names are pre-defined and known to the browser. Browsers can ignore services when they encounter unrecognized access methods, such as unrecognized variable names for URL templates. Specific examples of template features are given in sections ahead.

Messaging Protocol

At the core of the SAF is a messaging protocol. Figure 1 shows a message flow diagram with the messages M3 and M4, which are SAF protocol messages, as indicated by their solid lines.

Message M1 is an ordinary HTTP GET command shown as an example of a browser request to a Web service to retrieve a Web page or other URL-accessible object. The URL included in M1 is shown as “URL_1” in the diagram.

Message M2 is the Web service’s response to the HTTP GET. This response can include a Web page, an image to be presented in a Web page, an audio or video stream for playing in a media player, or any URL-accessible object. The browser detects the content in response messages and the MIME-type of the content, and then presents the content.

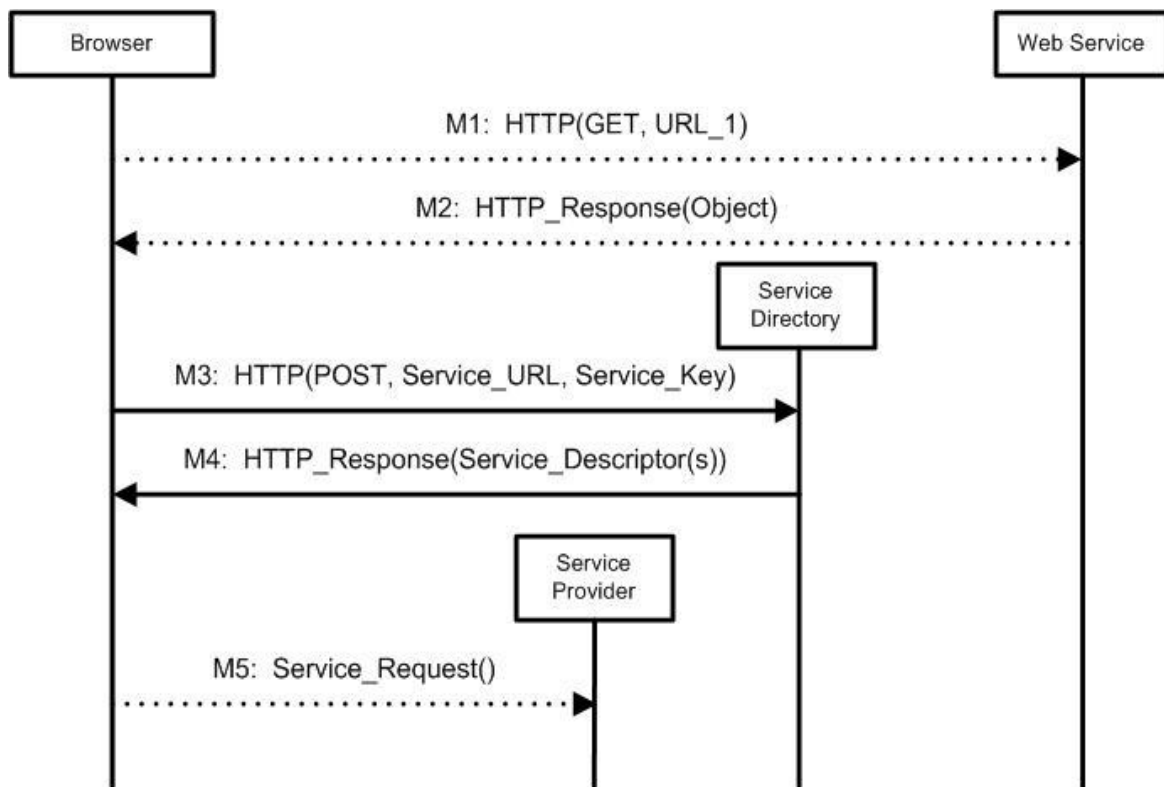


Figure 1. A message flow diagram that illustrates SAF protocol messages M3 and M4 in a typical usage scenario.

Browser Augmentation

To support SAF, the browser is augmented to perform the following:

- Detect a received object accessed through a URL, such as “URL_1” in Figure 1.
- Generate a property set (described below) for the object received.

- Locate an S-URL.
- Send an S-key based on the property set to an S-directory identified by the S-URL, as illustrated by message M3.
- Retrieve an S-description from a response sent by the S-directory, as illustrated by message M4.
- Construct a service control widget based on the S-description.
- Present the service control widget in the context of the presented object.
- When indicated by the service control widget, access the service using the method specified in the S-description, as illustrated by message M5 sent to a service provider.

