Telecom and Internet Service Mashups in Robotics:

Architecture, Design and Prototyping

Zhou Jiale

zje08001@student.mdh.se

School of Innovation, Design and Engineering
Mälardalens Högskola, Västerås, Sweden
Ericsson Solution area Service Delivery and Provisioning, Sweden

A thesis submitted for the degree of
Master of Science in Intelligent Embedded System

Oct 2010
Acknowledgement

Here, I would like to acknowledge all who played a role in my project either directly or indirectly. I begin by thanking my family, who always supported me and my decisions and allowed me to concentrate on my studies without any worry. Then, I would like to thank my supervisor at Ericsson, Elena Fersman, who has always taken time from her tight schedule to give me all the help I needed in every phase of the project. The insights provided by her expanded my knowledge and problems understanding, enriching the results profoundly. I would like to thank Prof Lars Asplund, who recommended me to Ericsson, and let me have a chance to take such an interesting topic as my master thesis. I also extend my gratitude to my supervisor at MDH, Damir Isovic, who gave me suggestions about my thesis and scheduled my presentation. A final special thanks to all my friends Chalotte, Young, and anyone else from all over the world, who is filled and fills my life with a countless number of wonderful moments. At the very end I thank myself.
Content

Acknowledgement ........................................................................................................... 2
Abstract ............................................................................................................................. 4
Introduction ......................................................................................................................... 5
  1.1 Background .................................................................................................................. 5
    1.1.1 REST API ............................................................................................................... 5
    1.1.2 GSMA OneAPI ...................................................................................................... 6
    1.1.3 Ericsson IPX .......................................................................................................... 7
    1.1.4 Ericsson Labs API................................................................................................... 8
    1.1.5 Robotics ................................................................................................................ 8
  1.2 Thesis Topic .................................................................................................................. 8
Framework .......................................................................................................................... 10
  2.1 Hardware ...................................................................................................................... 10
    2.1.1 Robot .................................................................................................................... 10
    2.1.2 Other hardware ...................................................................................................... 12
  2.2 Software ....................................................................................................................... 12
    2.2.1 Telecom API .......................................................................................................... 12
    2.2.2 Internet Enabler ..................................................................................................... 15
    2.2.3 Robot .................................................................................................................... 18
Use Cases ................................................................................................................................ 31
  3.1 Overview ....................................................................................................................... 31
  3.2 MySQL Database .......................................................................................................... 32
  3.3 Nao local application ...................................................................................................... 34
    3.3.1 Robot Scenes .......................................................................................................... 35
  3.4 Web API application ....................................................................................................... 39
    3.4.1 SMS ....................................................................................................................... 39
    3.4.2 Location .................................................................................................................. 42
  3.5 Nao Remote Module ...................................................................................................... 44
    3.5.1 Location .................................................................................................................. 44
    3.5.2 Face Recognition .................................................................................................... 46
  3.6 Interaction with Mobile Phone ........................................................................................ 48
    3.6.1 Video streaming ...................................................................................................... 48
Conclusion ............................................................................................................................ 52
Bibliography .......................................................................................................................... 53
Abstract
Modern society expects services to run on any device anywhere, be it an internet service like Google search, or a telecom service like SMS. Why cannot we run Spotify (a famous music player) on my refrigerator, publish a picture of a meal directly to Flicker using my oven, or make a video call from my TV? Thanks to standardized high-level APIs e.g., GSMA OneAPI, developers can nowadays build services for any device with internet connectivity and open platform without integration difficulties. Increasing penetration of such devices allows for increased variety of services built for different types of industries and user groups.
This thesis presents a study of a service mash-up that uses both telecom and internet enablers deployed on a Nao robot platform. The robot runs embedded Linux, has internet connectivity, and is equipped with sensors, motors, cameras, speakers, and microphones. This makes it a good prototyping platform for large number of use cases from e.g. consumer electronics, healthcare or manufacturing industries. The prototype implemented in the framework of this thesis shows the robot recognizing people’s faces, answering questions, sending SMS and Facebook updates, locating people, and streaming live video to mobile phones.
The thesis is also an extension of the knowledge learned from courses. It covers software architecture design, real-time system design, embedded system, image processing algorithm, and web technologies.
Chapter 1

Introduction

This report is a Master thesis made for the School of Innovation, Design and Engineering (IDT) at Mälardalens University. The thesis work was commissioned by Ericsson Solution area Service Delivery and Provisioning (SA SD&P).

1.1 Background

With the development of internet, more and more web and network services (such as Facebook Service, Messaging and Location) are available in our daily life. Now we can publish a picture at any time and places directly to Flicker using our mobile phone, let the car find the way by itself using a GPS device, and pay our bills just by sending a text message. Thanks to standardized high-level APIs e.g., GSMA OneAPI, developers can nowadays build services for any device with internet and mobile network connectivity without integration difficulties. Our life is becoming full of fun and convenience. In this section, we shall discuss some background knowledge which will be helpful to understand the current situation.

