Ostfalia
Hochschule für angewandte
Wissenschaften

Faculty of Electrical Engineering

FINAL PROJECT

"Conception and prototypical Implementation of a CTI application exemplified on Zarafa Groupware and SIPFoundry's SipXecs"

Author: Daniel Peinado López

Tutor: Diederich Wermser and Daniel Hartmann

Wolfenbüttel, February 27th, 2012

ABSTRACT

Abstract: This project is about Computer Telephony Integration (CTI). The main objective of this project is to integrate a groupware into Voice-IP PBX system and make a "click to dial plugin" for the Zarafa Collaboration Platform (ZCP) which is on open source collaborative software. We have one mail client (Zarafa Webaccess) and we have to be able to call our contacts with one click. I will use one SipXecs server, which is an open source voice over IP telephony server, it works like PBX and it is responsible for IP phones ringing. Every user has one IP-phone and all this has to run for a local network and is useful for connecting multiple users on a local network for free, such as several offices within a business.

Keywords: Zarafa Collaboration Platform, Zarafa Webaccess, PBX, CTI, Representational State Transfer (REST), voice IP, SipXecs, PHP, Groupware.

Index

INDEX OF IMAGES

1. INTRODUCTION AND OBJECTIVES

1.1 Introduction

The increasingly strong role of Internet in business and everyday life, and the current lines of technological development in telecommunications, where there is a strong trend toward the "all IP", make logical development of IP-based technologies allow these communications and voice services.

This project is about Computer Telephony Integration (CTI) and this is a common name for any technology that allows interactions on a telephone and a computer to be integrated or coordinated. The CTI technology try to integrate all communication channels in a company, like telephony, e-mail, web, chat, fax, SMS, etc. The idea of our Project is to integrate a groupware into Voice IP-PBX System, in particular Zarafa Groupware into SipXecs PBX and I will develop a software example to make a demonstration. The software example is a click to call application which two users can call directly each other through a button that will be located on the mail box. Our users will be registered in a MySQL database, and we will configure a Groupware server (Zarafa) through which we will make a phone call to another. Groupware can be used to communicate, cooperate and coordinate. Some benefits sought by implementing groupware for project work are:

-The Groupware encourages cooperation within an organization and helps people to communicate and collaborate on common projects.

-Groupware helps to define the flow of documents and then define the work that should be done to complete a project.

- The Groupware provides users a unique way to share information.

What we want to perform on the project is that two users of a Groupware can communicate by telephone without having to dial the phone number.

We will use Apache like web server, Java like programming language, MySQL like database and SipXecs like PBX. All this will be installed in Linux OS.

Click-to-call (CTC), also known as click-to-talk, click-to-chat and click-to-text, is a form of Web-based communication in which person clicks an object (button, image or text) to request an immediate connection with another person in real-time either by phone call, Voice over Internet Protocol (VoIP), or text. Click to talk requests are most commonly made on websites but can also be initiated by hyperlinks placed in email, blogs, wikis, flash animations or video, and other Internet-based object or user interfaces.

One of the biggest advantages of VoIP is that internal communication is quick and free.

1.2 Objectives

Companies are becoming more competitive and increasingly adopt more strategies to ensure success. With this project we will ensure that the telephony took part of optimization process and corporate savings.

The purpose of our project is to integrate groupware into Voice-IP PBX system and create a connection between two computers which have an IP phone and make calls to one another. This will be an example of CTI (Computer Telephony Integration)

The main objective of the program is to save and optimize resources for employees of companies to make calls to a click, reducing the time it takes the employee to remember and dial a phone number.

SECONDARY OBJECTIVES ARE:

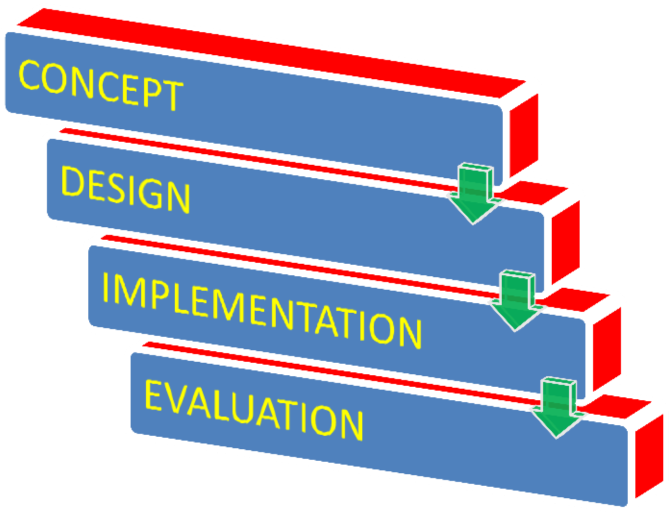
- Change the way that call centres are associated with customers.
- Making life easier for the sales department of a company connecting at any time with customers.
- Increase sales of a company.

Nowadays most of CTI applications designed are aimed for call centres. For a call centre we could understand a customer service centre which is accessed initially by telephone. Another choice would be the centre makes calls to the users offering services or requesting information instead of receive calls.

More and more companies offering telephony services, or focus their business on the telephone usage. Here we can consider different points of view. Users who wants a quick attention, the telemarketer who wants the best way for work and finally the company, which wants to obtain the best possible performance.

The user save waiting time, and its call is processed from the first moment to the right person and in many cases is not even required the intervention of the agent, allowing interactive access to databases.

1.3 Stages of development



CONCEPT

1) THEORETICAL CONCEPTS

DESIGN

2) SET UP OUR HARDWARE SYSTEM
3) FLOW CHARTS OF THE CLICK2CALL PLUGIN

IMPLEMENTATION

1) INSTALLATION AND CONFIGURATION SYSTEM
2) CHECKING ALL IS RUNNING
3) DEVELOPMENT OF THE PROGRAM CODE
4) DEBUG

EVALUATION

5) EVERYTHING WORKS FINE

2. PBX SYSTEM

2.1 Description

[2.1] A PBX or PABX (Private Branch Exchange and Private Automatic Branch Exchange) is in fact any telephone connected directly to a network through public telephone trunk lines to manage internal, incoming and outgoing calls. PBXs make connections among the internal telephones of a private organization (usually a business) and also connect them to the public switched telephone network (PSTN) via trunk lines.

PBXs are differentiated from Key systems in that users of key systems manually select their own outgoing lines, while PBXs select the outgoing line automatically.

The main advantage of PBXs was cost savings on internal phone calls: handling the circuit switching locally reduced charges for local phone service. As PBXs gained popularity, they started offering services that were not available in the operator network, such as hunt groups, call forwarding, and extension dialling.

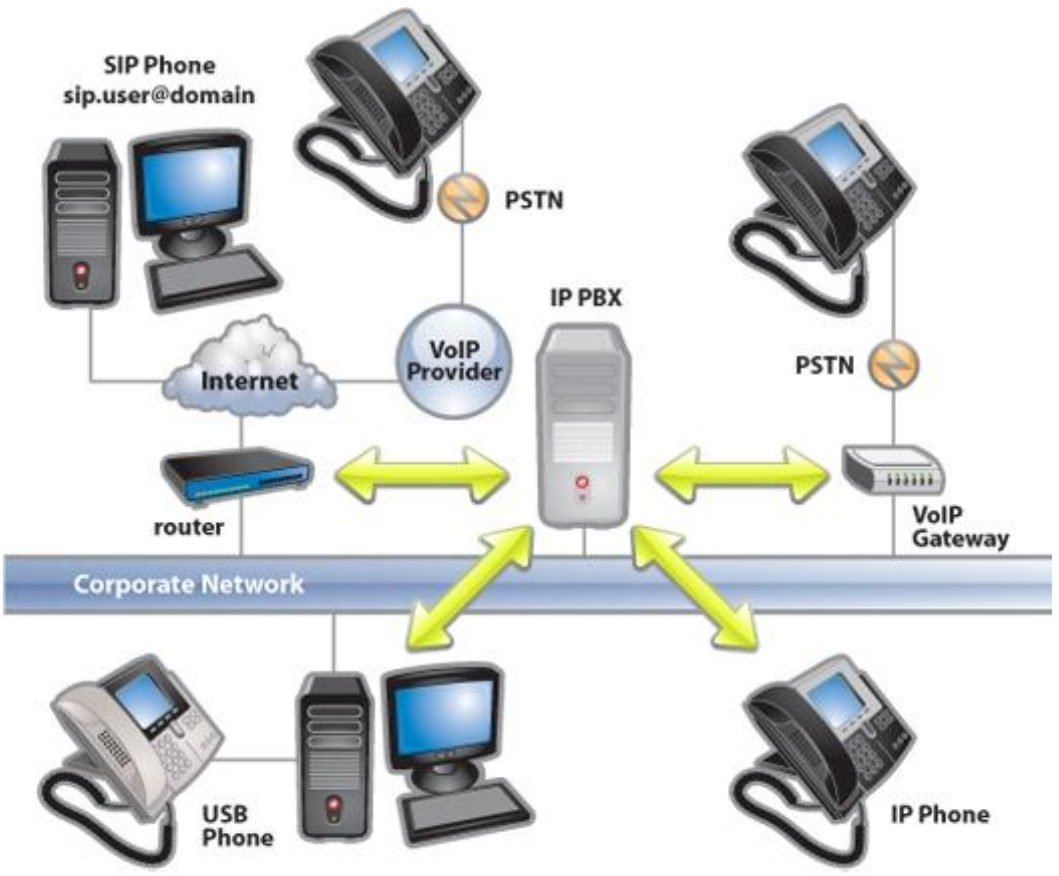


IMAGE 1 – PBX Example Architecture

2.2 History

[2.2] The term PBX was first used to designate the private central from the time that the operators manually handle perform the switching of circuits. When the first electromechanical centrals began to appear, they didn't need operators and they were known as PABX (Private Automatic Branch eXchange) and PMBX (Private Manual Branch eXchange). When these kinds of centrals were a little bit more complex, new services started to run, like switching between calls, conferences, etc.

For many years, companies with tight budgets continued using manual exchanges, even if automatic centrals were dominating the market. So the term PABX continues to be used if you have already spent more than a century of operation of the first exchange automatically.

With the rise of the first centrals with solid state components, the term EPABX (Electronic Private Automatic Branch eXchange) was established to distinguish them from electromechanical. By using electronic components and reduced in size, the cost was reduced also, allowing the emergence of private exchange to small businesses and even starting to be used in homes. With digital exchanges became possible to connect the exchange to different types of links such as that of a line or ISDN E1/T1. In addition, they multiplied the services available to users of the exchange.

During the 1990s led to new types of PBX systems. One was the massive growth of data networks and increased public understanding of packet switching. Companies needed packet switched networks for data, so using them for telephone calls was tempting, and the availability of the Internet as a global delivery system made packet switched communications even more attractive. These factors led to the development of the VoIP PBX. (Technically, nothing was being "exchanged" any more, but the abbreviation PBX was so widely understood that it remained in use.)

Currently the most used term is simply PBX, for more that no longer perform switching, no question of a branch, and sometimes that is not private. This term encompasses both large central international companies, as the centrals for local businesses and household centrals.

IMAGE 2 – Old PXB

2.3 System components

[2.1] A PBX often includes:

- The PBX's internal switching network.
- Microcontroller or microcomputer for arbitrary data processing, control and logic.
- Logic cards, switching and control cards, power cards and related devices that facilitate PBX operation.
- Stations or telephone sets, sometimes called lines.
- Outside Telco trunks that deliver signals to (and carry them from) the PBX.
- Console or switchboard allows the operator to control incoming calls.
- Uninterruptible power supply (UPS) consisting of sensors, power switches and batteries.
- Interconnecting wiring.
- Cabinets, closets, vaults and other housings.

Trunks: Are the connections between the headquarters and the PBX. (T1 and E1). Each trunk is configured independently. Businesses configure trunks as needed. For example, a customer service company needs a higher number of inbound links, whereas a telemarketing business needs more output links.

Extension lines: Each line is formed by a twisted pair, and the administrator assigned an extension number. A typical arrangement, is to have more stem extensions than trunks. But the proportion of lines per stem varies according to business needs.

Administration terminal: A computer that provides access to the administration of the PBX. It is used to configure a wide range of options and features available.

2.4 Operation

[2.2] A PBX system, is composed of four basic parts; switching matrix, control, user terminals and trunks. The PBX proper includes the control and matrix, along with matrix interface units, usually called line and trunk circuits, which terminate the transmission facilities that extend to user telephones or other switches. In addition, one usually finds "service circuits" in a PBX to apply ringing and call progress tones (busy, reorder, etc.), and to assimilate caller signaling information from telephone dials and tone pads, converting this information to something the control can use.

No matter how complex a modern PBX may become internally, it still works very much the way the old manual switchboards did in years gone by. In a manual switchboard, each line and trunk terminated in a suitable interface circuit which controlled one or more signal lamps to alert the attendant and passed the voice path to a "jack" that permitted easy inter-connection.

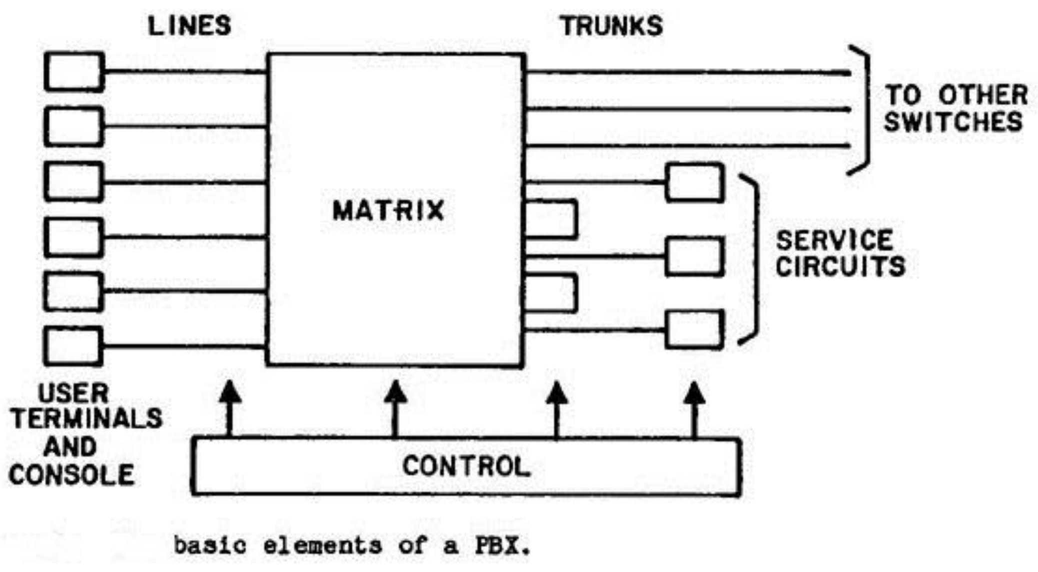


IMAGE 3 – Basic elements of a PBX

[2.1] Functionally, the PBX performs four main call processing duties:

- Establishing connections (circuits) between the telephone sets of two users (e.g. mapping a dialled number to a physical phone, ensuring the phone isn't already busy)
- Maintaining such connections as long as the users require them (i.e. channelling voice signals between the users)
- Disconnecting those connections as per the user's requirement
- Providing information for accounting purposes (e.g. metering calls)

In addition to these basic functions, PBXs offer many other calling features and capabilities, with different manufacturers providing different features in an effort to differentiate their products. Common capabilities include (manufacturers may have a different name for each capability):

- Auto attendant
- Auto dialling
- Automatic call distributor
- Automated directory services (where callers can be routed to a given employee by keying or speaking the letters of the employee's name)
- Automatic ring back
- Call accounting
- Call blocking
- Call forwarding on busy or absence
- Call park
- Call pick-up
- Call transfer
- Call waiting
- Camp-on
- Conference call
- Custom greetings
- Customised Abbreviated dialling (Speed Dialling)
- Busy Override
- Direct Inward Dialling
- Direct Inward System Access (DISA) (the ability to access internal features from an outside telephone line)
- Do not disturb (DND)
- Follow-me, also known as find-me: Determines the routing of incoming calls. The exchange is configured with a list of numbers for a person. When a call is received for that person, the exchange routes it to each number on the list in turn until either the call is answered or the list is exhausted (at which point the call may be routed to a voice mail system).
- Interactive voice response
- Music on hold
- Night service
- Public address voice paging
- Shared message boxes (where a department can have a shared voicemail box)
- Voice mail
- Voice message broadcasting
- Welcome Message

Today, it is possible to get hosted PBX service that includes far more features than were available from the first systems of this class, or to contract with companies that provide less functionality for simple needs.

In addition to the features available from premises-based PBX systems, hosted-PBX:

- Allows a single number to be presented for the entire company, despite its being geographically distributed. A company could even choose to have no premises, with workers connected from home using their domestic telephones but receiving the same features as any PBX user.
- Allows multimodal access, where employees access the network via a variety of telecommunications systems, including POTS, ISDN, cellular phones, and VOIP. This allows one extension to ring in multiple locations (either concurrently or sequentially).
- Supports integration with custom toll plans (that allow intra company calls, even from private premises, to be dialled at a cheaper rate) and integrated billing and accounting (where calls made on a private line but on the company's behalf are billed centrally to the company).
- Eliminates the need for companies to manage or pay for on-site hardware maintenance.
- Allows scalability so that a larger system is not needed if new employees are hired and so that resources are not wasted if the number of employees is reduced.

2.5 Current trends

Nowadays, is being developed in the world of free software programs that perform the functions of a PBX for Windows and Linux, as in the case of Asterisk or SipXecs program, which I use in this project. With these systems it is possible to integrate this and more functions on a single computer that provides telephone, Internet, fax, etc.

Asterisk or SipXecs can completely replace a PBX, as these programs perform all its functions and more, with no associated licensing costs. But they must deal with security and also the audio quality still not equal to the telephone.

Even though VoIP gets a great deal of press, the old circuit switched network is alive and well, and the already bought PBXs are very competitive in services with modern IP Centrex’s. Currently, there are four distinct scenarios in use:

- PBX (Private and Circuit Switched)
- Hosted/Virtual PBX (Hosted and Circuit Switched) or traditional Centrex
- IP PBX (Private and Packet Switched)
- IP Centrex or Hosted/Virtual IP (Hosted and Packet Switched)

Since in reality people want to call from the IP side to the circuit switched PSTN (SS7/ISUP), the hosted solutions usually have to manoeuvre in both realms in one way or another. The distinctions are seldom visible to the end user.

2.6 Advantages

[2.2] One of the most important advantages is that internal communication is quick and free. It also incorporates telephone lines or trunks from the public telephone network, and these can be shared and is not necessary to have a line per user. The PBX has a primary function for easy operation, and it is called call forwarding, and through this function the caller can be transferred from one user to another. This facilitates communications in a company.

As a rule an annex is receiving the calls and sends to user who request. It's known as operator. Now with modern technology, with a message welcoming the caller can be routed to the extension of the user requesting automatically by checking the requested selection.

Using a PBX avoid to connect all phones to network (PSTN) separately avoiding at same time having an own line. In small offices using phones with direct lines to the central public, private or hybrid plants, installation costs of PBX equipment would be high and the functions of these would not be fully exploited, for example, there wouldn't need to make internal calls if the office is small physically.

A PBX requires little maintenance and has an average of 10-15 years of life, for which they have become obsolete, defective, or just would not supply the capacity for growth of the company. This problem has been solved with the expandability that has got PBX. Expansion cards containing telephone jacks ports would be placed in slots designed for it and could increase the number of trunks lines connected to PBX.