Property Set

An object property is an attribute accessible to the browser that helps identify a service that is relevant in the context of an object received by the browser. The SAF uses a small set of properties and a simple schema to format a property set.

In “The User’s Perspective” scenario, when Vivien accesses the movie on the Web the properties that the SAF uses are the object’s MIME-type and source URL. Both of those properties are currently detected by browsers, so using them for SAF requires no new property detection code for the browser.

Object properties are specified as key value pairs with an XML format that is defined by a simple schema in the SAF protocol. Example 1 illustrates a valid S-key generated from the property set for the “Gone with the Wind” video stream. It specifies the “TYPE” of the object as a MIME type and specifies the “SOURCE” of an object as the URL used to retrieve it. An S-key message can be sent to an S-directory using an available S-URL.

```
<s-key>
  <key-value>
    <key>TYPE</key>
    <value>video/avi</value>
  </key-value>
  <key-value>
    <key>SOURCE</key>
    <value>http://watch.online/stream/gwtw.avi</value>
  </key-value>
</s-key>
```

Example 1. An S-key key for accessing services for the “Gone with the Wind” video stream (gwtw.mpv) sent in message M3.

Message and Object Transfer

To transport SAF messages, we first considered using the Extensible Messaging and Presence Protocol's (XMPP's) <iq> element. However, in keeping with the goal of simplicity, we used HTTP as the transport protocol with SAF messages transported as content using an application-specific MIME-type defined for the SAF schema.

In Figure 1, M3 depicts an HTTP POST. It includes the S-key shown in Example 1 that is sent from the browser to the S-directory. The S-directory is identified by an S-URL that can be provided by the user as a browser configuration option. Other methods for setting or obtaining an S-URL include receiving an S-URL from a Web page or other URL-accessible object and receiving an S-URL from a configuration service, such as a dynamic host configuration protocol (DHCP) service.

As illustrated in Figure 1, the S-directory receives the S-key in message M3 and uses the key to locate services with matching S-locks. Typically, the lock and key are matched by mirroring each other: in order to access a service specified in the S-profile, each value in the S-key must satisfy the corresponding key condition in the S-lock. We allow regular expressions to be used in S-locks as illustrated. In Example 2, an object will unlock the S-lock if the S-key indicates that the object TYPE is an image and that it includes a source URL with an HTTP scheme identifier. Otherwise, the set of services in the S-profile are not available to the object.

```
<s-lock>
  <key-value>
    <key>TYPE</key>
    <value>image/.+</value>
  </key-value>
  <key-value>
    <key>SOURCE</key>
    <value>http://.+</value>
  </key-value>
</s-lock>
```

Example 2. An S-profile's S-lock that is formatted for transmission in an SAF message. This lock holds a key that is designed to unlock all HTTP-accessible images.

An S-profile includes one or more S-descriptions, and each allows a service to be identified by the browser, presented to the user, and accessed in the context of the object that unlocked the S-lock of the service's S-profile.

Example 3 shows an S-description that is formatted for transmission in an SAF message for a single service, which is the video stream service used by Vivien in the “The User’s Perspective” scenario. The S-description includes a label and a reference to an image used in presenting a toolbar button, context menu item, or other user interface control. Description elements also support user help, such as help flyover text, and include information that tells the browser how to access a service when indicated.

```
<service-description>
<service>
<label>Watch online ...</label>
<icon>http://watch.online/wmv/icons/moviesvc.gif<icon>
<description>Watch this movie.</description>
<access><template><url>http://watch.online/wmv/tag?cmd=add&id="%S
SOURCE%</url><template></access>
</service>
</service-description>
```

Example 3. An S-profile’s S-description. It is formatted for transmission in an SAF message that identifies a service to view an image to be accessed using a URL template.

SAF Operation

In “The User’s Perspective,” Vivien’s browser received an S-URL provided by her ISP. In the context of Figure 1, the Web page that included the movie is retrieved by her browser through messages M1 and M2. Her browser then generates an S-key based on the MIME-type of the video and its source URL. The S-key is sent to the S-directory identified by the S-URL in message M3, and the S-directory locates an S-lock that is unlocked by the S-key received in message M3. The S-directory then returns the S-descriptions included in the S-profile of the opened S-lock, as depicted by message M4.

When Vivien’s browser receives the message M4, the S-descriptions are associated with the movie. The services specified in the received S-descriptions are displayed as buttons on her browser toolbar when her mouse pointer is over the video stream object.

