Bachelor thesis
for the course 388.041
"Seminar (mit Bachelorarbeit)"
in the winter term 2008/09

Creating a tool for converting SIP traces to SIPp XML format

Christian Kloimüllner
Matrikelnummer: 0628060 Studienkennzahl: 534

Technische Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Breitbandkommunikation

Betreuer: Dipl.-Ing. Michael Hirschbichler

Eingereicht am

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe, und ich diese Arbeit zuvor keiner anderen Stelle oder Institution als Studiums- oder Prüfungsleistung vorgelegt habe.

Wien, den

Contents

Abstract

This paper deals with the conversion of SIP traces to SIPp XML format. The SIP traces are recorded via "Wireshark" and are in the pcap format. The traces are parsed using Java and the "JPcap" library. After parsing, the converter makes a valid SIPp xml file out of the input information and the properties in the program.
A simple description of the SIP protocol and the used tools will be provided in the first chapters of this article. After that, the paper will describe the program at the user's view and the programmer's view.
The goal of this article is, that the reader can fully understand the program (user's and programer's view) and get a simple understanding of how the SIP protocol works.

1 Motivation and introduction

SIP is used for internet telecommunication services, also known as VoIP. It is a widespread protocol and used by many of today's VoIP providers. So it is easy to phone from one VoIP provider to another. But there are also exceptions (e.g. the well known "Skype" uses a proprietary protocol).
Every VoIP provider, who uses this protocol, has a SIP server. Therefore their users can communicate with each other over this SIP server and SIP servers from different VoIP providers communicate with each other too. The important thing is to test those SIP servers. Therefore a test suite is available, which is called SIPp. It accapts test cases in a special XML format. The structure of these XML files can be found at the website of the tool `http://sipp.sourceforge.net/` and some examples will be given in this article.
The effort for manually creating those XML files is very high. So the idea was to trace real SIP calls via Wireshark and convert them to SIPp XML format. This can also be done manually, but the effort is very high too. So the idea was to develop a converter, which automatically converts SIP traces (in pcap format) to SIPp XML files. The manual effort should be minimized to setting the properties in the converter and the user should get the correct SIPp XML file out of a simple SIP trace from Wireshark.

2 Basics

2.1 SIP (Session Initiation Protocol)

This section deals with the basic concept of SIP. The contents of this section are based on `http://www.radvision.com` [1]. More information can be found on their website.

Table 1: List of methods [2]

Method	Description
INVITE	The inititation of a call
ACK	Confirmation (acknowledgement) of the connection
BYE	Ends the call
CANCEL	Simply cancels the call
OPTION	The capabilites the opposite end point has
REGISTER	Registers the UA at the registrar (e.g. the telco)

Table 2: List of response classes [2]

Class	Description	Example
1xx	provisional (optional)	100 Trying, 180 Ringing
2xx	successfull	200 OK
3xx	redirection	300 Multiple choices
4xx	request failure	400 Bad Request, 403 Forbidden
5xx	server failure	500 Server Internal Error, 501 Not Implemented
6xx	global failures	600 Busy Everywhere, 603 Decline

User agents

Every SIP communication consists of two UA (user agents). A UA is an end point and can therefore be a physical SIP phone or a software tool. There can be distinguished between two types of UAs:

- User Agent Client (UAC)
- User Agent Server (UAS)

User agent client

A UAC sends requests, which are called methods. There are six of them, which are described in table 1.

These are the most often used methods. There can exist even more of them regarding to the capabilites of the client tool and the SIP server. Methods can also be extended in the future and are therefore not limited.

User agent server

An UAS receives those requests and creates responses. These responses are like HTTP status messages. You can divide them into different classes. Table 2 shows the meaning of these classes with examples.

SIP flow

Now the job of a UAC and UAS is clear: The UAC creates requests, the UAS receives those requests and responds (via status messages) to them (e.g. 180 TRYING). E.g. if the UAC sends an INVITE method, the UAS could respond with an 180 RINGING, showing that he got the INVITE. So it is clear, that on the request side the UAC is acting and on the opposite side (receiver) the UAS is acting. A sample SIP flow can be seen in figure 1.

Figure 1 shows a theoretical SIP flow. In reality there is also a registrar and a proxy server, where the registrar and the proxy server can be the same. The UA can register itself via the registrar (REGISTER method) and can call somebody else via the proxy server (INVITE). Therefore the registrar and the proxy server belong to the VoIP provider.