Here we have five important advantages:
1. It allows all employees to share phone system lines, thus reducing your overall telecommunication expenses.
2. The PBX system is totally programmable and can support complex installation or integration requirements.
3. **There are a wide variety of standard, as well as fancy, features with the PBX system.**
4. **The PBX system is easily expandable as your company grows**.
5. **The physical PBX hub is low profile and does not require a large amount of space.**

3. CTI–COMPUTER TELEPHONY INTEGRATION

3.1 Definition

[3.1]Computer Telephony Integration (CTI) involves integrating computer system with telephony resources to augment an organization's communications capabilities. One of the main objectives is save cost and increase productivity. I consider that if we have one computer and one telephone working at the same time, exists CTI. This is a technology that enables a computer to act as a call center, accepting incoming calls and routing them to the appropriate device or person.

For example, when we call a customer service team and an automatic system guide our call through different menus we are interacting with a CTI system.

Here we have one example of CTI structure:

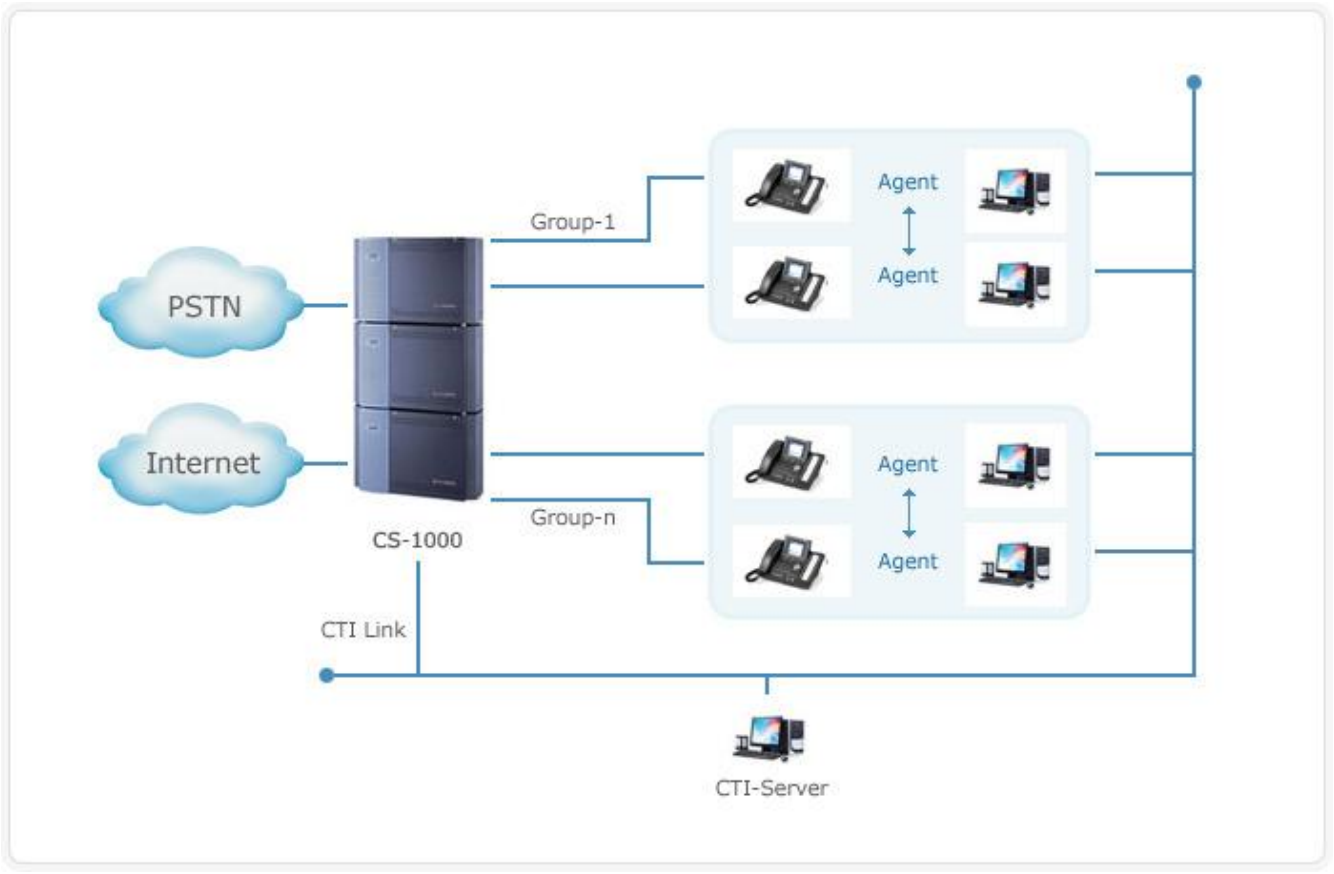


IMAGE 4 – Example of CTI Structure

3.2 CTI Components

The most useful hardware components in CTI technology are:

- PBXs
- Modems
- Computers

PBX (Private Brand eXchange): It uses to be the main component of CTI systems. The PBXs allow all kind of outside connection: analogic lines, basics and primary RDSI access, GSM lines, connections using VoIP.

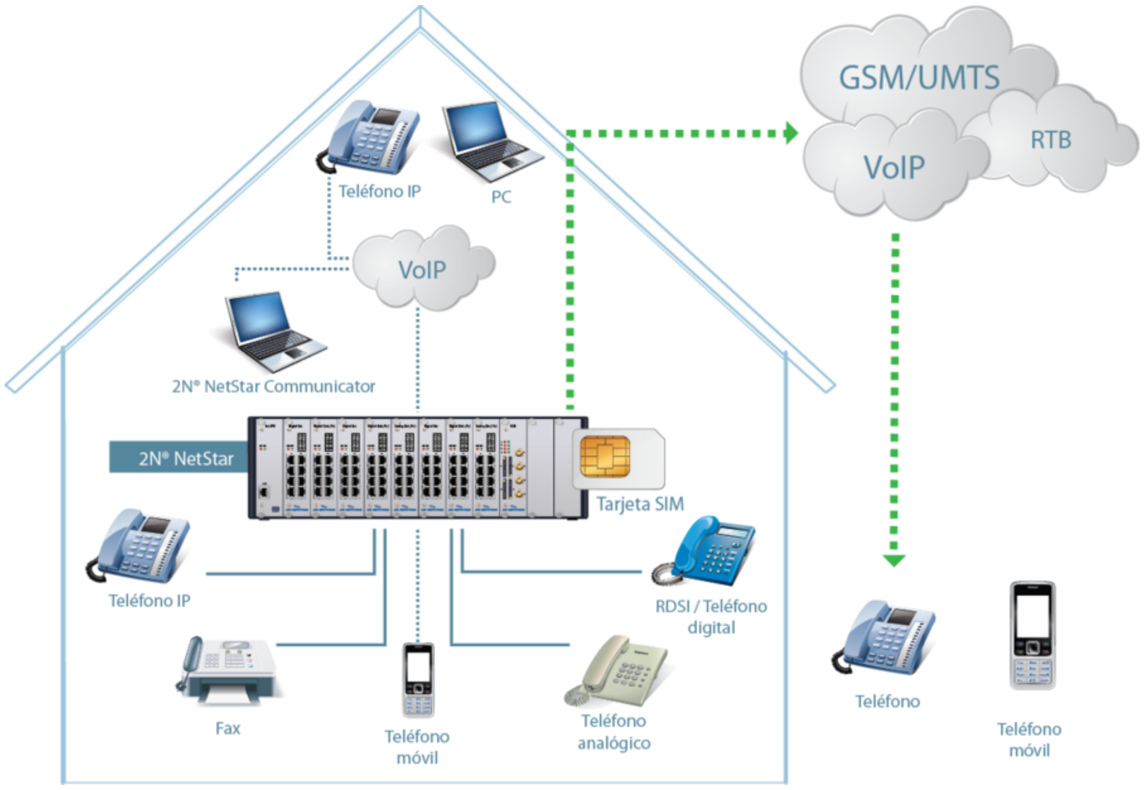


IMAGE 5 – Example of CTI with PBX as main component

Modems: Can provide CTI basic services, such as sending fax until small voice automatic response systems.

3.3 CTI Deployment Models

[3.1]We have the following models of CTI implementation:

- Third Party Model: In this model, the telephony server communicates with the electronic devices. Every computer can connect with every device if it has permission.

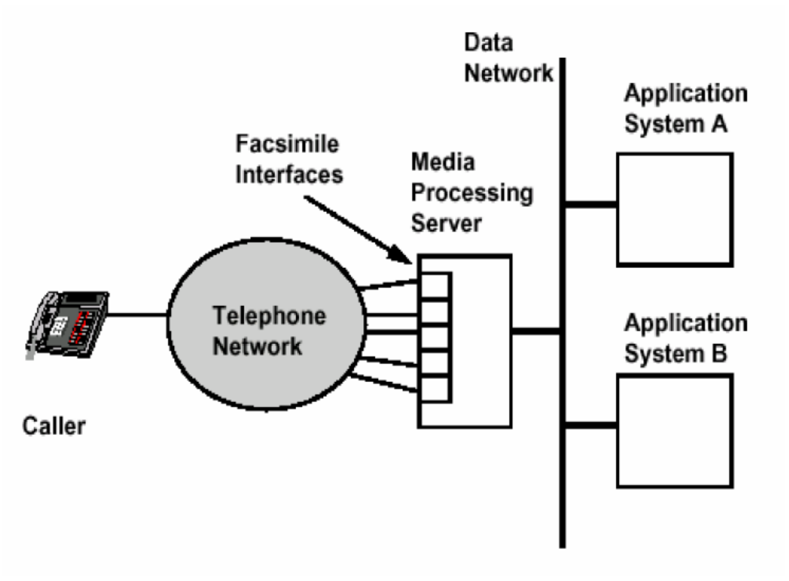


IMAGE 6 – CTI Third Party Model

- First Party Model: Each computer with CTI functions is connected to one or more electronic devices and it can access only to devices connected to it.

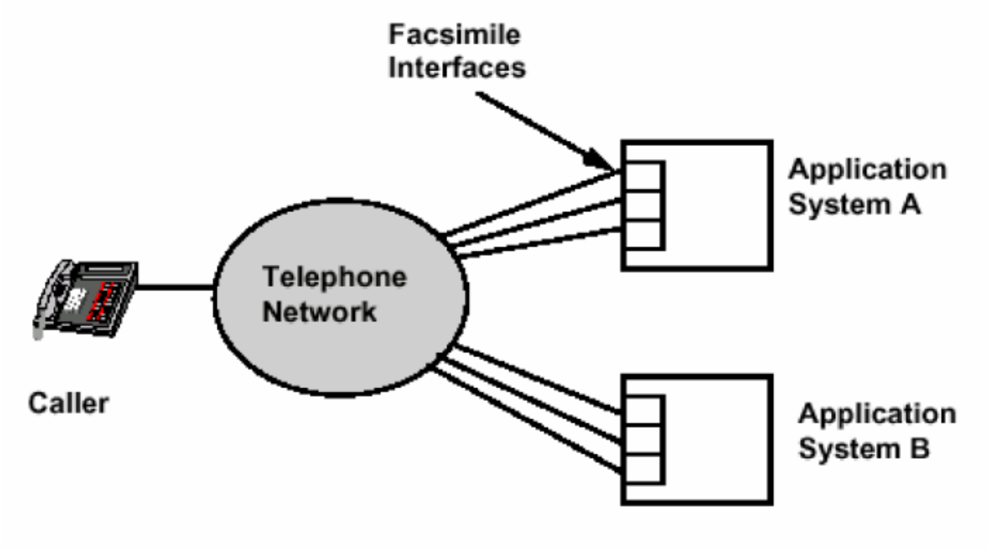


IMAGE 7 – CTI First Party Model

3.4 Types of connections

[3.2]We can classify the interconnections in CTI architectures:

- Telephony interconnections:
  - Digital: ISDN PRI, ISDN BRI, MCF R2, SS7, etc.
  - Analog
  - VoIP

- Data interconnections: Allow the exchange of information systems such as databases, sending or receiving messages over the network data, etc.

3.4 Applications of CTI

Development of CTI applications includes a big group of systems. It can be something easy such as sending a fax from a computer, or something more complicated like recognise of voice applications.

The Drivers TAPI[3.3] or TSAPI[3.4] permit a software can make a call to a telephone directly, it allows to call from the computer using the telephone to transmit voice.

For example, in this project, I'm going to integrate in Zarafa Webaccess the option of make calls between users of the mail client.

One opposite example could be that when one user receives one call and pick up the telephone, one database window opens for introduce data of one client.

In a CTI development is possible to relate the PBX with the management software of the company. It's very common to relate the database of the company so depending of the caller number, the PBX decide where to transfer that call.

Actually there is a long list of CTI applications, easy to implement and that allows us to provide a high value to our business communications.

4. VOICE IP (VOIP)

4.1 Definition

Voice over Internet Protocol (Voice over IP, VoIP) is a family of technologies, methodologies, communication protocols, and transmission techniques for the delivery of voice communications and multimedia sessions over Internet Protocol (IP) networks, such as the Internet. It is a technology that allows you to make voice calls using a broadband Internet connection instead of a regular (or analog) phone line.

VoIP is a resource that enables the voice signal to transmit over Internet. This means that the acoustic signal, i.e. the conversation between two or more people is sent to the network in data packets.

We have to differentiate between two terms in this service:

- Voice over IP (VoIP): The set of standards, devices, protocols, ultimately the technology that allows voice communication over IP protocol.
- IP Telephony: It's the phone service available to the people, with numbering, made with the prior technology.

4.2 IP Telephony versus Normal Telephony

In a normal telephony call, the central makes a permanent connection between both talkers. This connection is used to transmit the voice signal.

In one IP telephony call, the data packets contain the voice signal digitalized and compressed. These packets are sent to the IP address of the receiver. Each packet can use a different way for arrive to the destiny, and when all packets arrive to the receiver are arranged and converted to voice signal again.

The objective of IP technology is to bring technological advances to the common user and business interested in benefiting from the reduction of costs inherent in the use of internet as a platform of communication and business. The prize of this voice IP calls, are lower than normal telephony calls.

4.3 Elements

Customer: The client establishes and creates voice calls; this information is coded, packaged and transmitted through the microphone (input) user. In the same way the information is decoded and played through speakers or headphones (output). A customer could be a Skype user, or a user of a company that sells its services through IP technology equipment such as ATAs (Analog Telephone Adapter) or IP phone or softphone. The softphone is software that lets you make calls through a computer connected to Internet.

The servers: The servers are handling database operations, performed in real time. There operations are accounting, collection, routing, management and service control, registration of users, etc. Usually in the servers we have to install software called switches or IP-PBX (IP switches). Asterisk IP-PBX is one of the most used and open source.

Gateways: Gateways provide a bridge of communication between all users, its main function is to provide traditional telephony interfaces properly, which functions as a platform for virtual users (clients). Gateways are used to “finish” the call; the customer creates the call and gateway finish. That’s when a customer calls to a landline or mobile, must be the part that makes possible for the call that comes able to connect online with a customer of a fixed or Mobile Telephone Company.

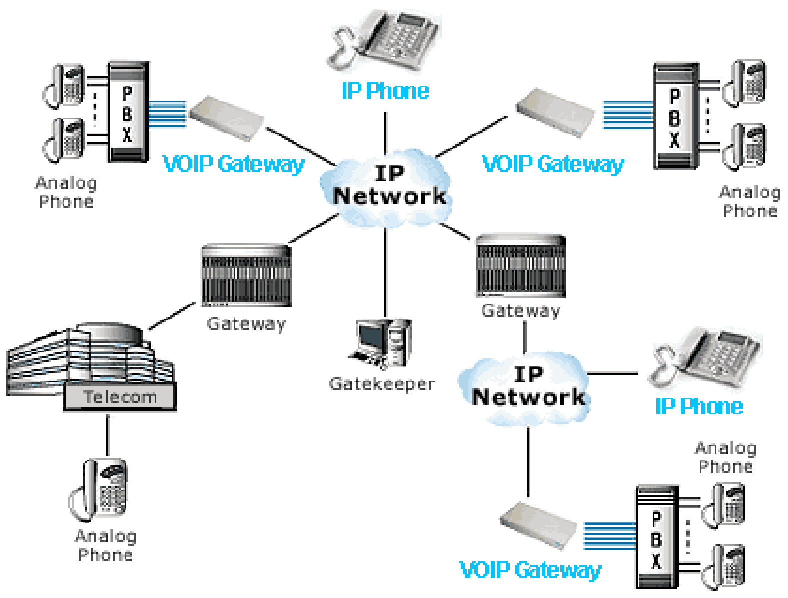


IMAGE 8 – Example of VoIP Network

4.4 Protocols

Voice over IP should be considered as a clarification of H.323. The VoIP/H.323 consists of a series of standards and is supported by a series of protocols covering different aspects of communication:

- Addressing
    - RAS (Registration, Admission and Status): Communications protocol that allows a H.323 station to locate another H.323 station through gatekeeper.
    - DNS (Domain Name Service): Service name resolution to IP address for the same purpose that the RAS protocol but through a DNS server.

- Signage
    - Q.931: Call signalling initial
    - H.225: Call control: Signalling, registration and admission, and packet/synchronization voice stream.
    - H.245: Control protocol to specify the open/close messages of channels voice stream.

- Voice Compression
    - Required: G.711 and G.723
    - Optional: G.728, G.729 and G.722

- Voice Transmission
    - UDP (User Datagram Protocol): The transmission is done using UDP packets. Although UDP doesn't provide data integrity, the use of the bandwidth is higher than with TCP.
    - RTP (Real Time Protocol): Manage aspects of timing, marking UDP packets with the information necessary for the efficient delivery of them at the reception.

- Transmission Control
    - RTCP (Real Time Control Protocol): It's mainly used to detect situations of network congestion and take corrective actions.

CALL SETUP AND CONTROL					
		Presentation			
Addressing		Audio Compression G.711 or G.723	DTMF		Addressing
RAS(H.225)	DNS	RTP/RTCP	H.245	Q.931 (H.225)	DNS
UDP Transport			TCP Transport		
IP NETWORK					
DATA LINK					
PHYSICAL					

1 - Protocol Stuck Voice-IP

Since the development of newer, less complex protocols such as MGCP and SIP, H.323 deployments are increasingly limited to carrying existing long-haul network traffic. In particular, the Session Initiation Protocol (SIP) has gained widespread VoIP market penetration.

	H.323	**SIP**
Standards body	ITU-T	IETF
Architecture	Distributed	Distributed, Peer to Peer
Call control	Gatekeeper	Proxy/Redirect server
Endpoints	Gateway, terminal	User agent
Signalling transport	TCP/UDP	TCP/UDP
Multimedia	Yes	Yes
DTMF – relay transport	RTP	RTP
Fax - relay transport	T.38	T.38
Supplemental services	By endpoints or call control	By endpoints or call control

Session Initiation Protocol (SIP) is the IETF's standard for establishing VOIP connections. SIP is an application layer control protocol for creating, modifying and terminating sessions with one or more participants. The architecture of SIP is similar to that of HTTP (client-server protocol). Requests are generated by the client and sent to the server. The server processes the requests and then sends a response to the client. A request and the responses for that request make a transaction.

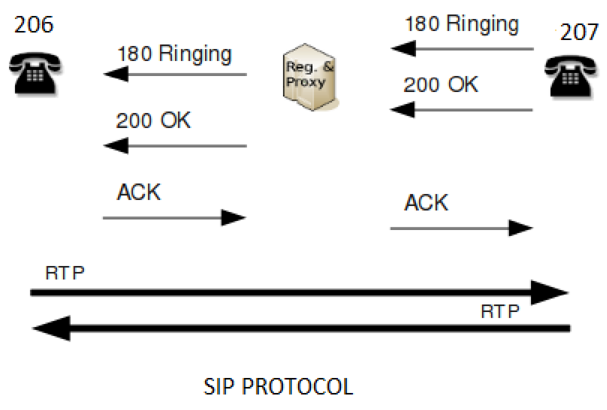


IMAGE 9 – Example of SIP Protocol

5. Representational State Transfer (REST)

5.1 Definition

[4.1] Representational state transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.

