A Case Study on the Automatic Composition of Network Application Mashups

Maxim Shevertalov and Spiros Mancoridis

Department of Computer Science, College of Engineering, Drexel University
3141 Chestnut Street, Philadelphia, PA 19104, USA
{ms333, spiros}@drexel.edu

Abstract

MaxMash is a tool that can compose select features of networked application and generate the source code for application mashups that can integrate those features. This paper presents a case study that demonstrates how MaxMash is used to combine the Jabber chatting protocol and the Microsoft Maps web application. The composed mashup is able to answer direction queries via a chat client.

1 Introduction and Background

Mashups integrate features of different networked applications and present these features in a novel and useful way. For example, the application RentoMeter [9] uses GoogleMaps [4] and its own collection of rent pricing information to provide customers with a comparative search of apartments for rent. RentoMeter annotates a GoogleMaps view of a geographic region with apartment labels that indicate rental prices.

The creation of mashups is a non-trivial task. Some applications simplify this task by providing an open API. However, the majority of the applications can only be accessed via a web browser or a closed source client. Even if an API is provided, developers still have to spend time understanding it, especially how the API functions can be called by other applications. Developers must also create adapter functions to convert the output of one application into the input of another.

To support the development of mashups, Microsoft and Yahoo have released tools that allow developers to visually program using specific APIs. The tool from Microsoft is named `Popfly` [12], and Yahoo calls their analogous tool `Yahoo Pipes` [3]. These tools provide adapters for several open APIs. They represent an API as a box in a diagram and allow developers to draw arrows to signify data flow between APIs. While these tools simplify the process of creating mashups, they do not eliminate the need for developers to comprehend complex APIs. MaxMash enables developers to create application mashups by extracting and reusing features of networked applications. It takes captured network traffic as input and produces as output the source code for an application implementing the features that were executed. Developers can specify how different features interact with each other by presenting those interactions in their use cases. As an example, MaxMash is able to generate an IM bot that listens for driving directions requests, queries a mapping website such as MSN Maps [8] or GoogleMaps [4], and returns a set of directions.

This paper describes a case study that presents how MaxMash was used to compose a Jabber chatting client with a web based mapping application into a mashup to answer direction queries. The details of the techniques implemented by MaxMash have been omitted for the sake of brevity.

2 Case Study

To demonstrate the capabilities and limitations of MaxMash, this section presents an example involving the Extensible Messaging and Presence Protocol (XMPP), commonly known as the Jabber protocol [10]. XMPP was designed to exchange structured information between two network endpoints in close to real time. While the definition of the XMPP protocol is general, it was developed with instant messaging applications in mind.

This case study uses select features of XMPP in conjunction with HTTP to create an application that can be queried for directions using a Jabber client. The study describes how MaxMash can be used to create such a application using the following steps:

1. Define the application we wish to create (Section 2.1)
2. Specify the use cases (Section 2.2)
3. Extract the protocol of the composite application (Section 2.3)
4. Generate the source code of the mashup (Section 2.4)

The workflow of this process is illustrated in Figure 1.

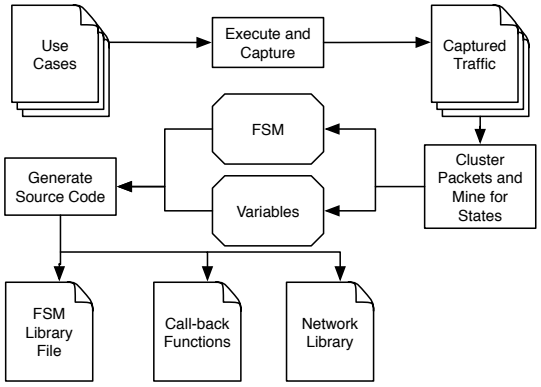

Figure 1. Workflow for generating a mashup's source code.

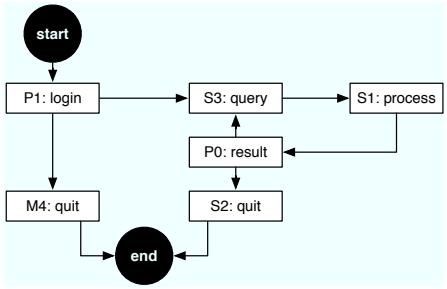

Figure 2. The FSM extracted by MaxMash. Each state is labeled by an id, chosen by MaxMash, and a description.

2.1 Defining the Application

The application we composed is called DirQ. The interface to it is a Jabber chat client. Users on the same Jabber network are able to query DirQ for directions between two geographic addresses. DirQ looks up directions using MSN Maps [8], and returns a textual result.

DirQ accepts one of two commands, a direction query or the `quit` command. The `quit` command instructs DirQ to shut down and leave the Jabber network. The direction query has the following form: "`DIR <Start Address> TO <End Address>`". DirQ accepts queries one at a time and processes them in the order the queries were received.