SIP packets

A captured SIP packet is shown below. It was captured by Wireshark [5]

```
Via: SIP/2.0/UDP 10.0.1.1:5062;branch=z9hG4bK748FFC26
CSeq: 650 INVITE
To: <sip:mhirschb@a1.net>
Proxy-Authorization: Digest username="ckloimuellner",
       realm="a1.net",
       nonce="8094686e654018a0a0e3e56d36261bb8",
       uri="sip:mhirschb@a1.net", qop=auth, cnonce="abcdefghi",
       nc=00000001, response="aaef1adb9778d6c4515c655c64f62a0d",
       opaque="", algorithm="MD5"
Content-Type: application/sdp
From: "Christian Kloimllner" <sip:ckloimuellner@a1.net>;
       tag=2A1B0645
Call-ID: 1782951970@10.0.1.1
Subject: sip:ckloimuellner@a1.net
Content-Length: 220
User-Agent: kphone/4.2
Contact: "Christian Kloimllner" <sip:ckloimuellner@10.0.1.1:5062;
       transport=udp>
```

This is an REGISTER method. The meaning of the fields are

- Via: Every participant fills in his address in the Via field, so that the packet can be traced back. When the packet arrives at the receiver, there are many addresses in the Via field, via which the response will be sent. Therefore the response will take exactly the same way like the request took. In case there are only two end points participated in the SIP call, there will be only one address in the Via field.
- Cseq: A sequential number and the method of the current packet
- To: The receiver of the packet.

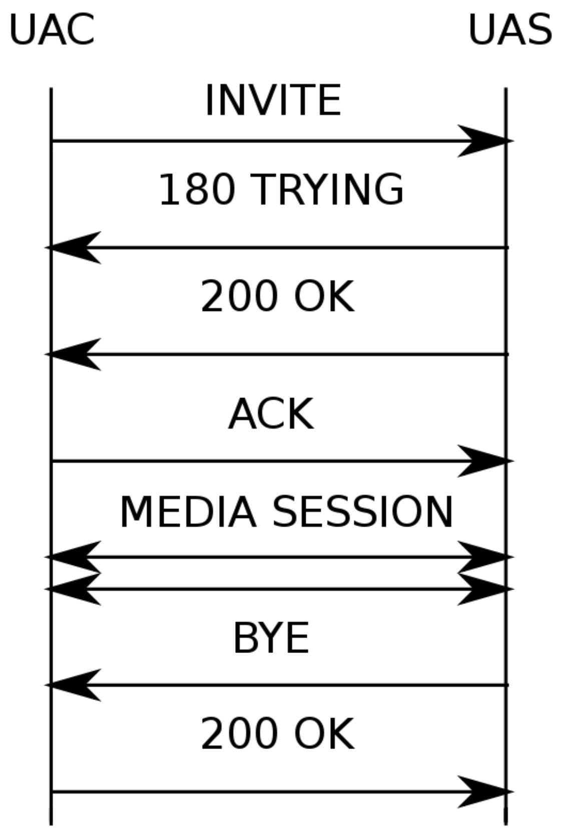


Figure 1: SIP flow [2]

- Expires: The number of hops, after which the packet should be dropped.
- From: The own address (the sender)
- Call-ID: The id of the current call. It identifies the call.
- Content-Length: The length of the content, if some payload is appended (e.g. SDP, which desribes the media used for this session)
- User agent: The physical VoIP telephone or the name of client application.
- Event: What is happening at the moment.
- Allow-Events: Which events are allowed.
- Contact: In the contact field is information about "how to contact" the sender. It consits of an SIP URI, the transport protocol (UDP or TCP) and some methods, which are supported by the sender's end gadget capabilities. It can be extended by tags, but it is not guarenteed, that all clients can read these tags.

SDP

For appending some payload SDP (Session Description Protocol) is used. E.g. this payload describes, which audio codecs can be used or should be used. The caller and callee will agree on one codec, which both understand. Of course it is also possible to add other information like video codecs. A sample packet of SDP is shown below. More information can be found in [3]

```
v=0
o=mhandley 2890844526 2890842807 IN IP4 126.16.64.4
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.cs.ucl.ac.uk/staff/M.Handley/sdp.03.ps
e=mjh@isi.edu (Mark Handley)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
```

2.2 SIPp [4]

This tool is developed for testing SIP infrastructures. It reads in XML files and generates the traffic of these files. The XML files specify the SIP flow. The user, who wants to test his SIP servers has to specify when the UAC client sends methods and defines the responses of the UAS. SIPp can simulate both, but it is also possible to simulate only one of them and to call an existent user (e.g. a software phone). There are many possibilities, which can be specified

in those XML files for SIPp. For example it is possible to specify pauses with distributions. More information can be found on the official site, especially in the documentation [4].
The tool is open source and there are executables for windows and linux. A platform indepent source distribution is available for download too.