Although the term REST originally referred to a set of architectural principles, now uses to describe any simple web interface that uses XML and HTTP, without extra abstractions of protocols based on message exchange patterns such as protocol SOAP web services. System can be designed according to web services REST architectural style of Fielding and XMLHTTP interfaces can be designed according to the style of remote procedure call, but not using SOAP. These two different uses of the term REST cause some confusion in technical discussions, although RPC is not a REST example.

[4.2] Why is it called Representational State Transfer?

The Web is comprised of resources. A resource is any item of interest. For example, the Boeing Aircraft Corp may define a 747 resource. Clients may access that resource with this URL:

http://www.boeing.com/aircraft/747

A representation of the resource is returned (e.g., Boeing747.html). The representation places the client application in a state. The result of the client traversing a hyperlink in Boeing747.html is another resource is accessed. The new representation places the client application into yet another state. Thus, the client application changes (transfers) state with each resource representation --> Representational State Transfer!

Here is Roy Fielding's explanation of the meaning of Representational State Transfer:

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use."

5.2 Constraints

[4.1] The REST architectural style describes the following six constraints applied to the architecture, while leaving the implementation of the individual components free to design:

Client–server: A uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state, so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

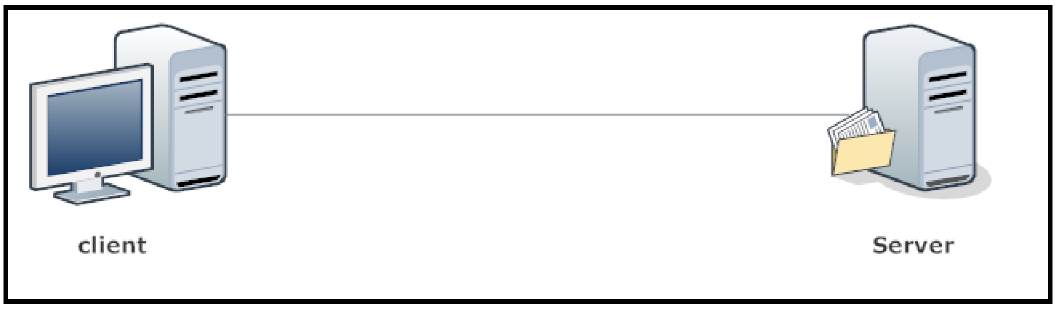


IMAGE 10 – Client-Server Communication

Stateless: The client–server communication is further constrained by no client context being stored on the server between requests. Each request from any client contains all of the information necessary to service the request, and any session state is held in the client. The server can be stateful; this constraint merely requires that server-side state be addressable by URL as a resource. This not only makes servers more visible for monitoring, but also makes them more reliable in the face of partial network failures as well as further enhancing their scalability.

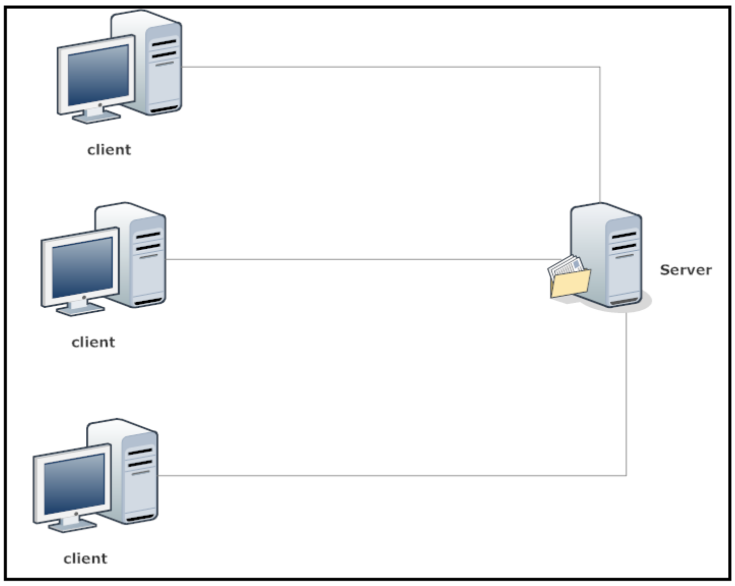


IMAGE 11 – Client-Server communication. Stateful Server

Cacheable: As on the World Wide Web, clients can cache responses. Responses must therefore, implicitly or explicitly, define themselves as cacheable, or not, to prevent clients reusing stale or inappropriate data in response to further requests. Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance.

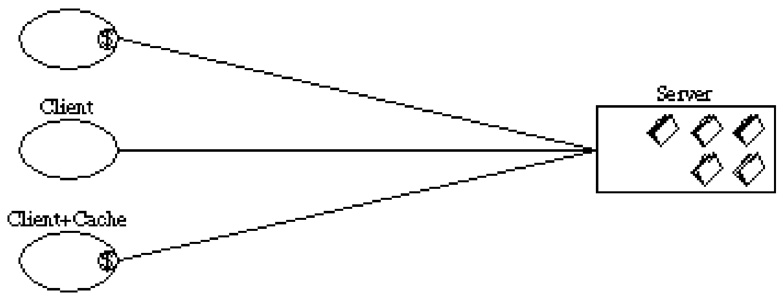


Figure 5-4. Client-Cache-Stateless-Server

IMAGE 12 – Client Cache Stateless Server

Layered system: A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. They may also enforce security policies.

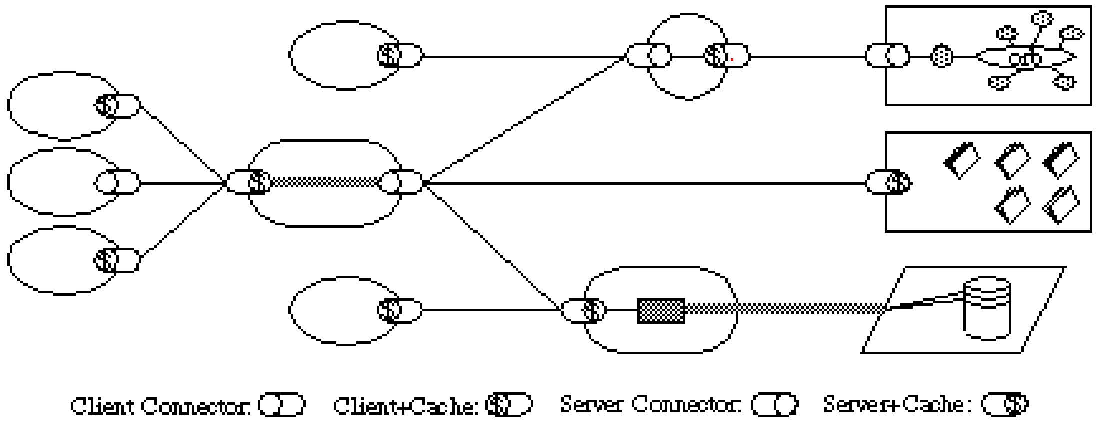


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

IMAGE 13 – Layered System.Uniform-Layered-Client-Cache-Server

Code on demand (optional): Servers are able to temporarily extend or customize the functionality of a client by transferring logic to it that it can execute. Examples of this may include compiled components such as Java applets and client-side scripts such as JavaScript.

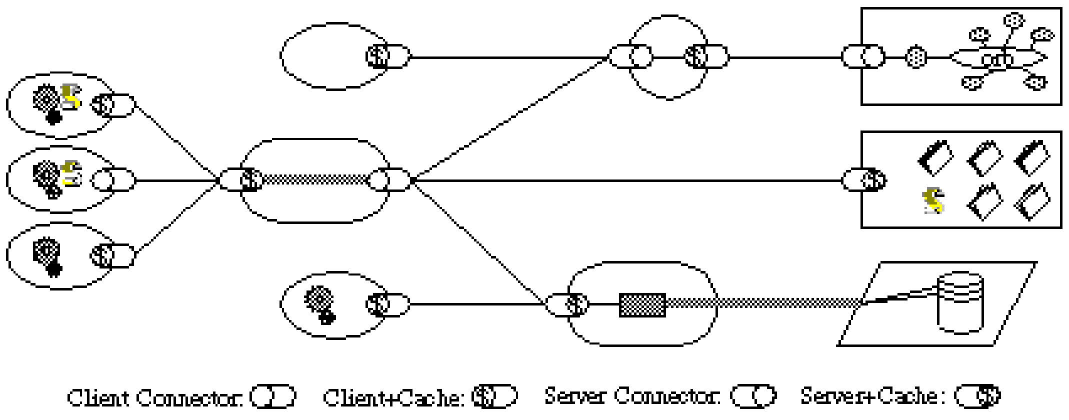


Figure 5-8. REST

IMAGE 14 – Code on demand

Uniform interface: The uniform interface between clients and servers, discussed below, simplifies and decouples the architecture, which enables each part to evolve independently. The four guiding principles of this interface are detailed below.

The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

Complying with these constraints, and thus conforming to the REST architectural style, will enable any kind of distributed hypermedia system to have desirable emergent properties, such as performance, scalability, simplicity, modifiability, visibility, portability and reliability.

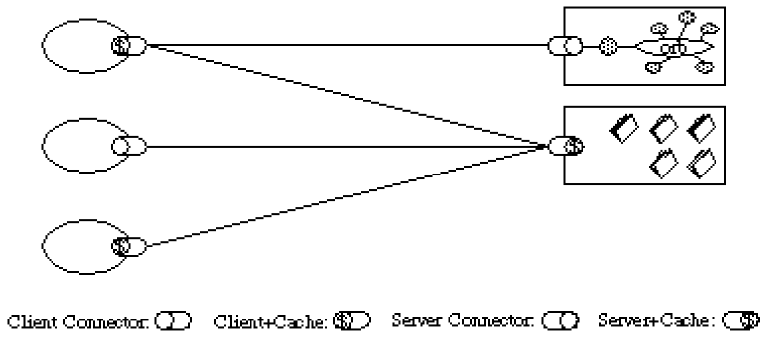


Figure 5-6. Uniform-Client-Cache-Stateless-Server

IMAGE 15 – Uniform Interface

5.3 Guiding principles of the interface

[4.3] Systems that follow the REST principles are often called RESTful. REST says that the web has enjoyed scalability as a result of a series of key fundamental designs:

A client/server protocol stateless: Each HTTP message contains all the information necessary to understand the request. As a result, neither the client nor the servers need to remember any communication state between messages. However, many applications based on HTTP cookies and other mechanism used to maintain session state.

A well-defined set of operations that apply to all information resources: HTTP itself defines a small set of operations; the most important are POST, GET, PUT and DELETE. Often these operations are equivalent to CRUD operations required for data persistence, but POST does not exactly fit in this scheme.

A universal syntax for identifying resources: In a REST, every resource is addressable only via its URI (Uniform Resource Identifier).

The use of hypermedia both for application information, like the state transitions of the application: The representations of this state in a REST system are typically HTML or XML. As a result, it is possible to navigate from one REST resource to many others, simply by following links without requiring the registers or other additional infrastructure.

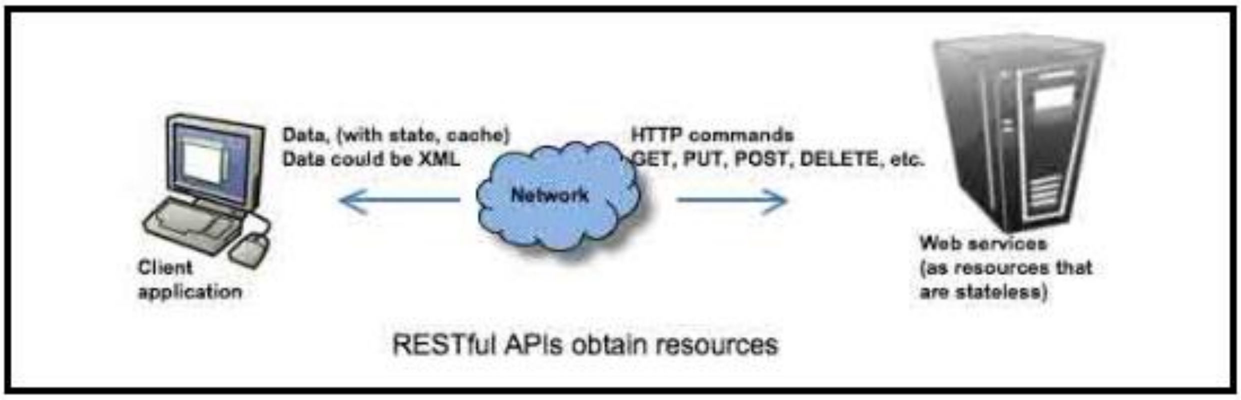

Data, (with state, cache)
Data could be XML
Network
HTTP commands
GET, PUT, POST, DELETE, etc.
Client application
Web services (as resources that are stateless)
RESTful APIs obtain resources

IMAGE 16 – Example of Computer sending Rest commands.

5.4 Key Goals

[4.1] Key goals of REST include:

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Intermediary components to reduce latency, enforce security and encapsulate legacy systems

REST has been applied to describe the desired web architecture, to help identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the Web successful.

5.5 REST Architectural Elements

[4.7] The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behaviour as a network-based application.

5.5.1 Data Elements

Unlike the distributed object style, where all data is encapsulated within and hidden by the processing components, the nature and state of architecture's data elements is a key aspect of REST. The rationale for this design can be seen in the nature of distributed hypermedia. When a link is selected, information needs to be moved from the location where it is stored to the location where it will be used by, in most cases, a human reader. This is unlike many other distributed processing paradigms, where it is possible, and usually more efficient, to move the "processing agent" (e.g., mobile code, stored procedure, search expression, etc.) to the data rather than move the data to the processor.

A distributed hypermedia architect has only three fundamental options: 1) render the data where it is located and send a fixed-format image to the recipient; 2) encapsulate the data with a rendering engine and send both to the recipient; or, 3) send the raw data to the recipient along with metadata that describes the data type, so that the recipient can choose their own rendering engine.

Each option has its advantages and disadvantages:

Option 1: the traditional client-server style, allows all information about the true nature of the data to remain hidden within the sender, preventing assumptions from being made about the data structure and making client implementation easier. However, it also severely restricts the functionality of the recipient and places most of the processing load on the sender, leading to scalability problems.

Option 2: the mobile object style, provides information hiding while enabling specialized processing of the data via its unique rendering engine, but limits the functionality of the recipient to what is anticipated within that engine and may vastly increase the amount of data transferred.

Option 3: allows the sender to remain simple and scalable while minimizing the bytes transferred, but loses the advantages of information hiding and requires that both sender and recipient understand the same data types.

REST provides a hybrid of all three options by focusing on a shared understanding of data types with metadata, but limiting the scope of what is revealed to a standardized interface. REST components communicate by transferring a representation of a resource in a format matching one of an evolving set of standard data types, selected dynamically based on the capabilities or desires of the recipient and the nature of the resource. Whether the representation is in the same format as the raw source, or is derived from the source, remains hidden behind the interface. The benefits of the mobile object style are approximated by sending a representation that consists of instructions in the standard data format of an encapsulated rendering engine. REST therefore gains the separation of concerns of the client-server style without the server scalability problem, allows information hiding through a generic interface to enable encapsulation and evolution of services, and provides for a diverse set of functionality through downloadable feature-engines.

REST DATA ELEMENTS	
DATE ELEMENT	**MODERN WEB EXAMPLES**
Resource	The intended conceptual target of a hypertext reference
Resource Identifier	URL, URN
Representation	HTML document, JPEG image
Representation Metadata	media type, last-modified time
Resource Metadata	source link, alternates, vary
Control Data	if-modified-since, cache-control

5.6 Central principle

[4.2] The motivation for REST was to capture the characteristics of the Web which made the Web successful. Subsequently these characteristics are being used to guide the evolution of the Web.

[4.1] An important concept in REST is the existence of resources (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP). In order to manipulate these resources, components of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange representations of these resources (the actual documents conveying the information). For example, a resource that represents a circle may accept and return a representation that specifies a centre point and radius, formatted in SVG (Scalable Vector Graphics), but may also accept and return a representation that specifies any three distinct points along the curve (since this also uniquely identifies a circle) as a comma-separated list.

CONNECTOR	MODERN WEB EXAMPLES
Client	libwww, libwww-perl
Server	libwww, Apache API, NSAPI
Cache	browser cache, Akamai cache network
Resolver	bind (DNS lookup library)
Tunnel	SOCKS, SSL after HTTP CONNECT

Any number of connectors (e.g., clients, servers, caches, tunnels, etc.) can mediate the request, but each does so without "seeing past" its own request (referred to as "layering," another constraint of REST and a common principle in many other parts of information and networking architecture). Thus, an application can interact with a resource by knowing two things: the identifier of the resource and the action required. It does not need to know whether there are caches, proxies, gateways, firewalls, tunnels, or anything else between it and the server actually holding the information. The application does, however, need to understand the format of the information (representation) returned, which is typically an HTML, XML or JSON document of some kind, although it may be an image, plain text, or any other content.

5.7 Advantages and Disadvantages

REST is a different style of designing web services. In a typical SOAP/WSDL service you expose to the network a small number of "service" objects (usually one), and each object can expose a large number of operations with custom names. A RESTful design is more data-oriented. The dataset is exposed to the network through a large number of standard HTTP objects called resources. Each resource supports a small, standardized set of operations (GET, PUT, DELETE...) that's always the same.

In a SOAP/WSDL service you devote your ingenuity to defining an API, a set of operations your service will expose. In a RESTful design you work on exposing your dataset through resources and connecting them together. You might think of a RESTful web service as a data structure, like a linked list, exposed to the network as a set of interlinked Web pages.

The most basic advantage of a RESTful design is that it works more or less like the World Wide Web. Most of the specific advantages follow from this. For instance, you can build a client for a service with nothing but an HTTP client library; there are lots of those for every language and they all work the same way. Every resource has its own URI, which means clients can pass around entry points into your service and combine your service with others. Resources can link to each other, which mean the service can guide a client through a complex multi-step operation. You can scale up your service using the standard tools developed for websites: caching proxies, HTTP load balancers, etc.

ADVANTAGES	DISADVANTAGES
Support for redirection, cache and different representations Easy to implement: no need for specific tools. Proven scalability Universal Holder and simply from any language and platform Real integration B2B communication Works with XML, but also with other formats.	We lose the strong typing. REST only Knows HTTP protocol

5.8 REST VS SOAP

[4.6] Both REST and SOAP are often termed "Web services," and one is often used in place of the other, but they are totally different approaches. REST is an architectural style for building client-server applications. SOAP is a protocol specification for exchanging data between two endpoints.

Comparing REST with the remote procedure call (RPC) style of building client-server applications would be more accurate. RPC is a style (rather than a protocol, which is what SOAP is) of building client-server applications in which a proxy (generally generated from metadata) is used in the client's address space to communicate with the server and the proxy's interface mimics the server's interface. Although SOAP doesn't require the RPC style, most modern SOAP toolkits are geared toward (at least they default to) using RPC.

Both REST and SOAP can be used to implement similar functionality, but in general SOAP should be used when a particular feature of SOAP is needed, and the advantages of REST make it generally the best option otherwise.

It is just as easy to make a RESTful service secure as it is to make a SOAP-based service secure. In the majority of cases involving either REST or SOAP, the security system is the same: some form of HTTP-based authentication plus Secure Sockets Layer (SSL). Although technically the technology for secure conversations over HTTP is now called Transport Layer Security (TLS), SSL is still the name most commonly used.