2.2 Specifying the Use Cases

This case study used the OpenFire Jabber server [2], the Jeti Jabber client [1], and the Firefox [7] web-browser when acquiring directions from MSN Maps. The `TCPDump` [6] tool was used to capture network traffic. At the end of the composition process, DirQ is able to imitate a user answering direction queries using Jeti and Firefox.

Because users can query DirQ any number of times, until the `quit` message is received, use cases that demonstrate this behavior were needed. The final application's behavior should be such that it can quit without processing a single query and that queries can be processed in a loop. This behavior was specified using three use cases.

In the first use case, the client logs into the Jabber server, receives a `quit` command, and logs out. The second use case has the client perform a single direction query. In the final use case the client performs three direction queries and then quits using a `quit` command.

The key is to provide enough examples of diverse input to capture the features required. In this instance, each use case involves request for directions between different addresses, thus ensuring that MaxMash can infer that addresses are variable. Similarly, all of the use cases use the same user name and password combination, hence guaranteeing that MaxMash would infer that the application has only one user. To generate an application that supports multiple users, use cases with variable user names and passwords should be executed. As will be demonstrated later, the lack of variability in the use cases can be compensated for by manual programming. However, better use cases decrease the need for MaxMash users to intervene in the automatic creation of the mashup.

A key point here is that MaxMash is not trying to extract the "correct" protocols of XMPP and HTTP, but a new protocol that is specific to the use cases.

2.3 Extracting the Protocol of the Composite Mashup

Once the use cases are executed and the data packets are captured, MaxMash clusters the packets based on their similarity. MaxMash users can set the clustering distance threshold on a scale of zero to one. A distance of zero signifies that the packets are identical and a distance of one signifies that the packets have nothing in common. In this case study we chose the clustering distance threshold to be 0.17. We chose such a small distance as a clustering threshold, because both the XML and HTML messages, used by XMPP and HTTP respectively, are verbose and similar in syntax. Packets contain similar data and, thus, clustering packets with a larger deviation would group unrelated packets together. Having chosen this value, MaxMash made one error in the clustering that had to be corrected manually. Specifically, two clusters that were composed of packets containing the driving directions produced by MSN Maps needed to be clustered together. The MaxMash user had to manually cluster those packets because they were not similar enough. This directly relates to our choice of geographic address used in the use cases. Had those addresses been more similar, this clustering error would not have occurred. Once the clustering was corrected, MaxMash generated the composite FSM found in Figure 2.

The composite FSM has six states, two of which are

quit states, M4 and S2. The difference between these two states is that M4 contains an additional packet, which was sent by the client immediately before disconnecting from the server.

One of the problems we ran into early in this case study, and the motivation behind using Jeti, was that Jabber clients did not always produce the same stream of packets for the same user input. Jabber clients are constantly announcing their status to the server and it was difficult to find a client that did not generate seemingly random packets throughout a use case. Some clients would send a presence packet to the server every few seconds, others just queried the server for information. We chose Jeti because it simply waits for user input. However, even with Jeti there were some inconsistencies. In the first use case, Jeti added a packet after the quit command was received. It is not clear what that packet does, but it introduced a deviation from the predicted FSM.

The loop in the final FSM in Figure 2 represents the query process. State S3 represents a packet containing the direction query, and state S2 represents a packet containing the quit message. Thus, the choice from P0 is made by receiving a message from a user to either query again or quit. State S1 queries MSN Maps and state P1 logs DirQ into the Jabber server.

2.4 Generating Source Code

With the FSM of the composed mashup extracted, MaxMash is able to generate a client application that implements the mashup. MaxMash first attempts to discover variable data in the provided use cases. It discovers variables on a cluster by cluster basis. In this case study about a third of the clusters contained variable information. Most of the discovered variables were message id values used by XMPP. These message id values increased by 1 with every packet the client sent, beginning at some arbitrary value. Due to the limited number of use cases, executed in a small time frame, all of the id values began with the same set of digits. Only the hundredth place and lower values changed. Therefore, MaxMash assumed that the higher order parts of the id values were constant to the protocol. This error had to be corrected manually via the MaxMash interface. However, if more use cases were provided, such that the full id value changed, MaxMash would not have made that mistake.