When she selects the toolbar’s online viewing button generated from the S-description for the movie services, the browser fills the URL template’s “%SOURCE%” variable with the URL of the video stream. The browser then accesses the online movie service by sending an HTTP command depicted as message M5 in Figure 1 to “watch.online” as identified in the access URL in the S-description (Example 3).

The HTTP response that is received is processed the way that browsers typically process HTTP responses. That is, the movie is presented in a Web page that allows Vivien to access and watch it in the context of the video stream.

Future Expansion

We have planned several avenues for the expansion of SAF. Some will depend on future application need and others may be the outcome of system use. The areas of expansion we explored most during the development of SAF are service locks, protocols, support, and use as a service-oriented architecture.

Service Locks

The simple S-key requests and S-description responses may become more sophisticated as additional properties are made accessible to a browser, such as:

- The direct properties of a received object.
- Properties generated from information inside the message that delivered the object.
- Properties generated from objects received with the object, such as a Web page that includes an image received.
- Properties generated from user activities detected in the context of an object. For example, whether or not a user has seen, clicked, or downloaded a particular object.

Also, metadata can be collected automatically using event-based and contextual notification systems. We may implement analogous techniques in collecting contextual properties of a browser and browsing activities for use in S-keys.

Protocol Expansion

The SAF protocol may be viewed as a very simple service access protocol. We have considered the pros and cons of using a standard protocol such as the Simple Object Access Protocol (SOAP).

SAF's simple protocol and SOAP both benefit from their Web interoperability. In its original form, SOAP was relatively simple and has grown in complexity over time and expanded use. For instance, it required extensions to overcome the bottlenecks that developed when it was used in high volume scientific applications (5, 6) and it has suffered performance problems in high volume mobile Web-service applications (7), often requiring more extensions and other modifications. As its use continues to expand to new applications and technologies, SOAP will

continue to expand to meet resulting needs. Whether SAF would benefit from a more sophisticated service access protocol such as SOAP is a question we hope to explore further.

Support

We plan to augment the S-directory with an update service that keeps the SAF in synch with the access methods configured in the S-profiles maintained by S-directories. This should be relatively straightforward because current browsers already support a browser extension update mechanism.

Service-Oriented Architecture

The way SAF operates, it may be viewed as a service-oriented architecture (SOA). The concept of SOA has evolved for years and will continue to evolve with the growth of Web services. Generally, an SOA is an architecture that is formed by the interaction of various services, each of which provides one or more resources (8). Together, the different services form a continuous web of resource-service-resource links and service-resource-service links.

Over time, webbed patterns of use formed by the services and resources linked with one another by SAF should emerge. These webbed patterns will be affected by different factors, such as the particular services that become available, S-lock and S-key configurations, and ownership and control of S-URLs. The significance of where the control of S-URLs will lie is of particular interest to us and is yet to be determined. For example, S-URLs available to a browser may be controlled by users; by network owners, such as an enterprise or Internet Service Provider; and by third-parties, such as a browser's default search provider. We plan to study the SAF to determine whether the detectable patterns correspond to workflows associated with the goals of small groups, and how those patterns are affected by the factors we've identified.

Acknowledgments

The author would like to thank Julie Tomlinson for her writing and editing and her support in making this paper possible.

References

1. "[The Internet Metadata Library.](#)" Scenera Research LLC, 2008. 14pp.
2. M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. 2007. Visualizing tags over time. *ACM Transactions on the Web (TWEB)*; 1(2):7, 1-22.

3. L. Baker. [Flickr Extensions for Firefox – Flickbar and FlickrFox](#). Search Engine Journal, Aug 25, 2005.
4. J. Golbeck and M.M. Wasser. 2007. Social Browsing: integrating social networks and web browsing. Conference on Human Factors in Computing Systems, San Jose, CA, pp.2381-2386.
5. N. Abu-Ghazaleh and M. J. Lewis, “Differential Deserialization for Optimized SOAP Performance.” *Proc. 2005 ACM/IEEE conference on Supercomputing*. 2005. pp 21-31.
6. K. Chiu, M. Govindaraju, and R. Bramley. “Investigating the Limits of SOAP Performance for Scientific Computing.” *Proc. 11th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 2002. pp 246.
7. S. Oh and G. C. Fox.”Optimizing Web Service Messaging Performance in Mobile Computing.” *Future Generation Computer Systems*, vol. 23, no. 4. 2007. pp. 623-632.
8. T. Erl. “Service-Oriented Architecture (SOA): Concepts, Technology, and Design.” Prentice Hall, NJ. 2005. 760 pgs.

Contact

Contact Scenera Research at info@sceneraresearch.com.