2.3 Wireshark [5]

Wireshark is a sophisticated recording tool for analyzing network traffic. It is possible to see most (nearly all) packets in detail. It is developed by an Open-Source community. There are 631 contributors for wireshark including the original author. The big advantage is, that wireshark is updated very often. New protocols are added time by time. There are many more features like output the result in very many different formats, read/write to different sniffer formats (e.g. pcap), support for decryption algorithms (e.g. WPA), support for display filters and much more.
Wireshark is an open source tool too (published under GPL). There are executables available for Windows and Mac OS. A platform independent source distribution is available too. But there are also executables for linux available, which are mostly maintained by the package maintainers (debian - deb, rel-based - rpm).

Relation to this article

Why Wireshark is so important for this article? Wireshark makes it possible to capture network traffic. The captured packets can be analyzed in details. Therefore SIP packets are supported by wireshark. These can be extracted to plain text. If there is a SIP call, the packets can be captured using Wireshark, then extracted to plain text and finally used as a scenario for SIPp.
The manual conversion of the SIP-trace to this special XML format of SIPp is complex. A better idea is to store the trace as a pcap-file (tcpdump format) and let it be converted by a program. The output of this program can be used as input for SIPp. This is the task of the converter, which is described in chapter 3.

2.4 JPcap [6]

JPcap is a java library for capturing network traffic and analyzing the captured network packets. It also supports the visualisation of captured network traffic. The official homepage on sourceforge is not maintained any more, but on the new homepage [6] there is also a discussion board for problems with this library. Traces recorded with a different pcap version than the one on the acutal file system are not supported and an exception is thrown on runtime.
An alternative solution to read the pcap-files, is to convert the hexadecimal content to plain text, where this text represents the packets of the dump file.

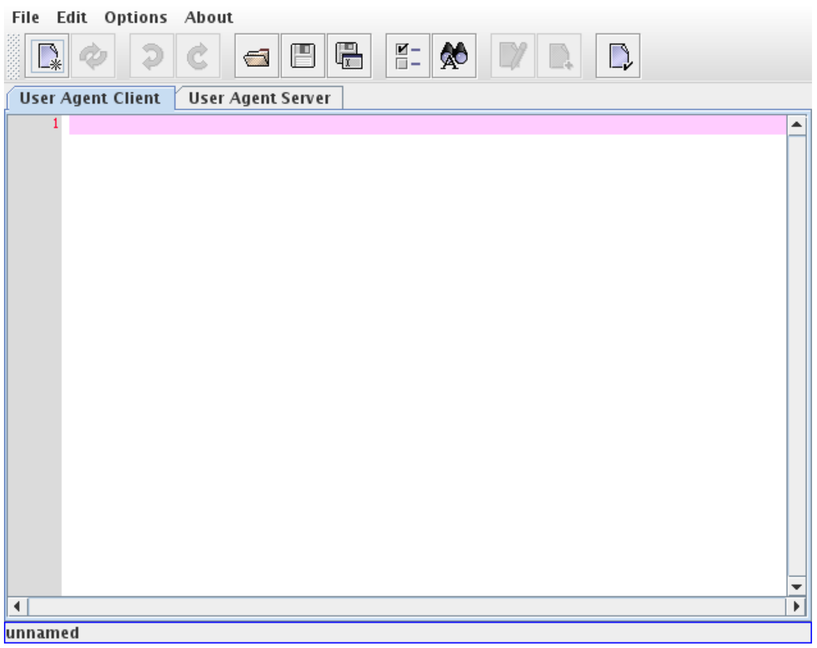


Figure 2: Screenshot of the base screen of the application

The functionlity of this GPL-licensed library is comprehensive. More information on this library can be found on the official homepage [6], especially in the javadoc, which can also be found on this website.

3 SIPp XML Converter

The name of the created tool is “Sippie”. The intention was to create a lightweight, easy to use tool, which should be expressed by a suitable program name.

3.1 User’s view

This section describes the converter from the user’s view. A screenshot of the whole application can be seen in figure 2.

The menu at the top is the same as in all other applications. Below the menubar is the toolbar. These items are shortcuts, which can be used instead of the menu. The next components show the output of the converter. Everytime a pcap file is converted, a xml file for a UAC and one for a UAS will be created. With the tabbed pane, the view between these two outputs can be changed. Below the content pane is the status bar. There will be displayed the name of the file, if it is saved. UAC and UAS will be saved in a different file so this status

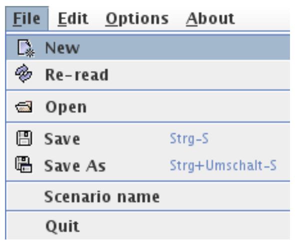


Figure 3: Screenshot of the file menu

bar will change, which depends on which scenario is selected. If the scenario is not saved so far, there will be a star ('*') displayed. In the next section the menu will be examined in detail.