If you define interoperability as the technical ability to communicate between two divergent endpoints, I assert that REST wins the interoperability battle hands down.
The problem in the SOAP is the large number of different standards (and versions of each of those standards) to choose from. And when a particular vendor chooses to implement a particular standard, that vendor often provides an implementation that is just slightly different from another vendor's (or all others). This leads to problems whenever you have to cross vendor boundaries (languages and operating system).

The answer to "Which is better, REST or SOAP?" is "It depends." Both the REST architectural style and SOAP have advantages and disadvantages when it comes to building services. In the discussion between REST and SOAP, most Web application developers take very extreme positions in favour of one or the other. The followers of SOAP often say that SOAP is more flexible than REST when implementing Web Services and say that a defect in REST is the restriction “stateless”. While followers of REST (Restafarians) criticize the lack of transparency of SOAP and think that makes things more difficult than they really are. They say that SOAP provides a step forward and two backward. They also said that SOAP can be a problem because it can lead to the creation of security holes in HTTP implementations.

5.9 REST Command Format

The format of a WLP REST command is:

<protocol>://<host>:<port>/<webapp>/bea/wlp/api/<type>/<action>/<label>?<params>

Command Part	Description
<protocol>	The transport protocol, typically http.
<host>	The IP host name, such as localhost.
<port>	The port number, such as 7001
<webapp>	The name of the web application hosting the services, such as myWebApp. Note that all REST commands also require a webapp parameter. The webapp parameter specifies the specific web application for the command to operate on. The web application specified in the parameter can be different from the web application you post to, as long as it is deployed in the same EAR file.
bea/wlp/api	Standard namespace path used for all REST commands
<type>	The type of portal artifact the command operates on, such as desktop, book, page, portlet, lookandfeel, shell, menu, layout, them.
<action>	The specific action to take, such as list, delete, add, and get.
<label>	The unique label of the object.
command-specific parameters	A list of command-specific URL parameters. See the REST API reference documentation on e-docs for a complete list for each command. Commonly used parameters are described in Commonly Used REST Command Parameters.

Each REST command takes a list of parameters that are described in the REST API reference documentation on e-docs. This section describes three commonly used parameters in greater detail.

The webapp parameter is always required. It specifies the name of the web application on which you are calling the REST command. If you have more than one web application deployed in a given EAR, this parameter lets you easily switch between them. The web application specified with the parameter can be different web application you are posting to (that is, the application specified in the base URL).

For example, following command is which I use to make the call from one telephone to another:

PUT https://myUser:pass@localhost:PORT/sipxconfig/rest/call/number
PUT https://206:0000@test.local:8443/sipxconfig/rest/call/207

We have the commands GET, POST, PUT and DELETE. But we only need PUT for this project. If you are interested in read more about it: reference [4.1]

6. GROUPWARE

6.1 Definition

[5.1] Collaborative software (also referred to as groupware) is computer software designed to help people involved in a common task achieve goals. One of the earliest definitions of Groupware is: intentional group processes plus software to support them.

The design intent of groupware is to transform the way documents and rich media is shared to enable more effective team collaboration. Collaboration, with respect to information technology, seems to have several definitions. Some are defensible but others are so broad they lose meaningful application. Understanding the differences in human interactions is necessary to ensure that appropriate technologies are employed to meet interaction needs.

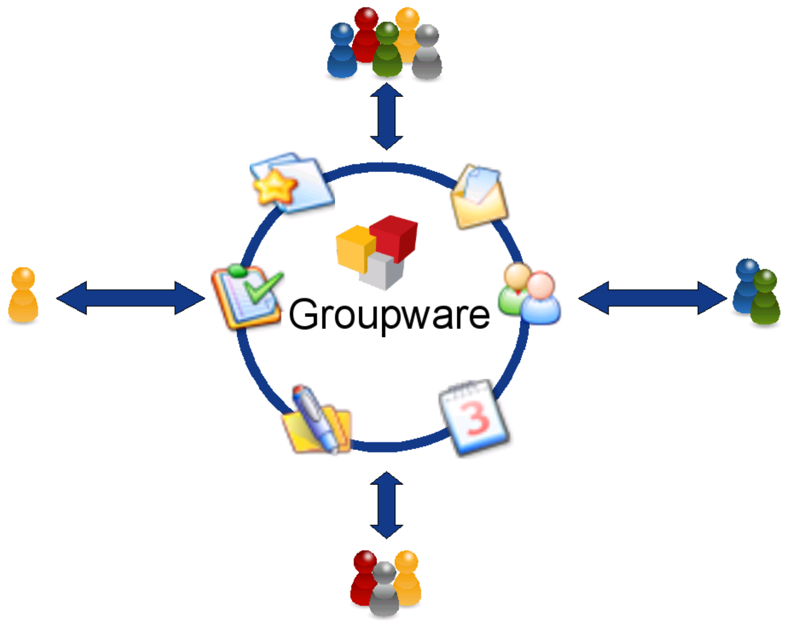


IMAGE 17 – Example of Shared Information in Groupware

Collaboration requires individuals working together in a coordinated fashion, towards a common goal. Accomplishing the goal is the primary purpose for bringing the team together. Collaborative software helps facilitate action-oriented teams working together over geographic distances by providing tools that aid communication, collaboration and the process of problem solving. Additionally, collaborative software may support project management functions, such as task assignments, time-managing deadlines, and shared calendars. The artefacts, the tangible evidence of the problem solving process, and the final outcome of the collaborative effort, require documentation and may involve archiving project plans, deadlines and deliverables.

We will work with Zarafa Collaboration Platform Groupware (ZCP)

6.2 Zarafa Collaboration Platform (ZCP)

[5.2] Zarafa (ZPC – Zarafa Collaboration Platform) is a full featured e-mail and groupware solution, focused towards clients using the MAPI standard. It's European open source collaborative software. The Zarafa groupware provides email storage on the server side and brings its own Ajax-based mail client called WebAccess. It's available only for Linux (not for Windows) and it's a high performance mail server with advanced features and free.

People who are accustomed to working with Outlook should be able to use without any problem WebAccess.

Features
Addressbook
Calendar
Notes
Tasks
Personal Folders/Public Outlook Folders
Permissions for every User and Folder configurable
Meeting invitation and free/busy option
Resources planning
POP3
IMAP
iCalendar
CalDAV
PDA synchronization via Z-push
BlackBerry integration over BES
Out-of-office message
Brick-level backup
Single sign-on
Hierarchical storage management (Zarafa Archiver)
Server-side message indexing

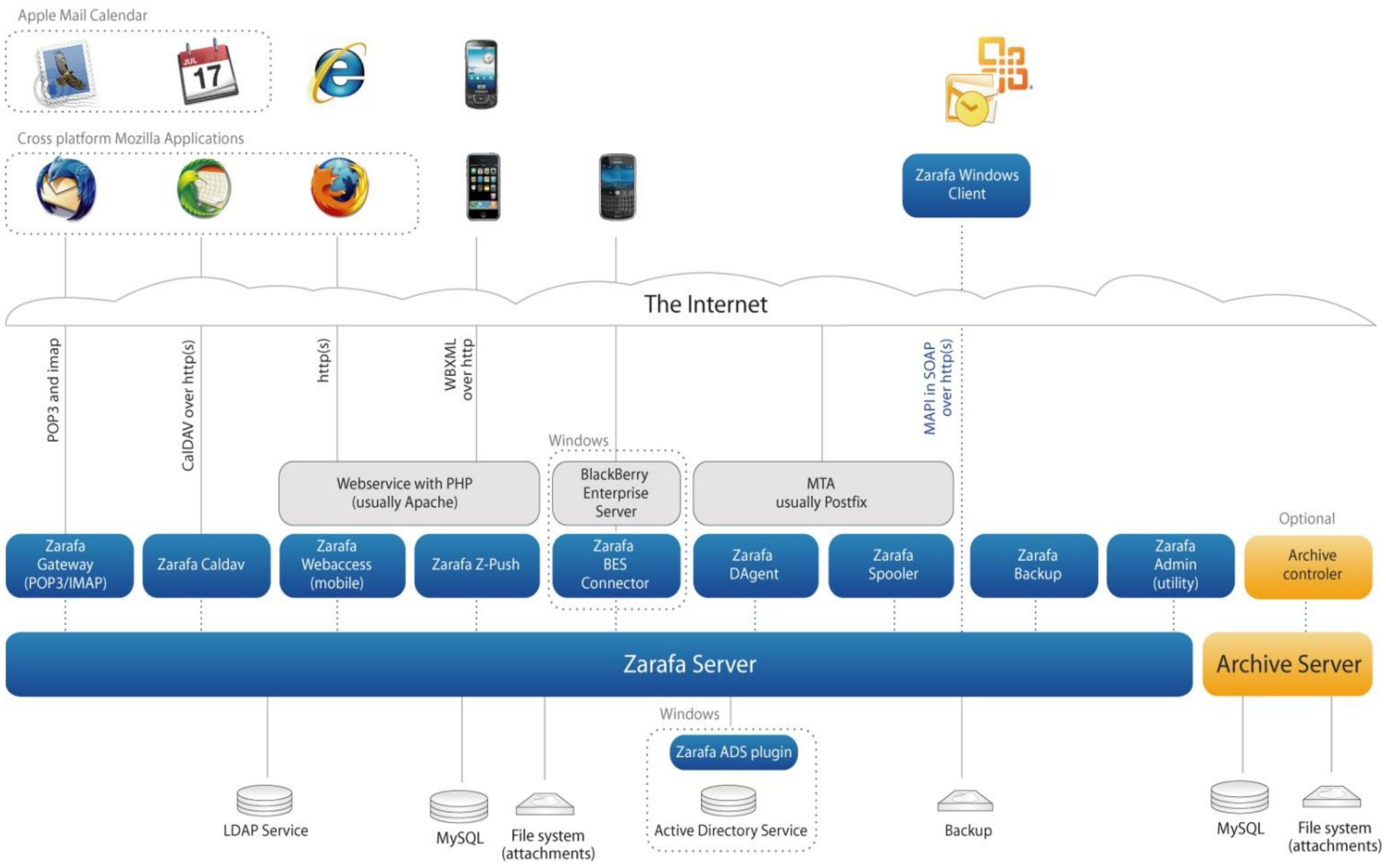


IMAGE 18 – Zarafa Architecture

Zarafa provides its groupware functionality by connecting the Linux-based server with Outlook clients using MAPI. The communication between server and client is based upon SOAP technology. The connection to Outlook clients can be secured using TLS/SSL, either directly between the Zarafa server program and the client, or via an HTTPS proxy. All data is generally stored in a MySQL database, although attachments can be saved on the file system. The Zarafa server can get its user information from LDAP, Active Directory, UNIX user accounts or the MySQL database. The webmail is based on AJAX technology, with a PHP backend using a MAPI PHP extension. Other clients can connect via POP3, IMAP and iCalendar/CalDAV.

Zarafa is part of the OpenMapi project which is developing an open groupware API based on MAPI. Its supported clients are:

Microsoft Outlook 2000-2010.
Web Access via Firefox or Internet Explorer 6, 7 and 8.
All ActiveSync compatible PDAs and smartphones via Z-push.
Blackberry devices via the Blackberry Enterprise Server.
POP3/IMAP4 email clients.
iCal/CalDAV calendar clients.

Zarafa is compatible with many other open source projects:

MTA: Postfix, Exim, qmail, Sendmail
Database: MySQL
Authentication: OpenLDAP, Kerberos, Cyrus SASL (via MySQL or rimap methods)
Web technologies: Apache, PHP.
Anti-Spam/Virus: ClamAV, SpamAssassin, AMaVIS/amavisd-new, DSPAM.

Zarafa developed and recently released an integration framework for 3rd party software. On the server side the Zarafa calendar, contacts, tasks and notes can be replicated in real time to other applications using the replication framework Z-Merge.

Zarafa server delivery MAPI calls, while storing data in a MySQL database. There are several different methods of user authentication. The most common are implementing LDAP servers.

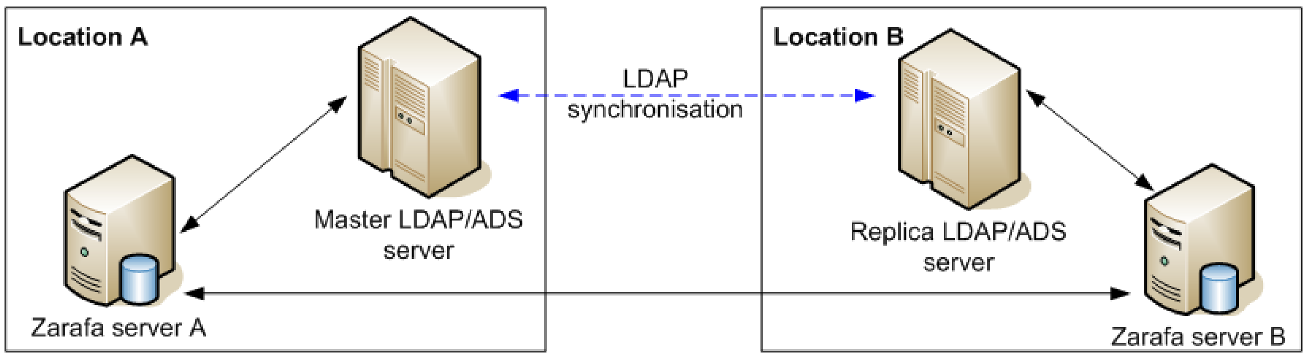


IMAGE 19 – Example of LDAP servers

6.2.1 Components

- **Zarafa Server (zarafa-server):** The server process accepts connections for all clients through SOAP (HTTP), and stores the data in an SQL database.

- **Zarafa License Manager (zarafa-licensed):** The licensed process check which features will be available dependent on the license chosen for the Community, Standard, Professional or Enterprise edition.

- **Zarafa Windows Client:** The Zarafa client provides access to Outlook through an interface known as MAPI. The connections with the server are handled by SOAP.

- **Zarafa WebAccess (zarafa-webaccess):** A full featured web interface (with an Outlook look and feel) that enables users to collaborate from any computer with an internet connection.

- **Zarafa Delivery Agent and Zarafa Spooler (zarafa-dagent, zarafa-spooler):** The tools which serve the email communication with the outside world. The dagent delivers mail from the Mail Transport Agent (MTA) to a Zarafa user. The spooler sends mail waiting in the outgoing queue to the specified MTA.

- **Zarafa Admin (zarafa-admin):** The command line administration tool is used to manage users, user information and groups.

- **Zarafa Gateway (zarafa-gateway):** Optional service to provide POP3 and IMAP access to Zarafa users.

- **Zarafa Monitor (zarafa-monitor):** Service which monitors user stores for quota exceeds.

- **Zarafa Caldav (zarafa-caldav):** Optional service that provides iCal and CalDAV support. CalDAV is recommended due to speed and less data transfer.

- **Zarafa Backup Tools (zarafa-backup, zarafa-restore):** A brick-level backup tools to create simple backups of stores and to restore (part of) those backups on a later point in time. This part is only available in Zarafa commercial editions.

- **Zarafa Indexer:** Optional service to provide full text indexing. This offers fast searching through email and attachments.

- **Apache:** Serves web pages of the WebAccess to the users browser.

- **PHP:** The WebAccess is written in this programming language.

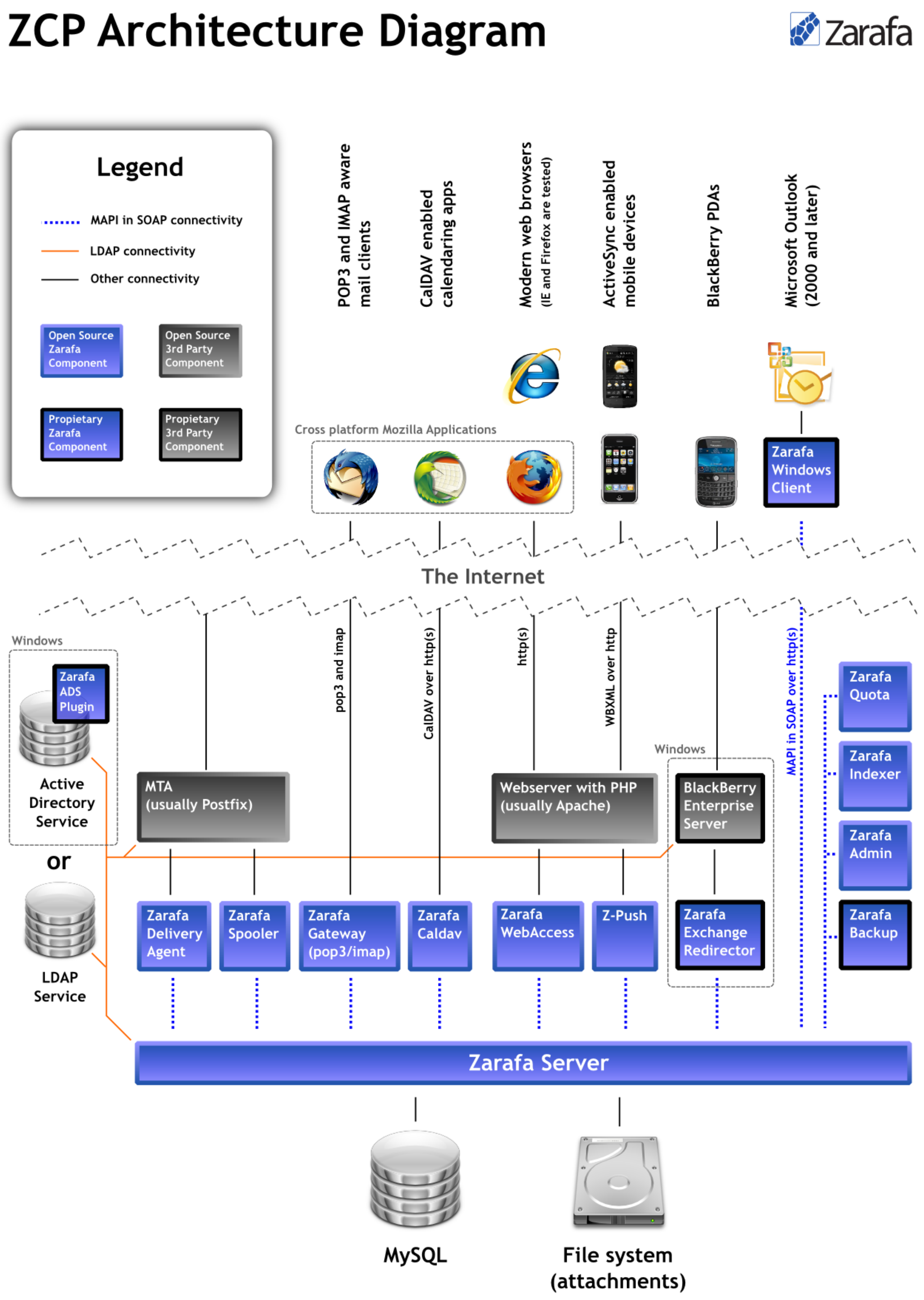
- **PHP-MAPI extension:** Module for PHP to enable use of the MAPI layer. Through this module, MAPI functions are made accessible for PHP developers. This effectively means that MAPI web clients can be written. The WebAccess is such a client.

- **Python-MAPI extension:** Module for Python to enable use of the MAPI layer. Through this module, MAPI functions are made accessible for Python developers.

Working with these components we can setup all the Zarafa parameters in our Linux SO. Most of these components have configuration files, that we can modify writing what we want to change. If you need more information about this, you can find in Zarafa Manual in the Chapter 7. [6.1]

6.2.2 Protocols and Connections

All applications which directly connect to the Zarafa Server use MAPI in SOAP to do so (see the Architecture Diagram). Even the WebAccess uses MAPI in SOAP (provided by the PHP-MAPI extension) to connect to the Zarafa Server. The Zarafa Windows Client is a standard Microsoft Windows compatible MAPI provider. It connects to the server (MAPI in SOAP) over the HTTP(S) protocol.