1.1.1 REST API

Many of those web service APIs that you may be interested in are usually designed in REST style such as Google Maps APIs and Facebook APIs. REST is not a standard, but an architectural style. The acronym REST stands for Representational State Transfer, which basically means that each unique Uniform Resource Identifier (URI) is a representation of some object or resource. REST API usually relies on Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML) for web communication. A typical REST operation involves sending an HTTP request to the server and then waiting for an HTTP response which is described usually in XML or JSON. The request contains a request method, a Uniform Resource Identifier (URI), request headers, and optionally, a request body and query string. The response contains a status code, response headers, and sometimes a response body. For instance, your web application can receive a text message forwarded by the operator using an HTTP GET, or send it using a HTTP POST.

The URI of objects or resources usually look like the following:

http://example.com/{api version}/apifunction

If doing a HTTP Get operation, just append the parameters to URI, and then we get a new address like this:

http://example.com/{api version}/apifunction?parameter1=value1&parameter2=value2

POST operation is a bit more complex. We should define the request header like this:

Host: example.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length of parameters
The parameters mentioned above depend on what the object the URI represents. This can be defined by service providers.

1.1.2 GSMA OneAPI

What really makes developers headache is that the world of telecommunications is complex and riddled with barriers, making the development of applications for mobile phones and other mobile devices a difficult task. Some examples of the complexity include a severely fragmented device platform environment and difficulty in coming into contact with end users/consumers, network capabilities, and advanced functionality. Ordinarily, only those developers who have in-depth knowledge of the underlying systems are able to access this functionality.

GSM Association (GSMA) recognizes the dilemma that developers are in and has taken active measures to make matters easier for them. OneAPI (Open Network Enablers API) is a ground-breaking GSM Association initiative. It is a standard that contains a set of APIs which may be accessed via a REST API style. OneAPI will help developers simplify the implementation of telecom functions in their services and applications. The standard is supposed to be supported across network operators to allow consistent access to enablers (message, location, payment, etc) and hence encourages Web developers to mash-up network capabilities in their desktop apps and stimulates cross-operator mobile applications.

![Image of OneAPI Accesses Multiple Operators](image)

As is shown in Figure 1.1, GSMA OneAPI works as a bridge connecting the internet and mobile network. OneAPI comprises two suits of Software Development Kit (SDK), one for application developers and the other for operators. The lightweight APIs encourages the portability of Mobile apps, but still allow for competition and differentiation between operators. Any network operator or service provider is able to implement their own telecom APIs according to OneAPI standard. Except providing an OneAPI implementation based on the operator SDK, operators should also have their own gateway for access by authenticated application developers.

At this time OneAPI is only in the initial phase. It contains messaging, location and charging APIs,
which can be used for on-time Website password, cross-operator friend finder, seamless mobile billing, etc. OneAPI is looking at the possibility of making delivery of video content to handsets easier, In-app billing, Click-to-call, and Call notification. These specifications should be available at the end of 2010. In the future, more APIs will be added and more operators will participate in. With more and more operators and companies participating in the project, GSMA OneAPI will be able to dramatically broaden the number and variety of mobile services available to end-users, enabling the creation of feature-rich applications to support the mobile lifestyle.

1.1.3 Ericsson IPX

Ericsson, as one of the main certified partners of GSMA OneAPI project, developed their own implementation of GSMA OneAPI - Ericsson IPX. Ericsson IPX is a leading mobile aggregator, providing commercial delivery and billing services. IPX comprises four main telecom solutions - Mobile Payments, Mobile Messaging, Operator Services, and Subscriber Information. It provides payment and messaging services to all kinds of providers, including content and service providers, mobile network operators, enterprises, banks, charitable organizations and media and marketing companies. It works closely with customers to build business and develop revenue generating services for customers through connections with more than 100 operators, with a global messaging offer and payment in 26 countries. Let us have a look at what Ericsson IPX will bring:

Mobile Payments: Mobile Payments is a new, simple and rapidly-adopting payment method. You can get rid of your bank card reader when you pay your bills online. Instead, Mobile Payments makes it simple to pay bills by merely sending a short text message. A wide range of services and digital or hard goods can be incorporated, such as music, video, online games, bus tickets, and books. Meanwhile, Mobile Payments enables payment and delivery of content all over the world.

Mobile Messaging: Mobile messaging is an alternative method of delivering information and creating a direct and immediate dialogue with your customers via text messaging. Ericsson IPX Messaging can help providers keep contact reliably with their customers at anytime and anywhere all over the world. The pretty low price is also an indispensable advantage which can help both the customers and providers save their money.

Operator Services: Operator services bend itself to helping providers get rid of the barriers brought by building a reliable and safe network infrastructure. Providers can focus on building attractive content without worrying about delivering. Operator Services will offer them the ability to deliver their content to anyplace worldwide at any time.