3.1.1 Menu

File

- New: Select a pcap file from an file open dialog and then choose the call id, which should be converted.
- Re-read: Rereads the last selected pcap file with the options given in the standard replacing rules. All changes are going to be undone.
- Open: Shows a file open dialog for importing an xml file.
- Save: Saves the current XML file, which is displayed in the editor (either UAC or UAS). If both scenarios are needed, the must be saved seperately.
- Save as: Saves the current XML file under a different filename.
- Quit: Closes the application. If at least one scenario is not saved, the user will be asked if he wants to save this scenario.

Edit

- Copy: Copies the selected text to the clipboard (CTRL-C is not working at the moment).
- Paste: Pasts the actual content from the clipboard to the current position of the cursor (CTRL-V is not working at the moment).
- Insert Tag: Inserts a new tag at the current position of the caret.

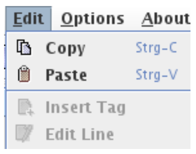


Figure 4: Screenshot of the edit menu

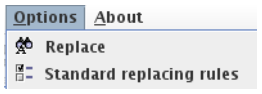


Figure 5: Screenshot of the options menu

- Edit line: The user has the possibility to edit one line of the editor. It depends on the current position of the cursor, what happens. It can be one of the following actions:
  - Tag edit dialog: If the cursor is on a line with a tag, the tag edit dialog will be shown. There the user has the possiblity to change attributes of this tag and also to add an action (sub tag) to specific elements.
  - Replacing dialog: If the user is on a line with normal text (no tag line), the application will show a normal replacing dialog.

Options

- Replace: With this option the user is able to make a “replace on the fly”. What the user forgot to replace with the standard replacing rules can be replaced here (e.g. ip address). But it is not possible to get the normal text from the replaced one.
- Standard Replacing Rules: Here can be defined, what should be replaced by default. The replacing rules defined here will be used directly for converting the pcap file. Everytime the standard replacing rules are changed, they are saved to a configuration file and are restored on start of the application.

3.1.2 Toolbar

All options found here are also accessible using the menu.

1. Select a pcap file from a file open dialog, then select a call id and convert it.

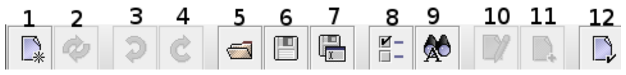


Figure 6: Screenshot of the toolbar

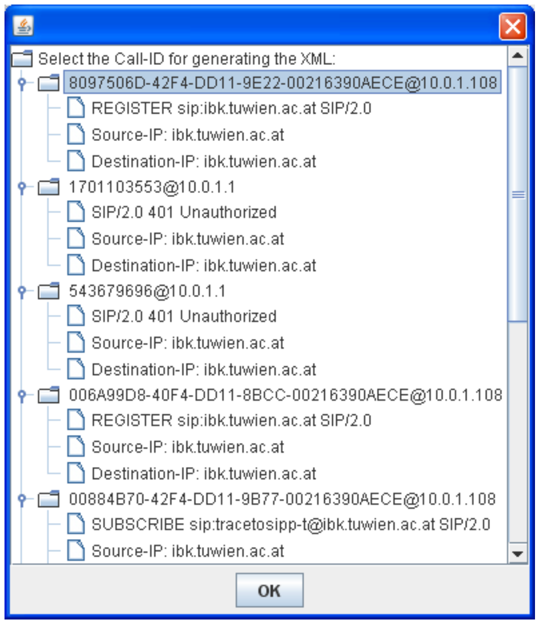
2. Reread the last selected pcap file.
3. Undo the last task made.
4. Redo the last undone task.
5. Open an xml file to be edited.
6. Save the xml file shown in the editor.
7. Save the xml file shown in the editor with a new name.
8. Set the standard replacing rules, which are applied immediately when reading a pcap file.
9. Replace information with tags used by SIPp (e.g. replace ip address).
10. Can be used to make a special edit of a line of the XML file. It can be a tag, if the caret is placed on a tag or a normal replace dialog if the caret is placed on normal text (e.g. “apply for all”).
11. Add a new tag to the xml file. This can be every tag, which is supported by SIPp and more (e.g. pause). You can use own tags. This is for future SIPp extensions.
12. Validate xml file. You have the possibility to validate the produced xml file against the sipp.dtd.

3.1.3 Call-ID selection

Figure 7 shows the call selection. When the “new” button is clicked, this dialog is shown. In a pcap file can be multiple call ids, each of them representing a SIP dialog. It is only possible to select the top entries of the tree, the children give only information about the associated call.