ZCP Architecture Diagram
Zarafa
Legend
MAPI in SOAP connectivity
LDAP connectivity
Other connectivity
Open Source Zarafa Component
Open Source 3rd Party Component
Propietary Zarafa Component
Propietary 3rd Party Component
POP3 and IMAP aware mail clients
CalDAV enabled calendaring apps
Modern web browsers (IE and Firefox are tested)
ActiveSync enabled mobile devices
BlackBerry PDAs
Microsoft Outlook (2000 and later)
Cross platform Mozilla Applications
Zarafa Windows Client
The Internet
Windows
Zarafa ADS Plugin
Active Directory Service
or
LDAP Service
MTA (usually Postfix)
pop3 and imap
CalDAV over http(s)
http(s)
WBXML over http
Webserver with PHP (usually Apache)
Windows
BlackBerry Enterprise Server
MAPI in SOAP over http(s)
Zarafa Quota
Zarafa Indexer
Zarafa Admin
Zarafa Backup
Zarafa Delivery Agent
Zarafa Spooler
Zarafa Gateway (pop3/imap)
Zarafa Caldav
Zarafa WebAccess
Z-Push
Zarafa Exchange Redirector
Zarafa Server
MySQL
File system (attachments)

IMAGE 20 – Zarafa Architecture

6.2.3 Secure HTTP (HTTPS)

The Zarafa Windows Client has the possibility to connect to the server over HTTP secured with SSL (HTTPS). When a MAPI profile from Outlook is created, it is possible to set the connection to use HTTPS. All connections over the network will then be encrypted, making eavesdropping virtually impossible. The Zarafa Server must be configured to also accept SSL connections. By default this is disabled, because it requires the creation of SSL certificates. When the server certificate is created, SSL connections can be directly accepted from a client. As an extra option other Zarafa components (like the Zarafa Delivery Agent and the Zarafa Spooler) can also connect over HTTPS to the server and authenticate using the Zarafa Server's private key.

7. INSTALLING OUR SYSTEM

7.1 First installation steps

1) Firstly I should download "Ubuntu 10.04 LTS 32 bit" in: http://www.ubuntu.com/download/ubuntu/download

2) After burn the image of Ubuntu, we install in a computer and we will have our Linux OS.

3) Now we can download Zarafa (ZCP). We must choose the right version compatible with our Linux OS. The file I used is this, but we can update after install:

   http://download.zarafa.com/community/final/7.0/7.0.1-28479/zcp-7.0.1-28479-ubuntu-10.04-i386-free.tar.gz

   We unzip this file in: home/daniel/Downloads/
   And we will have our files in the folder "zcp-7.0.2-29470-ubuntu-10.04-i386"

   We need some more software to run Zarafa: we need a database MySQL, Apache 2.0 /2.2 (or any server that supports PHP), SMTP server (I'm going to use Postfix)

4) We need to be "root user" for install the applications, so we must write the next commands:

   ```
   ~$ su
   ```
   Now we write our password (mine is "peinado") and we can work like root

   We install now MySQL database.
   ```
   ~$ apt-get install mysql-server
   ```

   We need to update to add the new files:
   ```
   ~$ apt-get update
   ```

5) We can install Zarafa from the file we have download or using apt-get:

   ```
   ~$ apt-get install zarafa zarafa-webaccess zarafa-libs zarafa-licensed
   ```

   or

   We go to the directory:

   ```
   cd home/daniel/Downloads/zcp-7.0.2-29470-ubuntu-10.04-i386
   ```

   We have some files in this directory, and we write the next command for install:

   ```
   ./install.sh
   ```

   It will start to install Zarafa and we must complete some options:

   - The serial number should be empty.
   - The password of MySQL would be the same: "peinado"
   - I continue with the next steps of the installation and I set the default options.
   - Now ask if we want to start zarafa-server and we say "yes". When we start our machine, zarafa-server will start.

   We have to write the MySQL password for Zarafa runs in the etc/zarafa/server.cfg

   ```
   ~$ sudo vi /etc/zarafa/server.cfg
   ```

   To check that apache works successfully write in the address bar: 127.0.0.1 and you should see in your browser "It **works!"**

6) The next step is install Postfix:

   ```
   ~$ apt-get install postfix
   ```

   And we have to add the next code to etc/postfix/main.cf :

```
mailbox_command = /usr/bin/zarafa-dagent "$USER"
mailbox_transport = zarafa: zarafa_destination_recipient_limit = 1
```

   And we have to add the next code to etc/postfix/master.cf :

```
zarafa    unix  -       n       n       -       10      pipe
  flags=DRhu user=vmail argv=/usr/bin/zarafa-dagent -R ${recipient}
```

7) We need to install the php5-curl to enable curl because we will use for the REST commands:

   ```
   $ apt-get install php5-curl
   ```

8) The last step is restart the application and create users for test:

```
$ sudo /etc/init.d/apache2 restart
$ sudo /etc/init.d/postfix restart
$ sudo /etc/init.d/zarafa-server start
```

And now I'm going to create two users for test:

```
zarafa-admin –c 207 –p test –f 'user1' –e 207@sipxecs.example.com
zarafa-admin –c 206 –p test –f 'user2' –e 206@sipxecs.example.com
```

I have 2 new users:

Name User1: 207	Pass: test	Mail: 207@sipxecs.example.com
Name User2: 206	Pass: test	Mail: 206@sipxecs.example.com

9) Finally, we have installed our Zarafa Groupware.

IMPORTANT: YOU MUST REGISTER YOUR USER WITH THE SAME NAME AS USER TELEPHONE. FOR EXAMPLE, IF YOUR TELEPHONE NUMBER IS 12345678, YOUR USER MUST BE 12345678. IF NOT, THE CALL DOESN'T WORK.

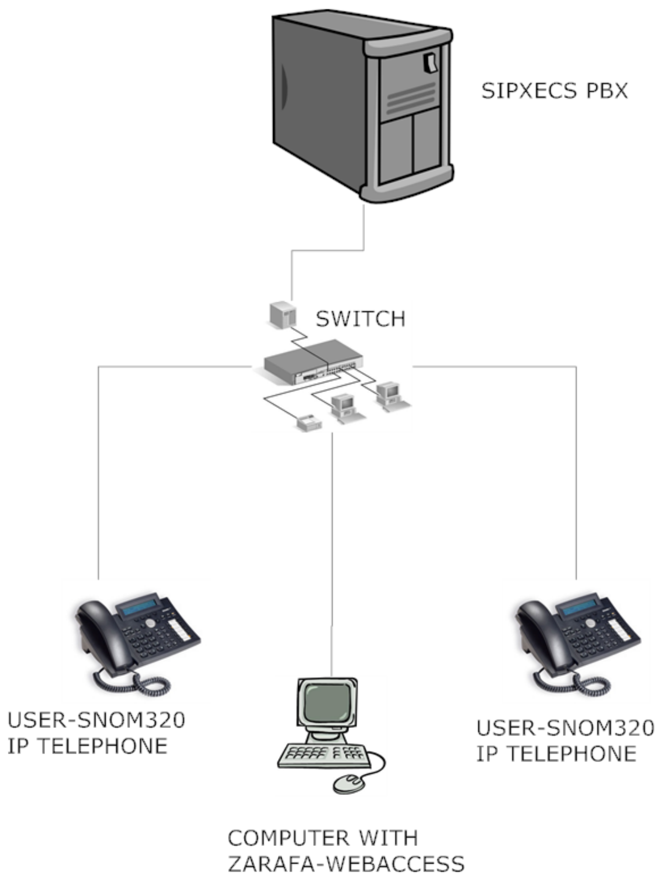


IMAGE 21 – Connections of my hardware to make the CTI example

This is the outline of how to connect the hardware.

7.2 SIPXECS CONFIGURATION [6.1]

Some of the parameters we have to configure in SipXecs are:

- **User name:** This is extension the user will use. Extensions typically start at 200 and go up from there.

- **Voice-mail PIN**: This is the password for the user to go into his or her voicemail. It also is the password for entry into the user portal.

- **SIP password:** The SIP password is the password the phone will use to register. It should be complex and must consist of at least eight letters (upper and lower case) and numbers.

- **User alias:** The user alias is something other than a number that can be used to dial the user. Consider adding their email alias here (if the user's email address is tuser@company.com, his or her alias would be tuser). This will allow the user to be dialled across the Internet from a softphone.

- **Email Address:** The user's email address can be used to forward his or her voicemail.

- **User group:** Define groups of users to better manage settings for multiple users. This field can be left blank if you are not sure about what groups you might use. User groups define permissions for users such as Superadmin Access, Change PIN from IVR, Configure Personal Auto Attendant, 900 Dialling, International Dialling, Local Dialling, Long Distance Dialling, Toll Free Dialling, Voicemail, Record System Prompts, and Internal Voicemail Server, or Microsoft Exchange UM Voicemail Server.

- **Phone serial number:** This will be the Ethernet MAC address of the physical phone that the user will have. If the user has a softphone, this will be left blank.

- **Phone model:** This will be the model of the managed IP phone that the user will have. If this phone will not be managed (that is, a softphone or not in the list of supported managed phones), this field should be left blank. If you know the web interface has support for the phone model you need, try to extrapolate the phone model from this list (for example, polycom9000). These are some of the phones that are managed as of this writing: polycom300, polycom430, polycom500, polycom550, polycom650, polycom600, polycom4000, cisco7960, cisco7940, cisco18x, cisco7905, cisco7912, gsPhoneBt, gsPhoneGxp, gsPhoneGxv3000, gsHt286, gsHt386, gsHt486, gsHt488, gsHt496, snom300, snom320, and snom360.

- **Phone group:** Just as with users, phones can be grouped and managed more easily. Consider groups of phones by department or by type of phone (for example, Poly330 or FrontOffice).

8. WEBACCESS AND PLUGIN ARCHITECTURE

8.1 Introduction

[6.2] With Zarafa Webaccess, you can access Zarafa via your internet browser. The Zarafa Webaccess contains features such as:

Easy email sharing

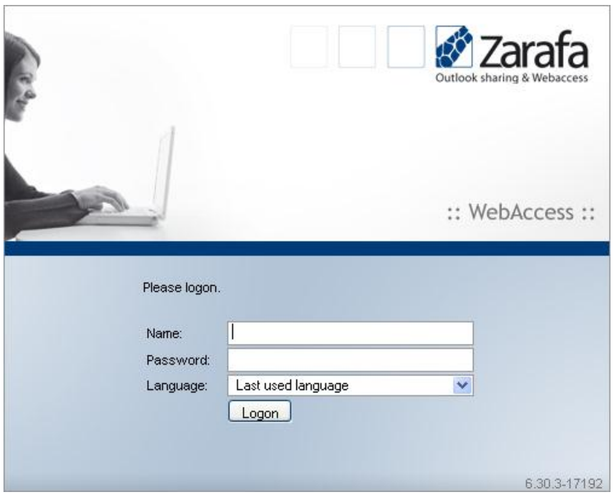
- drag and drop attachments
- public folders
- possibility of opening another user's calendar or mailbox

User-friendly

- multiple browsers: Firefox and IE
- free/busy and resource scheduling
- multi-user weekly calendar

Advanced search for any item

- Lucene indexing integrated
- advanced find search tool



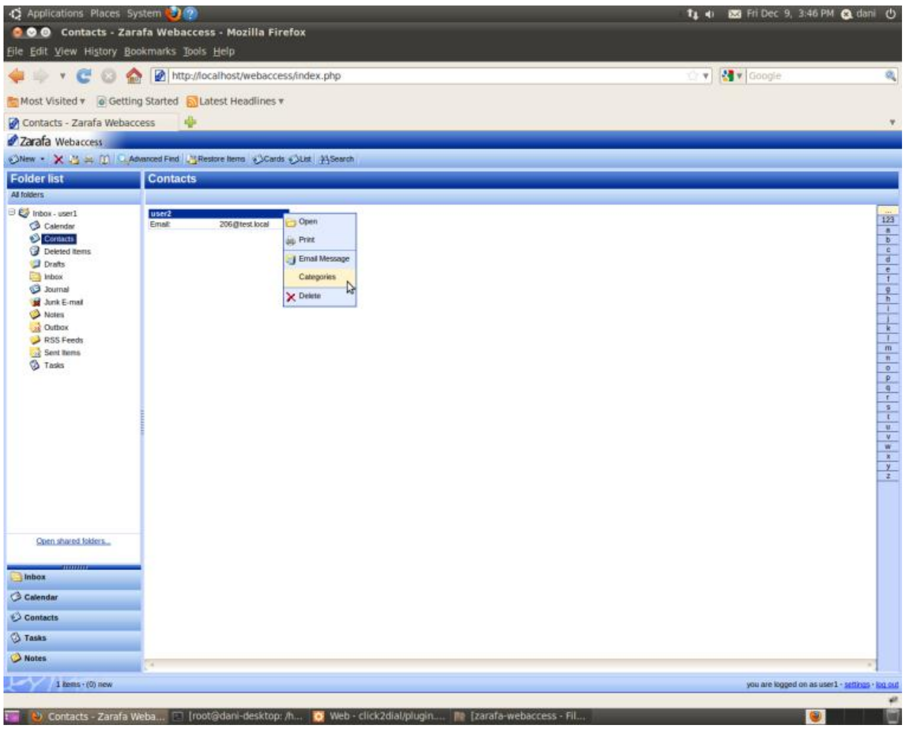


IMAGE 22 – Screenshot of Zarafa Webaccess

8.2 PLUGIN ARCHITECTURE

The plugin architecture allows the development of custom features, integration of third party applications and community cooperation in the development of new functionalities. The server administrator is able to add plugins to the Webaccess by simply adding the placing the plugin files in the plugins directory of the Webaccess. The Webaccess automatically detects the installed plugins that are placed in that directory.

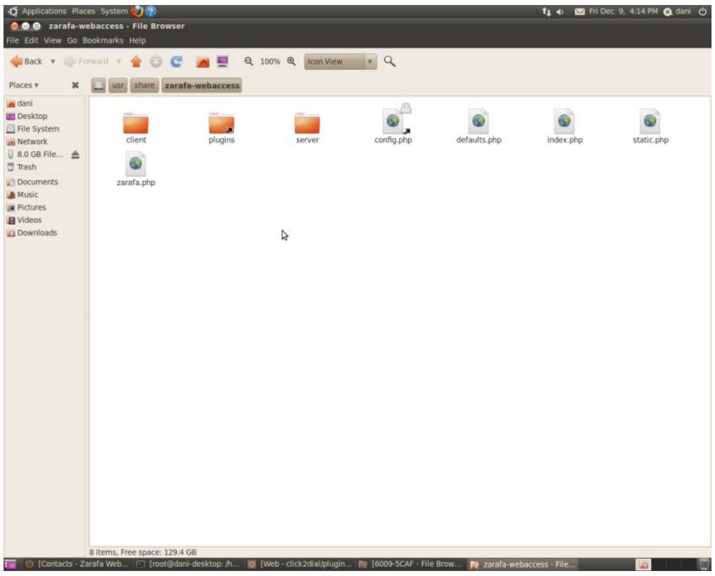


IMAGE 23 – Screenshot of Plugins Directory

8.2.1 Plugin anatomy

A plugin can interact with the Webaccess in a couple of different ways:

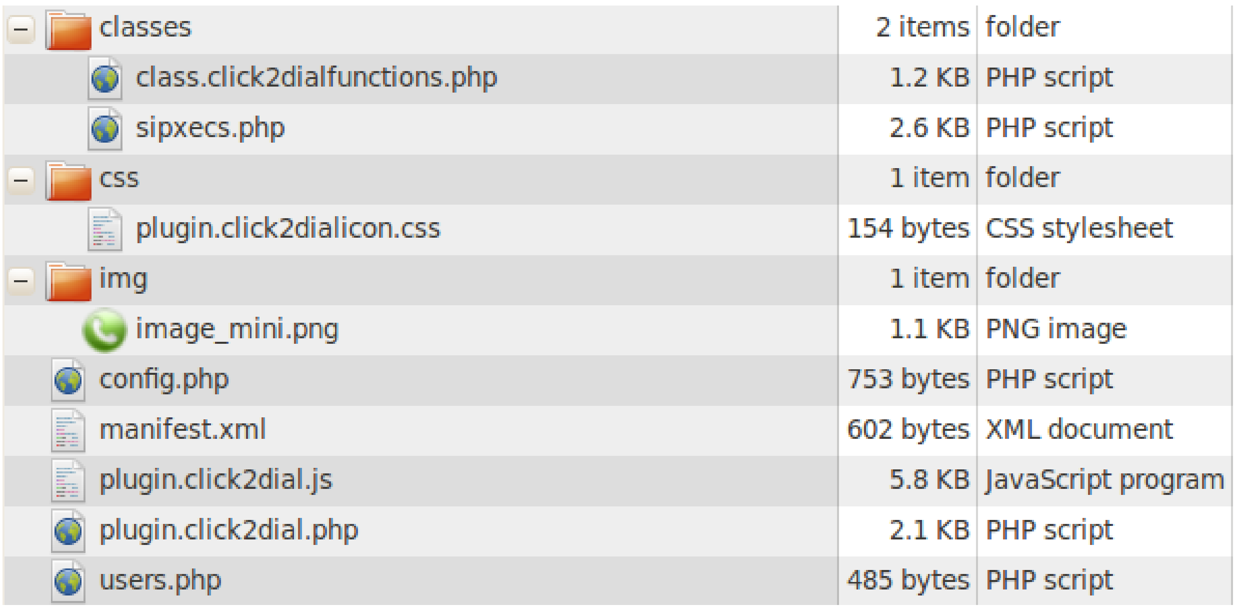
Adding new components: A plugin can create their own modules, views, widgets and sets of CSS rules and add them to the Webaccess through a simple plugin. This allows developer to create their own full integration with third party applications and create new interfaces to combine information stored in their account.

Extending existing components (client-side only): A plugin can extend existing components and add or change their functionalities. You can, for example, customize the operations of a module to rewrite the request to the server when retrieving the calendar items or you can change the way the drag and drop functions.

Hooks: A plugin can hook in at certain points in the code to stop the native Webaccess. The plugin then has the opportunity to modify objects and data before the native code finishes. He can detect addresses in the text and add an overlay contain a Google Maps image displaying the location. Adding their own items to menus and adding extra tabs in dialogs are also in the realm of possibilities. I have added in the CD, the pdf document with all predefined hooks. You can implement your own hooks, but I don’t need for this project. In the same document you can see the procedure to implement an own hook.

8.2.2 Plugin structure

In the root of the webaccess folder there is a plugin directory. Inside that directory each plugin has its own folder. The plugin is defined in the manifest.xml file that contains the configuration of the plugin in XMLformat. This file is stored in the folder of the plugin. This manifest contains information about descriptive information like name of the author, name of the plugin and a description. It also contains information about what files the Webaccess should include. Both on the server side and on the client side. Each component that you want to add is listed here. In order to place hooks you will also need to define a plugin class. In this class the developer can register to any hook in the system. When this hook is triggered the plugin class is notified and the developer will be able to create the features he wishes to have at that point in the Webaccess code. For the client side and the server side a separate class will need to be defined. One in PHP and the other one in JavaScript. To implement new functionalities it is also possible to add complete new components as modules, views, dialogs, CSS files, widgets, etc. These components will have all access and permissions the native Webaccess components have. You can also define your own translations for your plugin. In your XML manifest you simply define the folder that contains all the different translations and the Webaccess automatically includes the selected language.