OpenFire was configured to use a digest for authentication. The digest is computed by concatenating a special value provided by the server with the user's password and then hashing that produced string using the SHA1 algorithm [5]. Because the provided id was different in each test case, the digest value was different as well. However, these digests shared several characters in common and therefore MaxMash mistakenly identified those parts of the digest as constants to the protocol. Thus, MaxMash extracted about a dozen variables from the digest. That problem had to be corrected by hand. As with the id problem, this may also be corrected with more use cases.

```
def S_3(packet)
  #create constant variables
  tempV_0=60.chr+109.chr+101.chr+...
  tempV_1=34.chr+32.chr+116.chr+...
  tempV_2=32.chr+84.chr+79.chr+...
  tempV_3=60.chr+"\\"+47.chr+98.chr+...
  tempV_4=32.chr+84.chr+79.chr+...
  tempV_5=60.chr+"\\"+47.chr+98.chr+...

  #see if the regex matches the packet
  match = (/#{tempV_0}(.*)...}/m =~ packet) != nil

  #if it matches, extract information out of the packet
  if match
    id,from,to,from_2,to_2=/.../m.match(packet)[1,5]
  end

  #call back function to process the information
  processCluster_7(id, from, to, from_2, to_2)
end

def processCluster_7(id, from, to, from_2, to_2)
  #save state to globals, user modified
  $id   = id
  $from = from
  $to   = to
end
```

Figure 3. Ruby functions generated to process state S3. The ... represents code that was removed for brevity. .chr converts a numerical value to its respective ASCII character counter.

Once variables were corrected and renamed to semantically relevant names, to make the generated code more human-readable, MaxMash generated about 1000 lines of code, most of it to implement the FSM. Fewer than 50 lines of code had to be modified to get the final client functioning as specified, taking about 30 minutes of programming to do so.

Figure 3 contains the Ruby code responsible for processing state S3. It begins by first creating the constant parts of the packet to be processed. It then makes sure that the received packet matches what is expected, and extracts the variable data. That data is passed to the specified call-back function, called processCluster_7. This function was modified by the MaxMash user to store the variable data extracted from the received packet and passed in from S3. Note that the from_2 and to_2 variables are ignored because in every use case they were duplicates of the from and to variables. It is not clear why XMPP does this.

Two functions that require more attention from the MaxMash user were the call-back function for creating the digest and the call-back function for returning directions to the user (not shown in Figure 3). The algorithm to compute the digest was known because it is a configurable parameter in OpenFire. Thus, that task required only a few extra lines of code. The call-back function for returning directions required a few lines of simple parsing code. Because MSN Maps is HTML based, it returned directions in the HTML format.

Ruby's regex library was used to remove HTML-specific elements from the returned data and sent the response back as clear text.

At the end of the process, the generated code can be executed using the Ruby interpreter. It logs into the specified Jabber server and waits for direction queries. Once a direction query is issued, the client accesses MSN Maps to get the result. This result is then reformatted and transmitted back to the requesting user as a Jabber message. The user is also able to type `quit` into the Jabber client to instruct DirQ to quit. The entire process, including setting up the test suite, took less than three hours to complete.

3 Conclusions

This paper presents a case study using MaxMash. The study combined select features of the Jabber protocol with HTTP. In addition to this case study, we have applied MaxMash to several other network protocols. Specifically, an early version of MaxMash has been used to compare the implementation of the FTP protocol between different FTP clients [11]. It has also been used to generate a client implementing a subset of the FTP protocol.

In future work we will explore techniques to further remove the need for programming by the users. MaxMash may be able to minimize the need for programming by employing some of the methods other mashup tools use, such as visual programming. We want to isolate user modifications to a single part of the process. Currently manual intervention may be needed if clustering or variable identification mistakes were made. We would like to modify our technique so that all manual intervention happens before the source code is generated.

MaxMash was designed to assist developers in creating network application mashups. While other mashup creation tools work directly with application APIs, MaxMash has no such limitation. It uses the underlying network traffic to create the mashup application automatically with little human intervention.

References

[1] E. de Boer. Jeti, jabber in java. `http://jeti.sourceforge.net/`.

[2] G. Dombiak. Ignite realtime: Openfire server. `http://www.igniterealtime.org/projects/openfire/index.jsp`.

[3] J. Fagan. Mashing up Multiple Web Feeds Using Yahoo! Pipes. *Computers in Libraries*, 27(10):10–178, 2007.

[4] Google. Google maps. `http://maps.google.com/`.

[5] P. Jones. RFC3174: US Secure Hash Algorithm 1 (SHA1). *Internet RFCs*, 2001.

[6] JWS. Tcpdump public repository. `http://www.tcpdump.org/`.

[7] J. McHugh. The Firefox Explosion. *Wired 2005; 13 (02): 92*, 7.

[8] MSN. Msn maps and directions. `http://www.mymapblast.com`.

[9] Rentomatic. Rentometer: Enter an address and get rental comps back! `http://www.rentometer.com`.

[10] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Instant messaging and presence. `http://www.xmpp.org/rfcs/rfc3921.html`.

[11] M. Shevertalov and S. Mancoridis. A reverse engineering tool for extracting protocols of networked applications. In *Proceeding of the 14th Working Conference on Reverse Engineering*, 2007.

[12] J. Wong. Marmite: Towards End-User Programming for the Web. *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on*, pages 270–271, 2007.