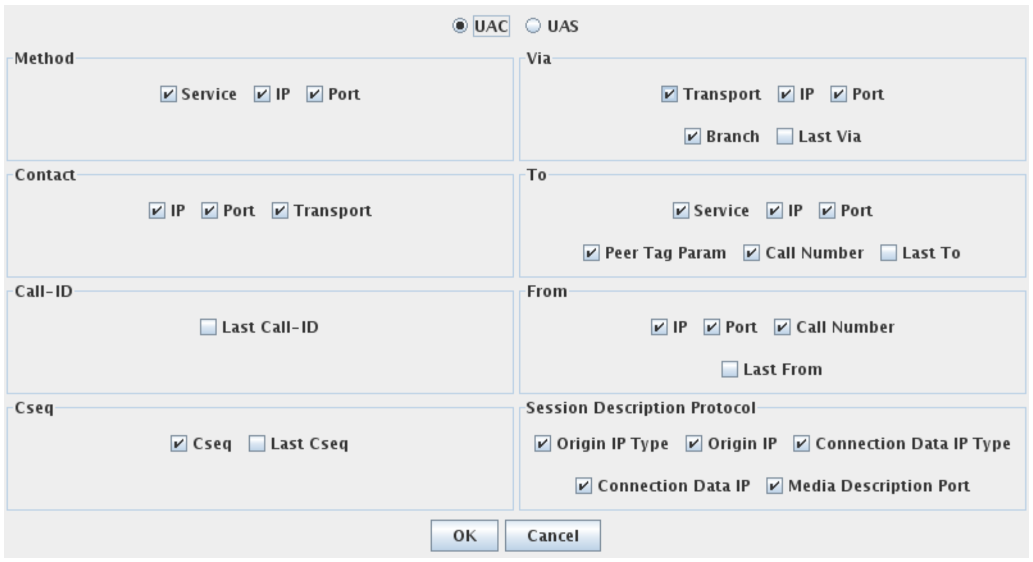
3.1.4 Standard replacing rules

The standard replacing rules are applied, when the user is converting a new pcap file or rereading the last one. On top of the dialog the user has the choice to change between UAC and UAS view. These two were handled different when conversion takes place. In section “service”, TCP oder UDP is replaced with the “[service]”-field. SIPp replaces the expression in brackets with the transport protocol specified as an option at the SIPp command line. The user



Select the Call-ID for generating the XML:
- 8097506D-42F4-DD11-9E22-00216390AECE@10.0.1.108
  - REGISTER sip:ibk.tuwien.ac.at SIP/2.0
  - Source-IP: ibk.tuwien.ac.at
  - Destination-IP: ibk.tuwien.ac.at
- 1701103553@10.0.1.1
  - SIP/2.0 401 Unauthorized
  - Source-IP: ibk.tuwien.ac.at
  - Destination-IP: ibk.tuwien.ac.at
- 543679696@10.0.1.1
  - SIP/2.0 401 Unauthorized
  - Source-IP: ibk.tuwien.ac.at
  - Destination-IP: ibk.tuwien.ac.at
- 006A99D8-40F4-DD11-8BCC-00216390AECE@10.0.1.108
  - REGISTER sip:ibk.tuwien.ac.at SIP/2.0
  - Source-IP: ibk.tuwien.ac.at
  - Destination-IP: ibk.tuwien.ac.at
- 00884B70-42F4-DD11-9B77-00216390AECE@10.0.1.108
  - SUBSCRIBE sip:tracetosipp-t@ibk.tuwien.ac.at SIP/2.0
  - Source-IP: ibk.tuwien.ac.at

Figure 7: Screenshot of the call selection



[● UAC] ○ UAS

Method: ☑ Service ☑ IP ☑ Port

Via: ☑ Transport ☑ IP ☑ Port ☑ Branch ☐ Last Via

Contact: ☑ IP ☑ Port ☑ Transport

To: ☑ Service ☑ IP ☑ Port ☑ Peer Tag Param ☑ Call Number ☐ Last To

Call-ID: ☐ Last Call-ID

From: ☑ IP ☑ Port ☑ Call Number ☐ Last From

Cseq: ☑ Cseq ☐ Last Cseq

Session Description Protocol: ☑ Origin IP Type ☑ Origin IP ☑ Connection Data IP Type ☑ Connection Data IP ☑ Media Description Port