– classes	2 items	folder
class.click2dialfunctions.php	1.2 KB	PHP script
sipxecs.php	2.6 KB	PHP script
– css	1 item	folder
plugin.click2dialicon.css	154 bytes	CSS stylesheet
– img	1 item	folder
image_mini.png	1.1 KB	PNG image
config.php	753 bytes	PHP script
manifest.xml	602 bytes	XML document
plugin.click2dial.js	5.8 KB	JavaScript program
plugin.click2dial.php	2.1 KB	PHP script
users.php	485 bytes	PHP script

IMAGE 24 – Files Structure of my Software

Here is the structure of my plugin, with all files I have programmed.

9. Development of my plugin

9.1 Configuration

The configuration file “config.php” can define three options for the Plugin System.

Configuration key	Default value	Description
PATH_PLUGIN_DIR	“plugins”	Defines the path to the directory that contains all the plugins.
ENABLE_PLUGINS	true	Enables or disables all the plugins.
DISABLED_PLUGINS_LIST	(empty string)	A semicolon separated list of the names of the plugins that will be disabled.

And here is one example of the code in the file:

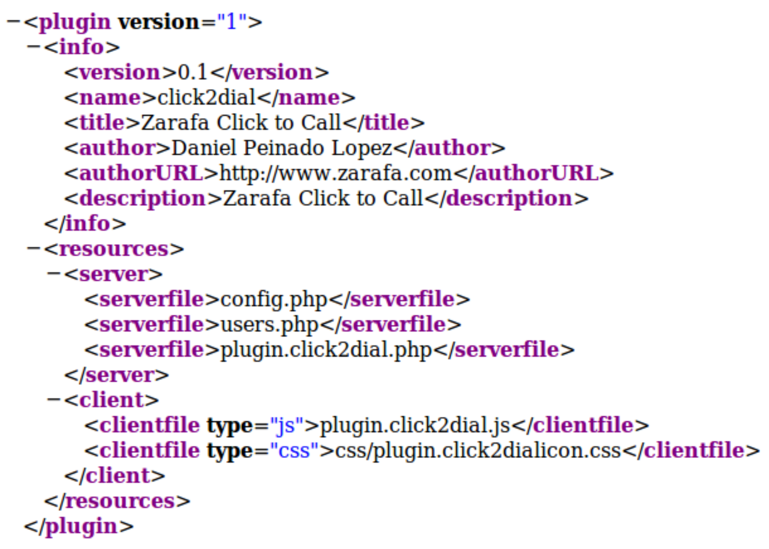
```
// Define the path to the plugin directory (No slash at the end)
define("PATH_PLUGIN_DIR", "plugins");

// Define the path to the plugin directory
define("ENABLE_PLUGINS", true);

// Define list of disabled plugins separated by semicolon
define("DISABLED_PLUGINS_LIST", 'gmap;spellchecker');
```

9.2 Manifest

The manifest defines the contours of my plugin. It defines besides the some descriptive information about the plugin, the files and the modules that need to be included by the Webaccess. This is my own manifest of my plugin:



```
−<plugin version="1">
  −<info>
     <version>0.1</version>
     <name>click2dial</name>
     <title>Zarafa Click to Call</title>
     <author>Daniel Peinado Lopez</author>
     <authorURL>http://www.zarafa.com</authorURL>
     <description>Zarafa Click to Call</description>
   </info>
  −<resources>
    −<server>
       <serverfile>config.php</serverfile>
       <serverfile>users.php</serverfile>
       <serverfile>plugin.click2dial.php</serverfile>
     </server>
    −<client>
       <clientfile type="js">plugin.click2dial.js</clientfile>
       <clientfile type="css">css/plugin.click2dialicon.css</clientfile>
     </client>
   </resources>
 </plugin>
```

The info part, describes some basic information. The resource part describes what resources the plugin uses.

The client portion contains CLIENTFILE-nodes. Each node links to a file that must be included on the client-side of the Webaccess. There are few file types that can be included in this way. The code example above indicates what type of files can be included. Every CLIENTFILE-node only contains the filename (and path to that file) except for modules and CSS-files. The two exceptions require a type attribute. The Webaccess requires modules to be specifically identified as modules and CSS files need to be included in a different way than JavaScript-files.

File type	XML Node	Attribute(s)	Value
JS Plugin class	clientfile	None	Path to file
JS Module	clientfile	type=module	Path to file
JS Widget	clientfile	None	Path to file
JS View	clientfile	None	Path to file
Other JS file	clientfile	None	Path to file
CSS file	clientfile	type=css	Path to file

It is also possible to say that a file must only be loaded in the main window or only in the dialogs. You can do this by specifying the load attribute with either the value “main” (load only in main window) or the value “dialog” (load only in dialogs). If you do not specify the attribute the file will be loaded in the both the main window as in the dialogs.

Load in	XML Node	Attribute(s)	Value
Main window	clientfile	load=main	Path to file
All dialogs	clientfile	load=dialog	Path to file
Main and all dialogs	clientfile	None	Path to file

The server portion contains SERVERFILE-nodes. Each node links to a file that must be included on the server-side of the Webaccess. The code above shows three file types. The first one is a plugin class file. This file is included without any specific attributes. The second type is a module file that needs to attributes: type and module. The type attribute identifies the file as a module and the module-attribute defines the class name of the module. As mentioned before the modules need to be identified as such and that is why module-files need these extra attributes. The third type is a PHP file that you want included in the server-side Webaccess. This file can contain another PHP class, extra functions or another PHP library for example.

File type	XML Node	Attribute(s)	Value
PHP Plugin class	serverfile	None	Path to file

PHP Module	serverfile	type=module
		module=CLASSNAME
			Path to file

Other PHP file	serverfile	None	Path to file

The Dialogs and Translations I don’t need to explain for this project.

9.3 Plugin on the Client-Side

The plugin developer can add completely new components (like modules, views, widgets, etc.) to the Webaccess. These components will have the same access and permissions as the native Webaccess components have. The syntax and structure of these components does not differ from native components. As mentioned before it is important to have the modules identified in the manifest XML. When extending existing components you can add a normal JS file where you extend the existing JS objects through prototyping. You can also add your own libraries through this way. The added components are only used when the Webaccess directly instantiates the component. A module, view or widget has to specifically be called before it will get executed. The developer can also define a plugin class to let the plugin interact directly with the Webaccess. When the plugin class is initialized it has the ability to register to hooks. When the Webaccess reaches a point in the code where a hook has been placed all plugins that have registered to that hook will be notified and will have the opportunity to execute their code before the native code will continue.

The plugin class on the client-side is a prototype JavaScript object/class. The class is always instantiated when it exists. It is possible to create a plugin that does not have a plugin class.

```
/**
 * Example plugin class
 */
Pluginexample.prototype = new Plugin;
Pluginexample.prototype.constructor = Pluginexample;
Pluginexample.superclass = Plugin.prototype;
function Pluginexample(){}
Pluginexample.prototype.init = function(){
      this.registerHook("main.hierarchymodule.sharedFoldersPane.buildup");
}
Pluginexample.prototype.execute = function(eventID, data){
      switch(eventID){
            case "main.hierarchymodule.sharedFoldersPane.buildup":
                  this.doExampleStuff(data);
                  break;
      }
}
Pluginexample.prototype.doExampleStuff = function(data){
      // Do stuff
}
```

The structure of such a class looks like the code shown above.

```
Pluginexample.prototype = new Plugin;
Pluginexample.prototype.constructor = Pluginexample;
Pluginexample.superclass = Plugin.prototype;
```

These lines are needed for all class components in the Webaccess. The plugin class is called “Pluginexample” and it extends the superclass “Plugin”.

```
function Pluginexample(){}
```

This is the constructor. You need use the constructor to register to hooks.

```
Pluginexample.prototype.init = function(){
      this.registerHook("main.hierarchymodule.sharedFoldersPane.buildup");
}
```

The init-function is called to initialize the plugin class. In this function you can start registering to hooks. This can be done by using the registerHook function that the plugin class inherits. Each hook is identified by a string. The hook that is placed above is identified by the string "main.hierarchymodule.sharedFoldersPane.buildup". This specific hook allows you to place links under the hierarchy list in the Webaccess. If you want to define more hooks you can call this method multiple times.
Here is where I have opened the context menu to add my button after.

```
Pluginexample.prototype.execute = function(eventID, data){
      switch(eventID){
             case "main.hierarchymodule.sharedFoldersPane.buildup":
                    this.doExampleStuff(data);
                    break;
      }
}
```

The execute function is called when the plugin class has been registered to one or more hooks and that hook is triggered by the native Webaccess code. The two arguments that are supplied are the identifier of the hook that is triggered and an object that contains the data that you can modify. The data object contains references to the data that the plugin is allowed to change. In the example of the hook "main.hierarchymodule.sharedFoldersPane.buildup" the data object will contain a reference to an HTML DOM element. You can use this reference to add your own HTML (read: links) to the pane below the hierarchy list. Another example can be that the data object contains a list of items that will be added to a context menu, which is exactly what I must do. It is best to create a switch statement in this execute function so you can create separate function for each hook. This is what is done in the code above. Note that in order to do that you have to pass on the data object to that separate function.

```
Pluginexample.prototype.doExampleStuff = function(data){
      // Do stuff
}
```

In the code above you can define all the actions that you want to do when the hook is triggered. This can mean adding items to menus or changing the output that will be displayed (so you can add mouse over hover functionality like Google Maps or linking telephone numbers to Skype when they are clicked on).

9.4 Plugin on the Server-Side

The plugin developer can add completely new components to the server-side as well. On the server-side this means that new modules can be added. These modules have the same access and permissions as other native Webaccess modules. The syntax and structure of these modules does not differ from native modules. As mentioned before it is important to have the modules identified as such in the manifest XML. When extending existing components you can add a normal PHP file where you extend the existing PHP classes. You can also add your own libraries through this way. As on the client-side, the server-side modules are only used when the Webaccess or other plugins directly instantiate the modules. They has to be specifically called before it will get executed. The developer can also define a plugin class on the server-side to let it interact directly with the Webaccess. When the plugin class is initialized it has the ability to register to hooks. When the Webaccess reaches a point in the code where a hook has been placed all plugins that have registered to that hook will be notified and will have the opportunity to execute their code before the native code will continue.

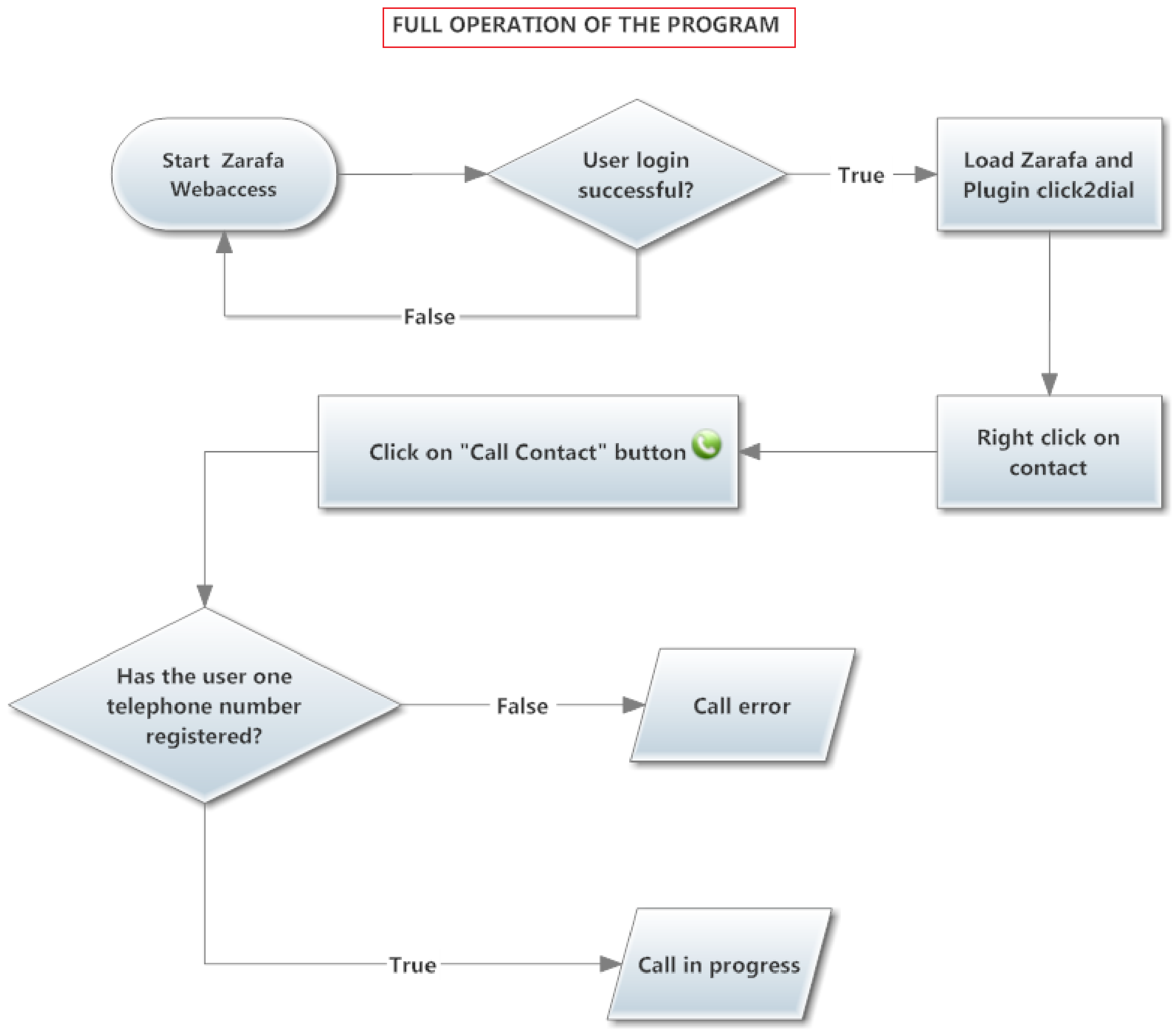
The plugin class is always instantiated when it exists. It is possible to create a plugin that does not have a plugin class.

```php
<?php
/**
 * Example Plugin class
 */
class Pluginexample extends Plugin {
      function Pluginexample(){}
      function init(){
            $this->registerHook('dialogs.general.buildMenu');
      }
      function execute($eventID, &$data){
            switch($eventID){
                  case 'dialogs.general.buildMenu':
                        $this->addDialogMenuItems($data);
                        break;
            }
      }
      function addDialogMenuItems(&$data){
            // Do stuff
      }
}
```

It’s almost the same like the JavaScript Plugin Class. We have the same functions but with php language. I’m not going to explain every function like in the case before.

9.5 FLOWCHARTS OF MY PLUGIN

9.5.1 General operation of the program


FULL OPERATION OF THE PROGRAM
Start Zarafa Webaccess
User login successful?
True
Load Zarafa and Plugin click2dial
False
Right click on contact
Click on "Call Contact" button
Has the user one telephone number registered?
False
Call error
True
Call in progress

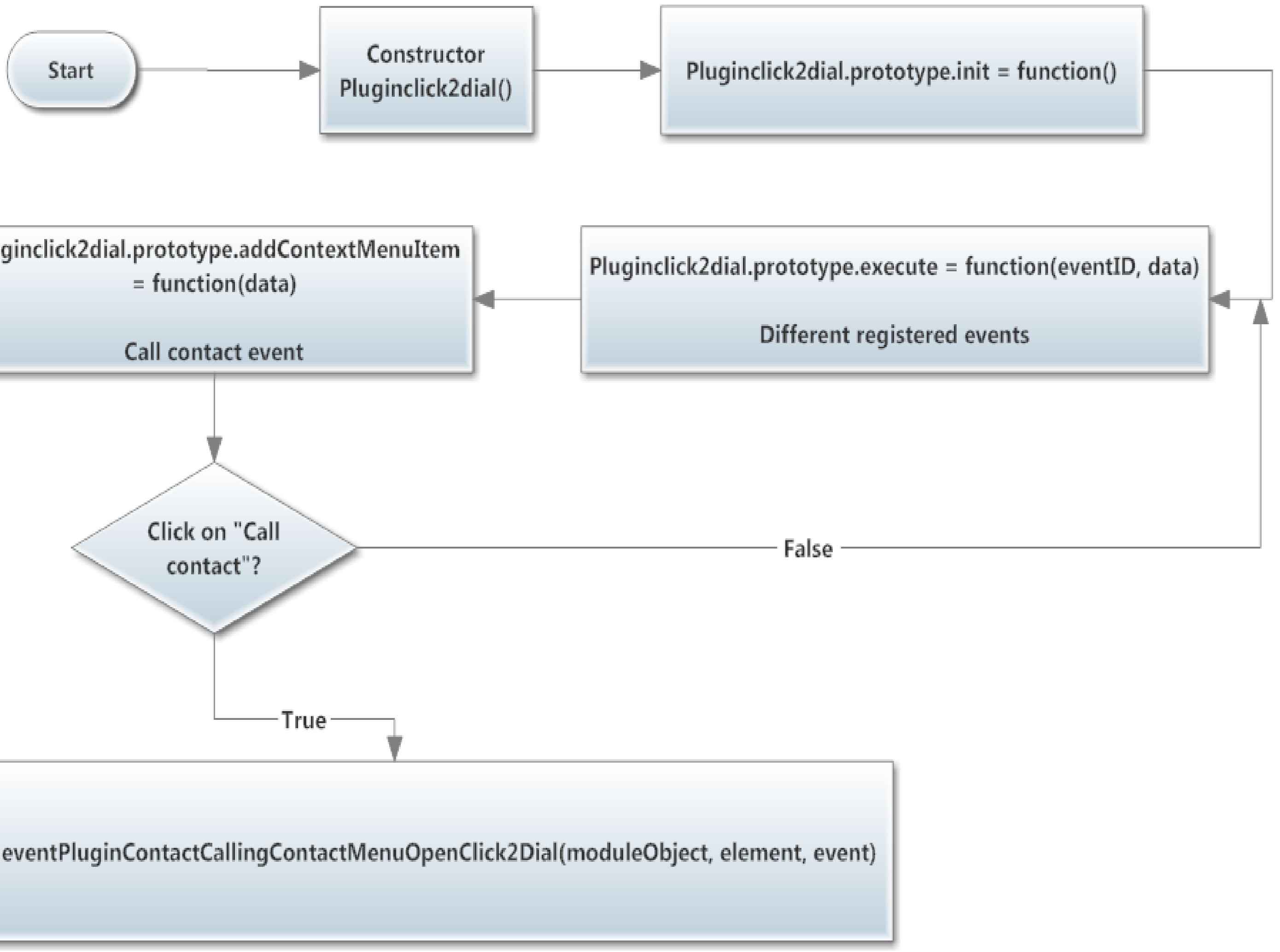
Here we have a picture for understand easily the running of the plugin.

We can see that we need to login successful. After we should make right click mouse on one of our contacts and press “Call contact” button. If we have telephones and telephones number installed correctly, we could call our contacts by one click.