Figure 8: Standard replacing rules

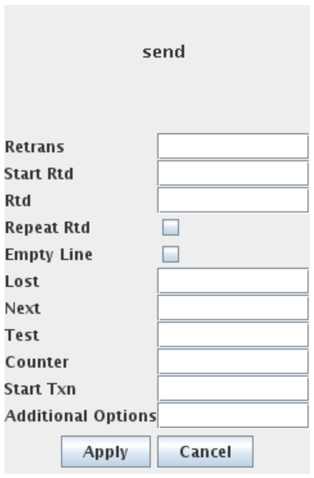


Figure 9: Edit tag dialog

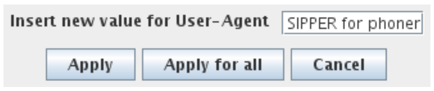


Figure 10: Edit line dialog

can specify all replacements by only activating the appropriate checkbox. The Last_* checkboxes create [last_*] fields, which are replaced automatically with the last header by SIPp. This is most useful, when creating a UAS scenario. Some options are tags of the SIP message. For example the call number in the "To" section is a tag which will be replaced if the corresponding checkbox is selected. At last, the user is also able to replace strings of the SDP. The meaning of these expressions (in brackets) should be examined with the SIPp documentation [4].

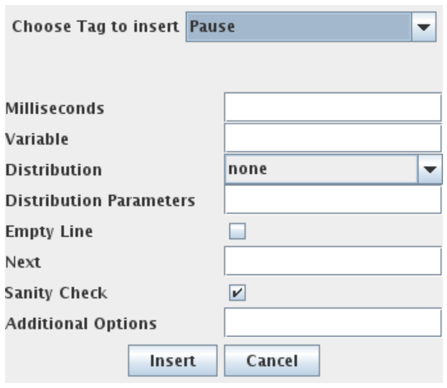
3.1.5 Edit Line

When editing a tag the dialog in figure 9 is shown.

Here, the user has the possibility to edit the attributes of a tag. If the user edits true/false attributes (checkbox values), these attributes will be set according to their default value. Other attributes will be set to the value specified. By clicking the "OK" button the changes will be applied and if the "Cancel" button is clicked the previous state will be unchanged.

If a line with "normal" text is edited the dialog in figure 10 is shown:

Here the user is able to change the text of a line. If the changes are canceled


Choose Tag to insert: Pause

Milliseconds:

Variable:

Distribution: none

Distribution Parameters:

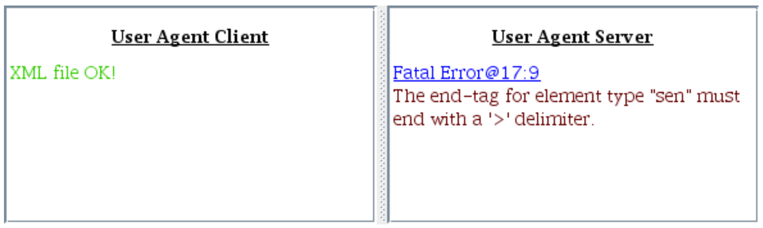
Empty Line: ☐

Next:

Sanity Check: ✓

Additional Options:

Figure 11: Add tag dialog


User Agent Client

XML file OK!

User Agent Server

Fatal Error@17:9

The end-tag for element type "sen" must end with a '>' delimiter.

Figure 12: Validate xml file dialog

the line will not be edited. If the changes are applied the line will be replaced according to your changes. If the “Apply for all” button is pressed all lines of this attribute will change to the edited value.

3.1.6 Add tag

This option can only be activated, if the caret is on the position of an end tag.

In this dialog the user has the option to add a new tag to the xml file. If a new tag should be added, which’s name is not known by the converter (maybe it was added by a new SIPp version) the user can add an “unkown tag”. There the user can specify the tag name and optional some parameters with the normal syntax “attribute=value”. The values of the attributes can be changed like mentioned in the previous chapter. The “OK” button adds a new tag to the xml file. The “Cancel” button cancels the operation.

3.1.7 Validate the xml file

In this dialog the xml file is validated. The validation displays “XML file OK!” in green color, if the validation returned with no errors. Errors are marked with red color and fatal errors with dark red. If there is an error, a link will be displayed, which can take the user to the line, where the error probably is. Another click on the validation symbol revalidates the xml file. This validation takes place with the sip.dtd file. So, if validation succeeds the xml file should be parsed by SIPp without a problem.

3.1.8 Installation instructions

The installation procedure of the program is very straightforward. The most important things to mention are the dependencies.

Dependencies

For windows systems WinPcap must be [7] installed. It should also be packaged with Wireshark [5], because it needs this library for storing the results to pcap files. The second dependency for Windows is the JPcap library. It can be found on the JPcap project site [6]. The library is also distributed as an .exe file, which can be installed by double clicking the file.

For unix systems the libpcap library must be installed. In most (nearly all) unix systems this is done by default, because it is needed by many networking tools (like as Wireshark). The only dependency to install is the JPcap library. For debian systems (e.g. (K)Ubuntu) there exists a .deb file, which can executed by typing 'dpkg -i filename.deb'. For rel-based (e.g. RedHat, Fedora) systems there is also an easy to install executable available (for rpm package management). This can be done doing 'rpm -i filename.rpm'.

For all other systems (MacOS, FreeBSD and other unix systems) there is a source distribution (packaged as tar.gz) of JPcap available too. If Wireshark is installed on a system, the libpcap library is installed too.

This is a java application. Therefore Java must also be installed. For assured compability it is recommended to install at least Sun Java version 1.6.

Installation

Sippie is packaged with a README file. This README includes information about configuration files and which of them can be edited by the user.

For installation only place the whole folder (Sippie) on a place of the harddisk. The program is executed by double clicking the jar file in the folder or by typing “java -jar Sippie.jar” on the command line. It is also possible to make a symbolic link to this jar file for easier use.

This installation instruction is identical for all operationg systems, because the program is platform independent.

3.2 Developer's view

3.2.1 Technologies

The technology used in this project includes:

- Java 1.6.0 update 10
- Swing
- external Java libraries (JDom, JPcap)
- XML
- libpcap
- SIP and SDP

The whole application is written in Java. The swing framework of Java is used for visualizing the views. Every component of the application is a swing component[1].

JDom is an external java library for xml support. It is a more enhanced version of the DOM implementation packaged with the JRE and JDK. It is used because of the easy formatting of the xml output. The JPcap library is used for reading in pcap files, which contain the recorded network traffic. Parts of this library are written in C.

The pcap files are converted to xml files. These xml files must apply to the sipp.dtd, which is located in the root directory of the application.

The libpcap library is used for reading the pcap files. It is a layer between the OS and the desktop application and/or libraries.

SIP (session initiation protocol) is the protocol which is most important for the whole application. The SIP packets are examined using the JPcap library and then converted according to standard rules defined by the user. The SDP (session description protocol) is used to deliver the media in a SIP packet and is also analyzed and converted in the xml file.

[1] Java has two packages for painting components: Swing and AWT. The difference is, that Swing draws the components by itself, whereas AWT uses the components of the operating system

3.2.2 Naming conventions

As a general rule only digits and letters are used for constructing a name (class, method, interface, variable, enum names). All names, except of variables, should be in singular.

Variables

Components, which are displayed on a frame or a dialog, have at the beginning a letter, which determine their types (e.g. `bName` is a button, `lName` is a label). There is no difference between AWT components and Swing components (e.g. `bName` can be a swing or an awt button). This rule is not used for primitives and data structures, because these types are mostly unique.

Classes and interfaces

Class names and interfaces have to begin with an upper case letter. As an additional rule interfaces must begin with an "I", so it is clear that this file contains an interface. Therefore class names must not begin with an "I". Classes in the command package must end with the word "Command".

Packages

Package names must only contain lower case letters and digits.

3.2.3 Package structure

`converter`

This is the main package, which contains all other packages. It is on the top of the other packages.

`converter.gui`

This package contains all elements concerning the GUI. It contains:

- The mainframe
- Commands
- Dialogs
- File Filters

The mainframe is in the top of the gui package. It contains four other packages, which were named below.

`converter.gui.command`

This package (`converter.gui.command`) contains alle commands, which can be executed during runtime of the main program. This package was introduced for using the "command pattern". Every command is an own class, which defines only the function of this single command. This has many advantages:

- Expandability: New commands can be added easily. A new class (command) must be added and connected to the mainframe.
- Clearness (manageability): The program is easier to understand due to the isolation of functionality
- Maintainability: The program is easier to maintain and errors (e.g. exceptions) can be isolated and therefore easier detected.

In this case every command is derived from a listener of the `java.awt.action` framework.

`converter.gui.dialog`

This package contains all dialogs used for this program. The mainframe is the only frame here (derived from javax.swing.JFrame). The other windows are only dialogs[2] on the top of the mainframe. These dialogs are for example tag editing, setting the standard replacing rules, validating xml files, etc.

`converter.gui.filefilter`

This package contains all file filters for the project:

- xml file filter: includes a filter for files ending with *.xml.
- pcap file filter: includes a filter for files ending with *.pcap.

Optionally the user has the possibility to show all files regardless of the file's ending. However this is controlled via the dialog or the mainframe and is no feature of the file filters. It's a special method, which has to be called on the filefilter object.