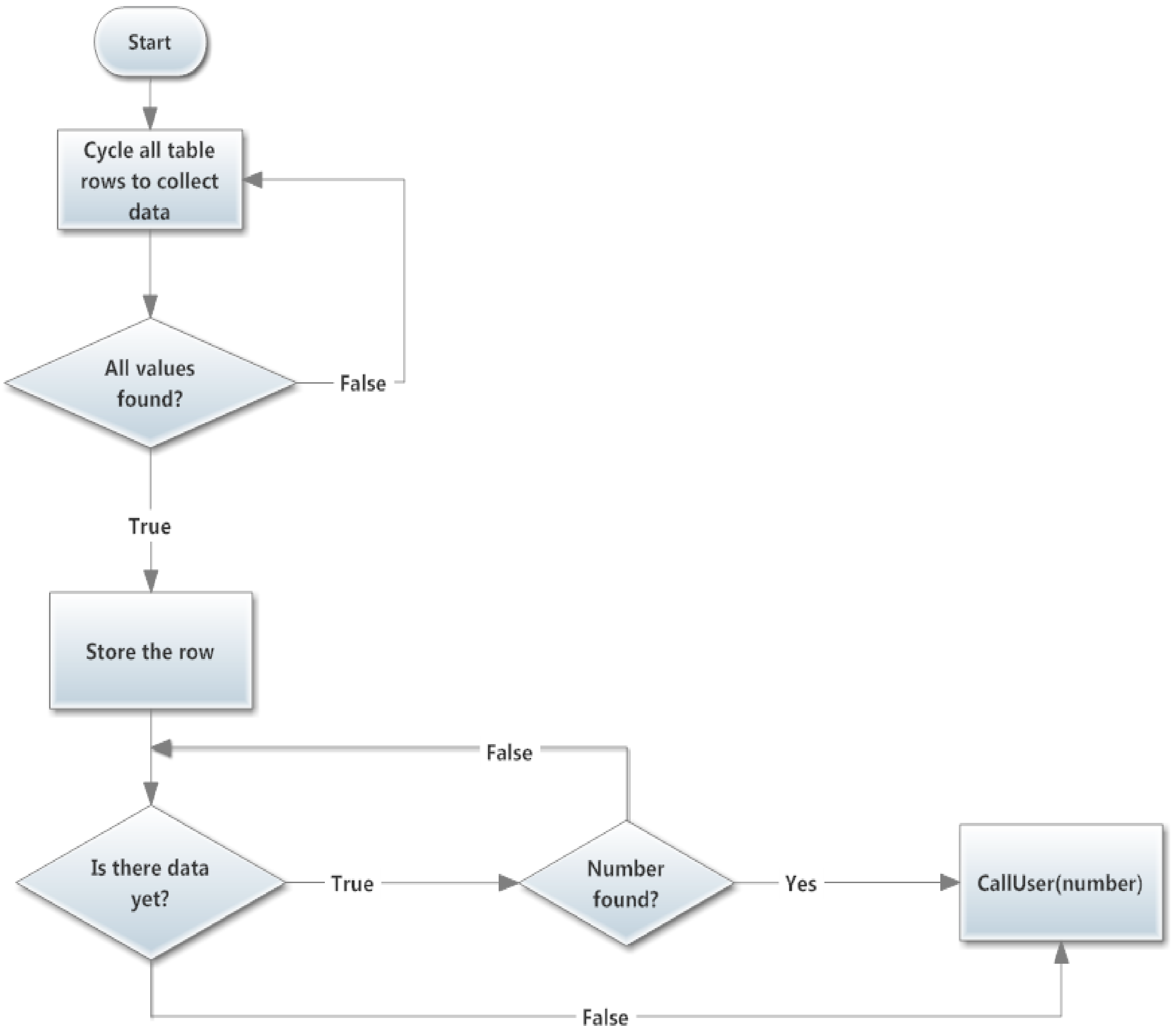
9.5.2 Flowcharts of classes and functions

This is the main class of the plugin in JavaScript in the client side. Here I have programmed how to launch the event from Zarafa Webaccess. You can see the program code in annex documentation.

File: plugin.click2dial.js

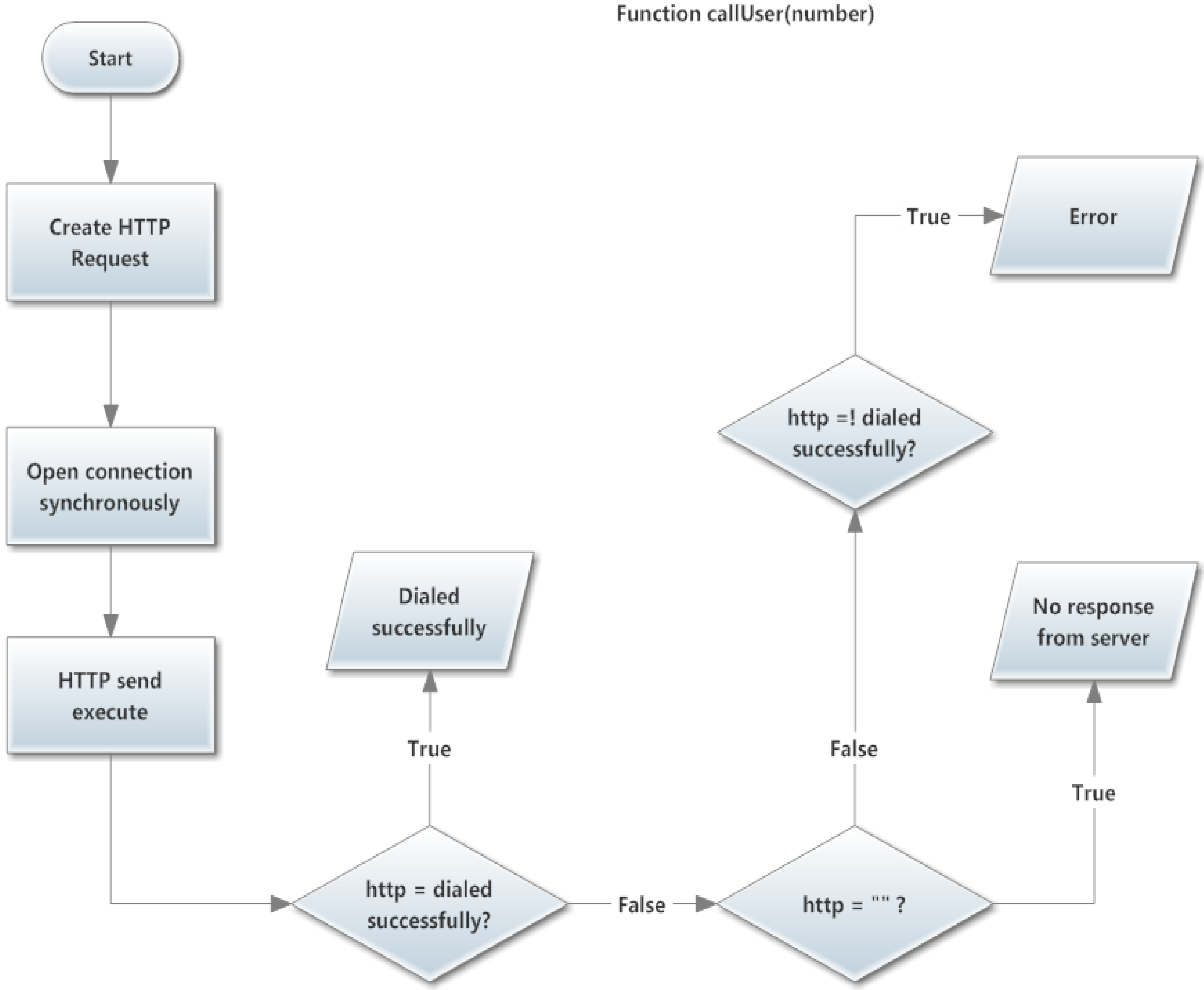

Start
Constructor Pluginclick2dial()
Pluginclick2dial.prototype.init = function()
Pluginclick2dial.prototype.execute = function(eventID, data)
Different registered events
Pluginclick2dial.prototype.addContextMenuItem = function(data)
Call contact event
Click on "Call contact"?
False
True
function eventPluginContactCallingContactMenuOpenClick2Dial(moduleObject, element, event)

File: plugin.click2dial.js
Function eventPluginContactCallingContactMenuOpenClick2Dial(moduleObject, element, event)

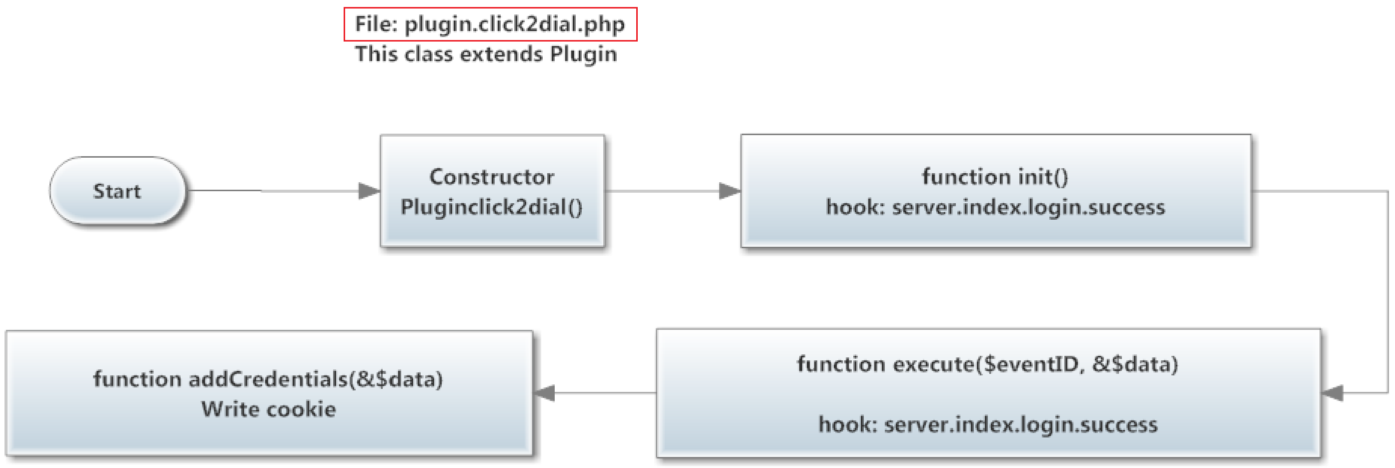


Here is the code of the JavaScript function that we launch when we click on the button. This is the event function.

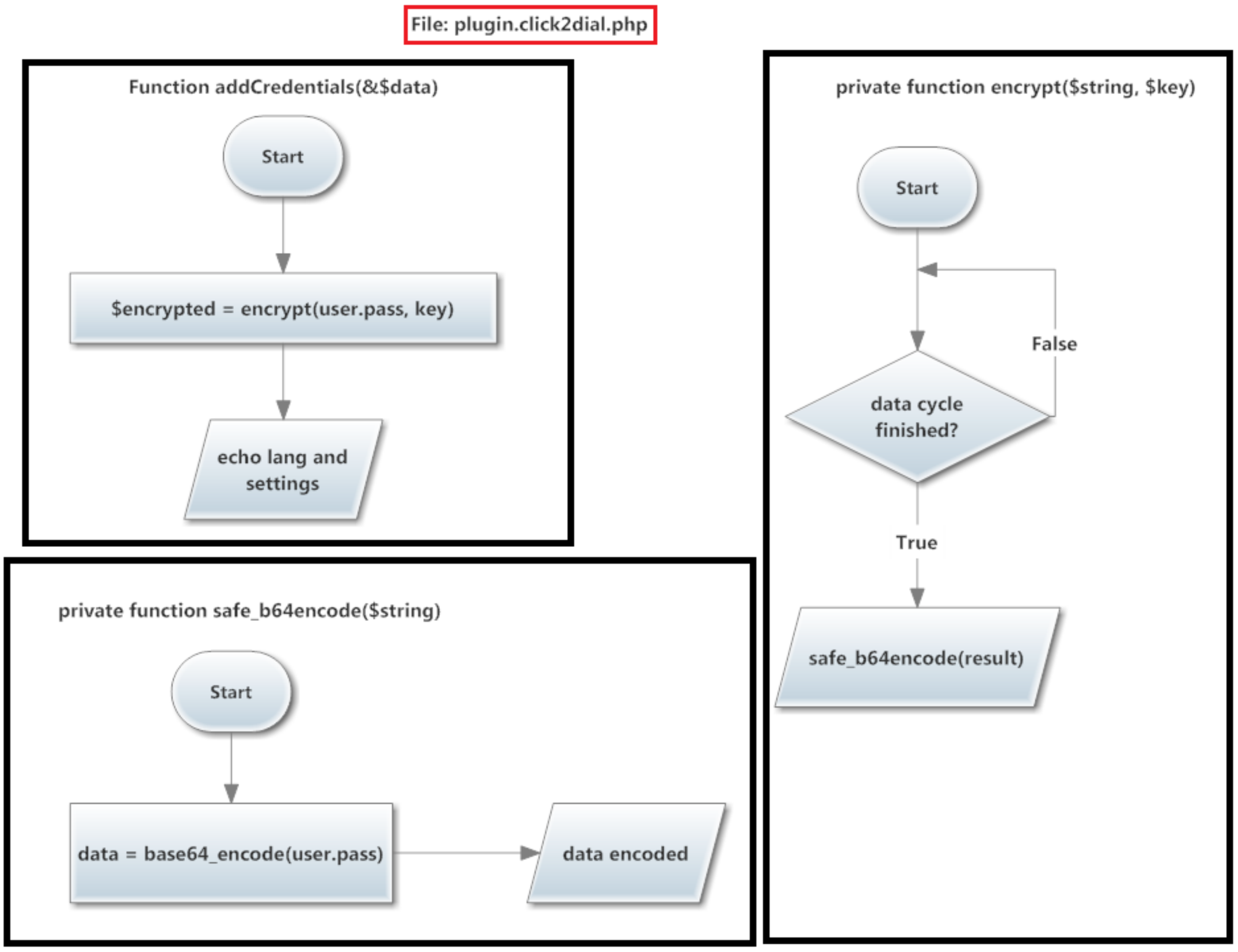
File: plugin.click2dial.js
Function callUser(number)

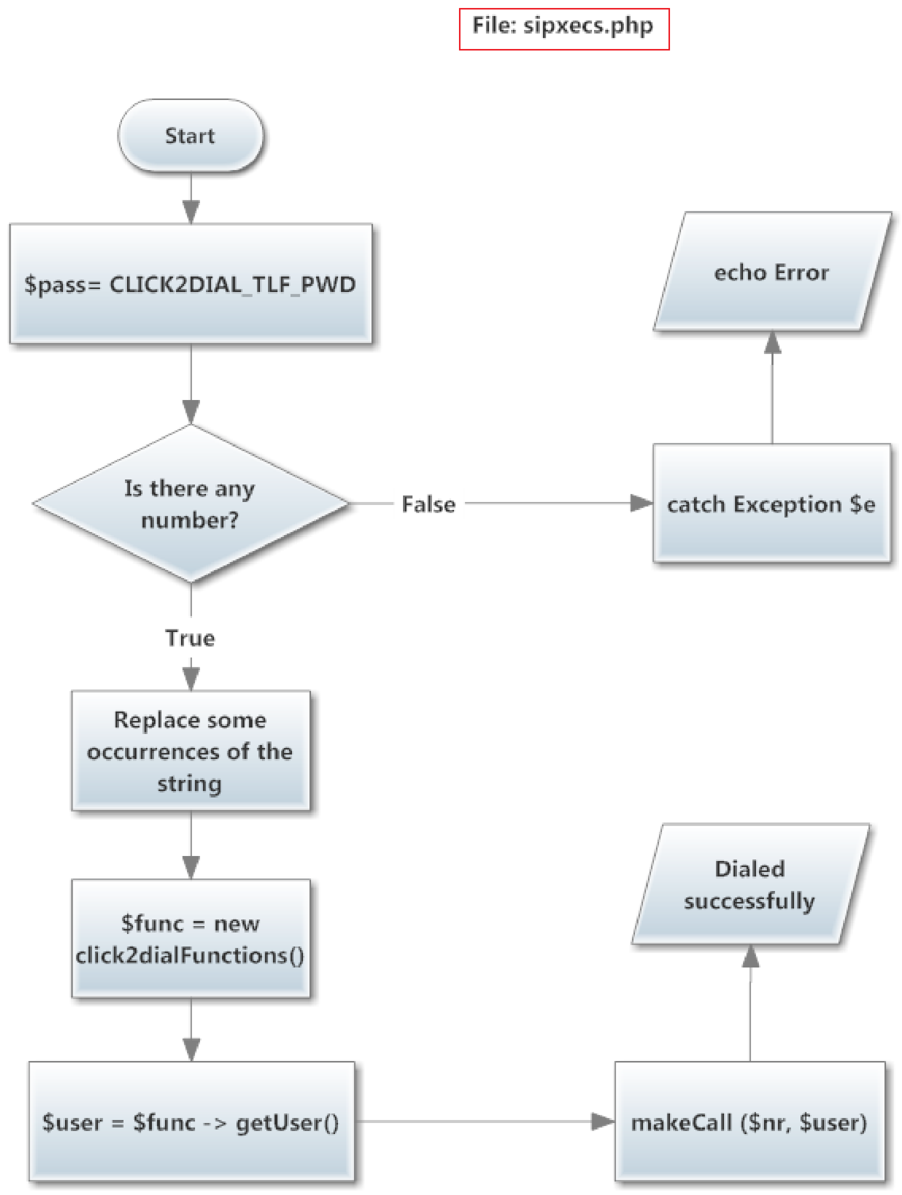


This function uses the number which gets from sipxecs.php class to make a response depending on the result of the call.


File: plugin.click2dial.php
This class extends Plugin
Start
Constructor Pluginclick2dial()
function init() hook: server.index.login.success
function execute($eventID, &$data) hook: server.index.login.success
function addCredentials(&$data) Write cookie

This plugin.click2dial.php file, is like the JavaScript file, only that in the server side. When we want to implement something in the server side we must follow the steps I explained previously (Plugin on the server-side). The hook I have used is launched when the user makes login successful and then he is redirected to Webaccess. Then the $_GET variable contains the key “logon”. And these functions are inside the plugin.click2dial.php file.

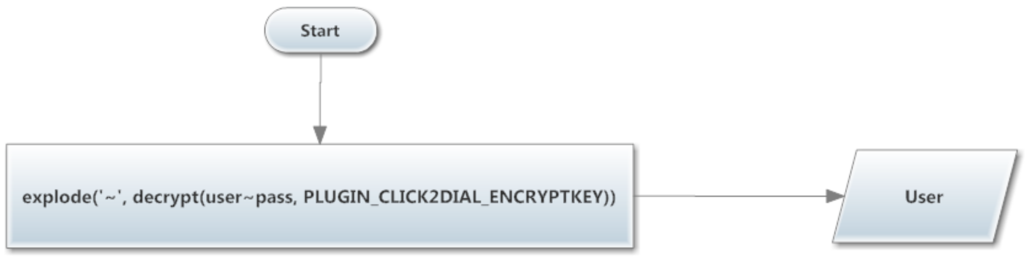

File: plugin.click2dial.php
Function addCredentials(&$data)
Start
$encrypted = encrypt(user.pass, key)
echo lang and settings
private function encrypt($string, $key)
Start
False
data cycle finished?
True
safe_b64encode(result)
private function safe_b64encode($string)
Start
data = base64_encode(user.pass)
data encoded


File: sipxecs.php
Start
$pass= CLICK2DIAL_TLF_PWD
Is there any number?
False
catch Exception $e
echo Error
True
Replace some occurrences of the string
$func = new click2dialFunctions()
$user = $func -> getUser()
makeCall ($nr, $user)
Dialed successfully

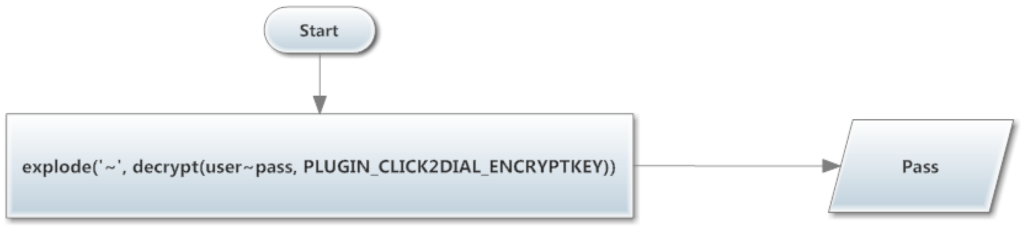
This file is program in PHP and it works in server side. This part of the code is actually which sends the REST command to the SipXecs server for make the call. The click2dialFunction() is a class in PHP which is used in decryption process and get the user name. I show in the next page the 4 functions inside this class.