`converter.op`

This package contains the operations for convertion. The JPcap library is used, the preparation for the call id selection is done, tags are replaced and the scenarios are converted. This is all done in one single class (singleton pattern) representing the business logic of the program. The logic in the background is defined here with no graphical interface. All dialogs are using this class in order to execute the logic to represent the calculated data in their view.

[2]Dialogs are more lightweight components than frames

`converter.res`

This packages defines some useful ressources of the application:

- one entity: The entity class is called `CallSelection` and is used for capsulating one specific call with the most important data (request line, source ip and destination ip). It is very useful for delivering the data of the different calls from the operation class to the call id selection dialog.
- two data structure: The two data structures are sequential map and the undo redo list. In Java doesn't exist a map, which holds sequential data. The problem is, that the data of the call (the packets) is sequential. It is a very straightforward class, containing two `ArrayList`S, that hold the data of this map. The second data structure (the `UndoRedoList`) is used for doing undo and redo operation. It contains two stacks in the background, one for the undo operations and one for the redo operations. A stack is used, because the recent action has to be redone, so it is the easiest to push the last task to the top and if an undo operation should be executed the last task is popped from the top. It is the same for redo list. For these two data structure exist many convience methods to get some information of the data held by these data structures, which can be easily expanded, if needed.
- a clipboard connection: The class `TransferClipboard` is used for the connection with the clipboard. This class can read from the clipboard of the operating system and can write to the clipboard of the operating system. This functionality is also known as "copy and paste". It is realised with a builtin functionality of Java, the `ClipboardOwner` interface.
- an interface package: This is explained in the next paragraph.

`converter.res.interface`

This subpackage of the ressources package contains only three interfaces. One interface (`IConstants`) contains only constants used in the whole operation (operating system seperator, the width and the height of the mainframe). These constants are often used to position the dialogs in the middle of the mainframe. The `ICheckbox` interface contains the checkboxes used for visualizing the standard replacing rules. In the operation class the decision, which tags to replace, are based on the state of each of the checkboxes. Every variable in this interface is a two dimensional array of checkboxes to allow individual settings for the UAC and UAS. The last interface is `IOptionGUIElements`. This interface contains components, which show the options of xml tags (e.g. they are used for representing the dialog "edit tag"). The components can be:

- text fields: If the option is represented by text.
- checkboxes: If a yes or true value is possible.

- panels: For organizing the components on the screen.
- labels: For displaying the name of the components.

4 Conclusion

SIPp is a powerful program for testing SIP servers, but it is a hard task to write XML files for testing those servers with SIPp. Therefore the idea was to build a tool, which has the possibility to create those XML files for SIPp out of real pcap traces and feed SIPp with the output of the program. With this approach the job of testing SIP servers should be done with less effort.

Now it is possible after recording real SIP traces (by using Wireshark) to get an XML file, which can be used in conjunction with SIPp. The highlights of Sippie are:

- Easy use and converting: By pressing the convert button and selecting a pcap file the task is finished.
- Use of predefined actions: This feature makes an easy integration of predefined actions in the XML file possible (e.g. regular expressions).
- Syntax Highlighting: This program includes an editor with full support for XML syntax highlighting. Thanks to [8]
- Supports adding new tags and editing tags
- Highly reusable and extendable code

The program is available under [9]. As this tool is produced as Open Source, we are optimistic, that there will be a community using and extending Sippie.

List of Figures

References

[1] Radvision Ltd. *SIP Protocol Overview.* http://www.radvision.com/NR/rdonlyres/51855E82-BD7C-4D9D-AA8A-E822E3F4A81F/0/RADVISIONSIPProtocolOverview.pdf, Radvision Ltd., 2001.

[2] IETF1. *SIP - Session Initiation Protocol.* http://tools.ietf.org/html/rfc3261, Network Working Group, June 2002.

[3] IETF2 *SDP - Session Description Protocol.* http://www.ietf.org/rfc/rfc2327.txt, Network Working Group, April 1998.

[4] SIPp *Official site with documentation.* http://sipp.sourceforge.net/, February, 8^{th} 2009, 9:11.

[5] Wireshark *Official site* . http://www.wireshark.org/, February, 8^{th} 2009, 9:34.

[6] JPcap *Official maintained site.* http://netresearch.ics.uci.edu/kfujii/jpcap/doc/, February, 8^{th} 2009, 9:56 a.m.

[7] WinPcap *Official site for WinPcap.* http://www.winpcap.org, March, 25^{th} 2009, 1:07 a.m.

[8] JEdit *Official site for JEdit.* http://www.jedit.org/, June, 8^{th} 2009, 0:31 a.m.

[9] Sippie *Official site for Sippie on sourceforge.* http://sourceforge.net/projects/sippie, June, 8^{th} 2009, 0:39 a.m.