File: class.click2dialfunctions.php

function getUser()

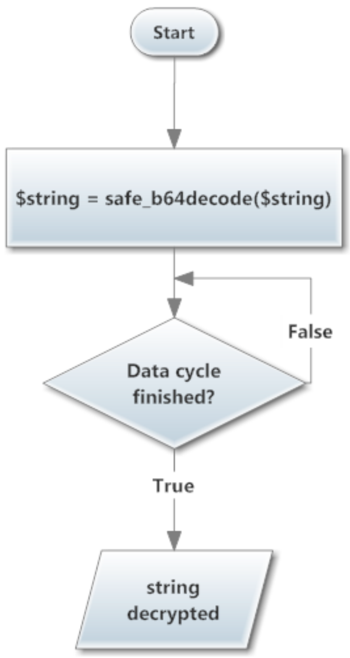

Start
explode('~', decrypt(user~pass, PLUGIN_CLICK2DIAL_ENCRYPTKEY))
User

function getPassword()

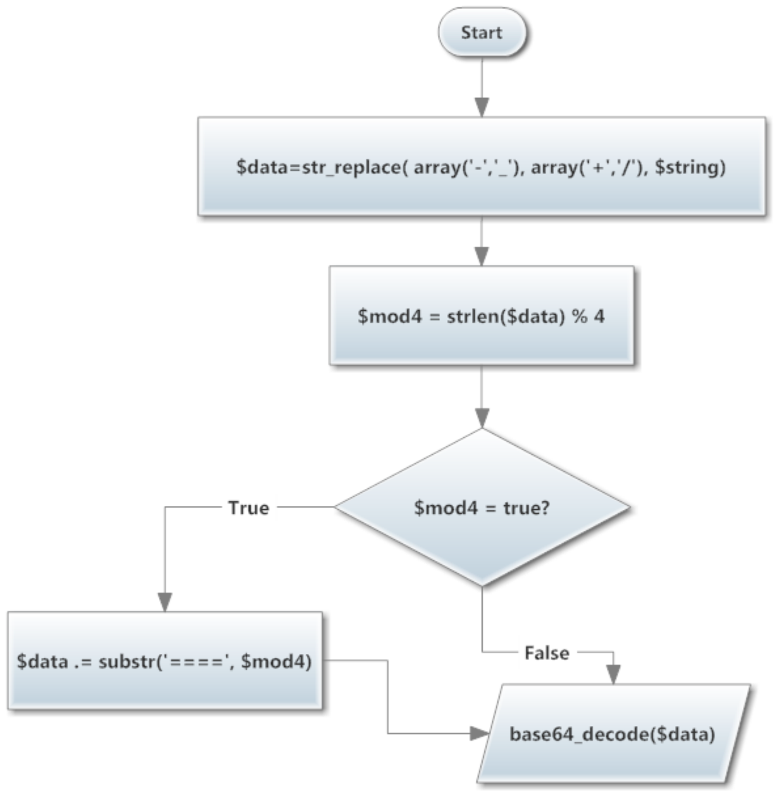

Start
explode('~', decrypt(user~pass, PLUGIN_CLICK2DIAL_ENCRYPTKEY))
Pass

File: class.click2dialfunctions.php

private function decrypt($string, $key)

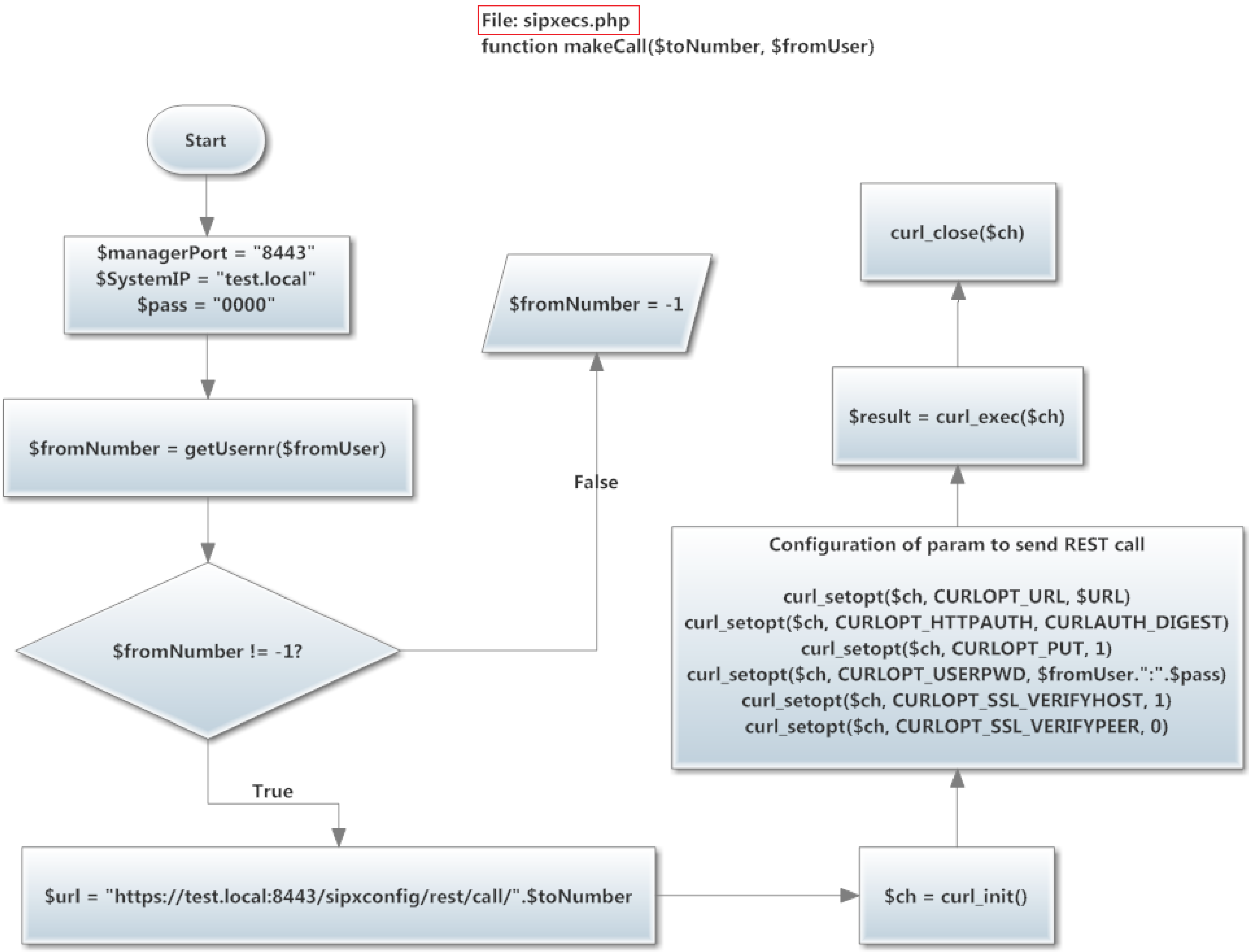


private function safe_b64decode($string)

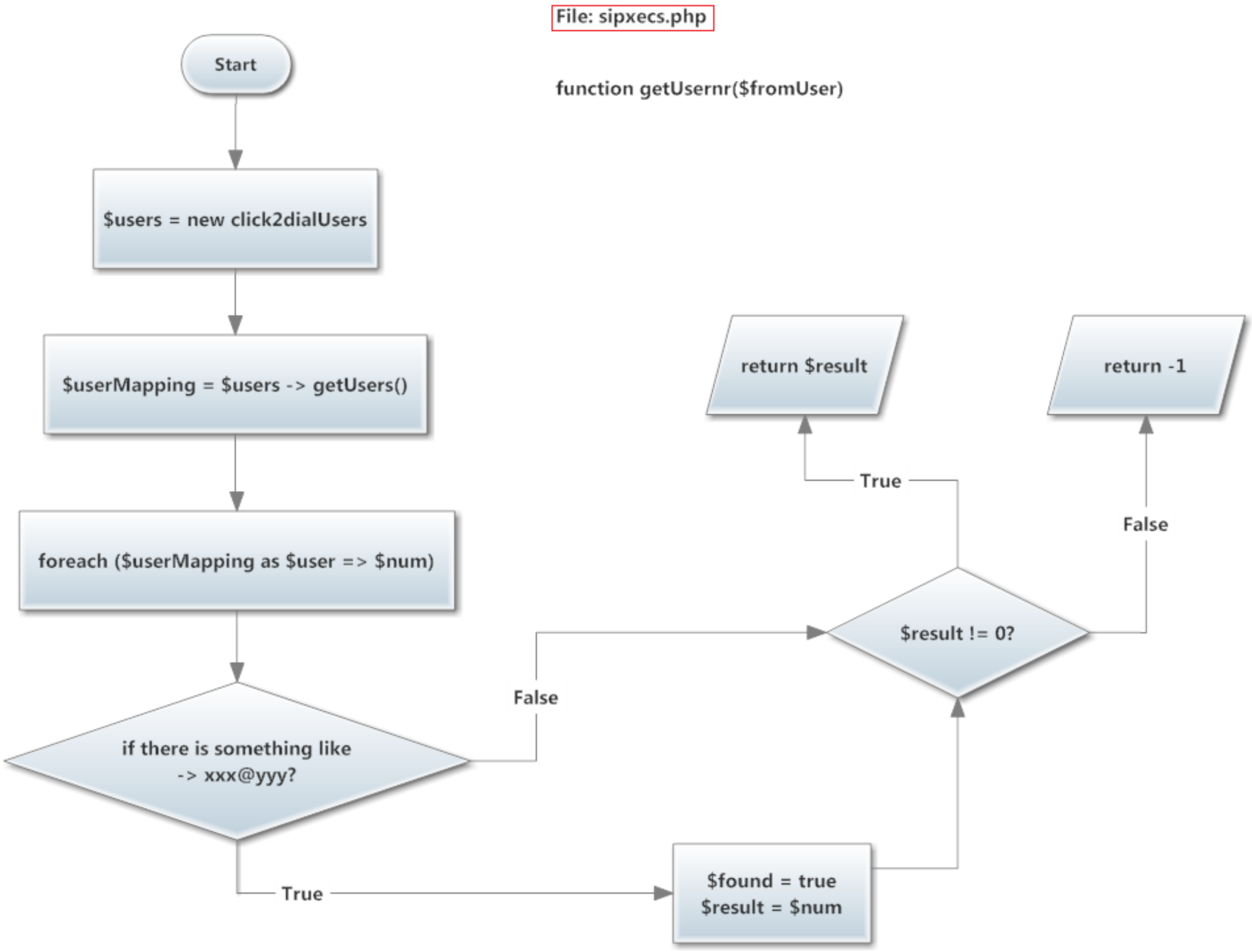

Start
$data=str_replace(array('-','_'), array('+','/'), $string)
$mod4 = strlen($data) % 4
$mod4 = true?
True
False
$data .= substr('====', $mod4)
base64_decode($data)

And I guess here we have the most important function. This function makes a call from one user to another. It needs some parameters, like: COMMUNICATIONS PORT, SERVER ADDRESS, and TELEPHONES PASSWORD. These parameters we can change in one "config.php" file, which I will show after this flowchart. The "config" file is good because if we change the domain of our server, or the password of the telephones, or the communications port we don't have to change the code of the program. Only change the information in the "config" file.

If we get a valid telephone number, we proceed to call. I will show the flowchart of the function getUsernr($fromUser) after. Now we have all data we need to build an "http command". We configure the options for the REST call with the parameters I have written. And the REST command is sent to SipXecs.


File: sipxecs.php
function makeCall($toNumber, $fromUser)
Start
$managerPort = "8443"
$SystemIP = "test.local"
$pass = "0000"
$fromNumber = getUsernr($fromUser)
$fromNumber != -1?
False
$fromNumber = -1
True
$url = "https://test.local:8443/sipxconfig/rest/call/".$toNumber
$ch = curl_init()
Configuration of param to send REST call
curl_setopt($ch, CURLOPT_URL, $URL)
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_DIGEST)
curl_setopt($ch, CURLOPT_PUT, 1)
curl_setopt($ch, CURLOPT_USERPWD, $fromUser.":".$pass)
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 1)
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0)
$result = curl_exec($ch)
curl_close($ch)

This is the getUsernr function, which search the telephone number of the user who wants to call. All users' addresses and numbers are in one file "users.php". If we can find one valid user, return the number. If not, return -1.


File: sipxecs.php
function getUsernr($fromUser)
Start
$users = new click2dialUsers
$userMapping = $users -> getUsers()
foreach ($userMapping as $user => $num)
if there is something like -> xxx@yyy?
True
False
$found = true
$result = $num
$result != 0?
True
False
return $result
return -1

10. CONCLUSION & FUTURE WORK

The CTI technology is the result bringing together telephony and computers to provide services. This project has achieved the main objective, which was made one example of software to make calls possible between users of a mail client. I have designed a program that allow us add new users with IP telephone and make calls to others registered previously in Zarafa Server. Further I have programmed a specific file, where we can change some parameters, for example to adapt our program to different PBX servers, so we only need change there the parameters and not change all the code of the program. We need to register users and telephones with the same name for program makes calls successfully. I think that’s a good thing to get better in the future, so we can create a database with all telephone numbers and its user names.

For make calls possible I have put the same password in both telephones (“0000”) but if you want to put your own password it doesn’t work, so we need to put the same password in both telephones when we register in SipXecs and after write the same password in the configuration file that I talked before. Then another thing to get better is to be able to change the password of the user and the password of the telephone from the Zarafa Webaccess Interface and be able to use different telephone passwords.

If we want to use this application with big telephony networks to call users in different geographic places, we should program a code which adds a prefix to the telephone number depending on the geographic place that the telephone is connected. It could be one more thing to improve.

Specially I have found some difficult in the beginning, because I didn’t know so well how to start and guide the first steps, and after I found some difficult in the programming task, because I have never seen before PHP and JavaScript language, but finally I could get my project runs successfully after reading some documents.

Finally, I think I have learnt many things while doing this project, because I have learned to deal by myself with problems I have found, and now I know a little bit more about IP-Telephony, Click2Call Technology and SipXecs PBX.

11. References

Chapter 2 - PBX:

[2.1] http://es.wikipedia.org/wiki/PBX

[2.2] http://en.wikipedia.org/wiki/Business_telephone_system

[2.3] http://www.3cx.es/voip-sip/sistema-telefonico-pbx.php

Chapter 3 – CTI (Computer Telephony Integration):

[3.1] http://en.wikipedia.org/wiki/Computer_telephony_integration

http://wiki.siemens-enterprise.com/wiki/CTI

[3.2] http://en.wikipedia.org/wiki/CT_Connect

[3.3] http://en.wikipedia.org/wiki/Telephony_Application_Programming_Interface

[3.4] http://en.wikipedia.org/wiki/TSAPI

Chapter 4 – Voice IP:

[4.1] http://transition.fcc.gov/voip

[4.2] http://es.wikipedia.org/wiki/Voz_sobre_Protocolo_de_Internet

[4.3] http://en.wikipedia.org/wiki/Voice_over_Internet_Protocol

[4.4] http://en.wikipedia.org/wiki/SIP_Trunking

[4.5] http://en.wikipedia.org/wiki/Session_Initiation_Protocol

Chapter 5 - REST:

[5.1] http://en.wikipedia.org/wiki/Representational_state_transfer

[5.2] http://www.xfront.com/REST-Web-Services.html

[5.3] http://es.wikipedia.org/wiki/Representational_State_Transfer

[5.4] http://en.wikipedia.org/wiki/SOAP

[5.5] http://www.desarrolloweb.com/articulos/1557.php

[5.6] http://msdn.microsoft.com/es-es/magazine/dd942839.aspx

[5.7] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_4

Chapter 6 - GROUPWARE:

[6.1] http://en.wikipedia.org/wiki/Collaborative_software

[6.2] http://en.wikipedia.org/wiki/Zarafa_%28software%29

Chapter 7 - INSTALLING OUR SYSTEM:

[7.1] [Michael W.Picher] Building Enterprise - Ready Telephony Systems with SipXecs 4.0 **Michael W. Picher**
Copyright © 2009 Packt Publishing
BIRMINGHAM – MUMBAI - ISBN 978-1-847196-80-4

[7.2] Webaccess Plugin Architecture.pdf

[7.3] Webaccess Plugin Paper.pdf

[7.4] The Zarafa Administrator Manual.pdf

Chapter 8 – WEBACCESS AND PLUGIN ARCHITECTURE:

[7.2] Webaccess Plugin Architecture.pdf

[7.3] Webaccess Plugin Paper.pdf

Chapter 9 – DEVELOPMENT OF MY PLUGIN:

[9.1] Webaccess Plugin Hooks.pdf

[9.2] JavaScript Manual

[9.3] Telephone Manual Snom 320

[9.4] http://php.net/manual/es/index.php "PHP online manual"

IMAGES USED IN THIS PROJECT:

Inside Images Folder

OSTFALIA LOGO:

http://www.mbm.med.uni-goettingen.de/bilder_artikel/Logo_Ostfalia.jpg

IMAGE 1:

http://pbx-system.waterheatingsystem.co.uk/what-is-a-pbx-system/

IMAGE 2:

Take from the website.

IMAGE 3: (MODIFIED BY ME WITH PAINT)

http://www.leegoeller.com/PBX/ONEPBX1.jpg

IMAGE 4:

http://www.tech-faq.com/computer-telephony-integration-cti.html

IMAGE 5:

http://www.2n.cz/images/cms/case-study/full/netstar-pbx-system-end-user-providers-es.png

IMAGE 6:

Draw with smartDraw 2012 Demo version

IMAGE 7:

Draw with smartDraw 2012 Demo version

IMAGE 6:

http://www.tech-faq.com/wp-content/uploads/images/cti.jpg

IMAGE 7:

http://www.ics.uci.edu/~fielding/pubs/dissertation/ccss_style.gif

IMAGE 8:

http://www.samusystems.in/images/voip1.gif

IMAGE 9:

I have got from the website and modificate

IMAGE 10 and 11:

Draw by myself with SmartDraw 2012 demo version.

IMAGE 12:

http://www.ics.uci.edu/~fielding/pubs/dissertation/uniform_ccss.gif

IMAGE 13:

http://www.ics.uci.edu/~fielding/pubs/dissertation/layered_uccss.gif

IMAGE 14 and 15:

http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#fig_5_4

IMAGE 16:

Got from the website, but website closed after I got image.

IMAGE 17:

http://habilidadespensamientocritico.wikispaces.com/file/view/image_groupware.jpg/152228263/image_groupware.jpg

IMAGE 18:

http://www.zarafa.com/images/support_technical-background/20110621_zarafa_architecture.jpg

IMAGE 19:

http://doc.zarafa.com/6.40/Administrator_Manual/en-US/html/_multi_server_setup.html

IMAGE 20:

http://doc.zarafa.com/7.0/Administrator_Manual/en-US/html/images/ZCP_ArchitectureDiagram.png

IMAGE 21:

I draw this image with SmartDraw 2012.

IMAGE 22:

Screenshot of my Zarafa Webaccess.

IMAGE 23:

Screenshot of my Zarafa Plugin Folder.

IMAGE 24:

Screenshot of my Plugin files structure

Flowcharts done by myself with SmartDraw 2012 Demo Version