Subscriber Information: Subscriber Information means the information, such as location, which can help providers improve their experience for subscribers and offer more personal and adaptive services. Subscriber Information can also take the full advantage of unique attributes of mobile phones to avoid fraud in order to offer safer services.
1.1.4 Ericsson Labs API

Ericsson Labs is a web portal that contains beta enablers and applications that enable developers to reach operators, simplify the implementation of basic telecom functions and enable access to advanced telecom functionality. Ericsson Labs also serves as an experimental environment for Ericsson IPX, which helps experimental developers facilitate the beta testing of new telecom and internet multimedia enablers. Some of Ericsson Labs’ APIs are based on IPX APIs, offering developers an easy and free access to test the capabilities.

There are two main parts at Ericsson Labs, which are a developer site, which contains the enabler APIs and associated resources, and an application showroom, which contains developed end-user applications. A backend system provides developer support and tools for creating, publishing and evaluating applications.

In this thesis, telecom APIs from Ericsson Labs such as messaging and location, have been used.

1.1.5 Robotics

Robotics is a research field that has experienced several changes in the last years. The NAO Humanoid Robot from Aldebaran Robotics is a well-proven companion, assistant and research platform. It is one of the most famous robots worldwide. In July 2007, Nao was nominated as the official platform for the standard league of the RoboCup which is a football game played by robots.

ALDEBARAN’s robots already provide a platform to deliver a range of daily services. Nao is being introduced on the market as the ideal introduction to robots and aspires to eventually, become an autonomous family companion. Finally, with more sophisticated functions, it will adopt a new role, assisting humans with daily tasks. Nao is equipped with Embedded Linux 32 bit processor with 256 MB RAM, 2 GB flash-memory, and WIFI. Programming languages that can be used for developing applications running on the robot are C, C++, Urbi script and Python.

1.2 Thesis Topic

The objective of this thesis is to design a service that would run on an unusual platform, which
would be using a mixture of telecom APIs (such as messaging and location) and internet APIs (such as Facebook APIs). The target hardware for the demo is a PC, a mobile phone and a Nao robot. The purpose of the thesis is to demonstrate usefulness of the online telecom APIs provided by Ericsson, that can cooperate with other proven APIs on any platform, including Nao robot platform.

The prototype shall consist of three parts:

1. Server-side application running on server computer, which communicates with the thin clients running on both mobile phone and robot.
2. Application running on Nao robot, which is developed to interact with human.
3. Mobile phone receiving SMS, visiting video streaming service, and acting as the target located by Nao
Chapter 2

Framework

This section introduces the hardware and software we used to build our prototype in detail. Based on this introduction, readers are supposed to get a better understand on how we build our use cases.

2.1 Hardware

2.1.1 Robot

Nao is a biped robot with 25 degrees of freedom. It is capable of executing a wide range of movements (walking, sitting, standing up, dancing, avoiding obstacles, kicking, seizing objects, etc.). Nao’s motion is based on DC Motors (direct and stepper types), and for each motor, there are two types of speed reduction ratio.

Inside the robot, 5 chains of joints are defined. You can send commands to individual chains without having to send commands to the whole body or to a single motor. This can be useful when you only wish to describe an arm motion.

![Figure 2.1 Nao’s Chains of Joints](image)

Also, to provide the largest flexibility and precision, 25 joints can also be controlled respectively by the joint’s name.

<table>
<thead>
<tr>
<th>Head</th>
<th>LArm</th>
<th>LLeg</th>
<th>RLeg</th>
<th>RArm</th>
</tr>
</thead>
<tbody>
<tr>
<td>HeadYaw</td>
<td>LShoulderPitch</td>
<td>LHipYawPitch1</td>
<td>RShoulderPitch</td>
<td></td>
</tr>
<tr>
<td>HeadPitch</td>
<td>LShoulderRoll</td>
<td>LHipRoll</td>
<td>RHipRoll</td>
<td>RShoulderRoll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LShoulderPitch</td>
<td>RShoulderPitch</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LHipPitch</td>
<td>RHipPitch</td>
<td>RELbowRoll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RShoulderPitch</td>
<td>RShoulderPitch</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RELbowRoll</td>
<td>LKneePitch</td>
<td>RKneePitch</td>
<td>RELbowRoll</td>
</tr>
<tr>
<td></td>
<td>RShoulderPitch</td>
<td>LAnklePitch</td>
<td>RAnklePitch</td>
<td>RWristYaw</td>
</tr>
<tr>
<td>LHand2</td>
<td>RAnkleRoll</td>
<td>LAnkleRoll</td>
<td>RHand2</td>
<td></td>
</tr>
</tbody>
</table>
Nao has many sensor devices to obtain robot’s close environment information:

**Sonar** Nao Robot has 4 sonar devices that provide space information. They are divided into 2 sides and situated in both right and left chest. The detection range goes from 0 cm to 70 cm, but less than 15 cm there is no distance information, the robot only knows that an object is present.

**Cameras** The last Nao Robot version (3.0 plus) was provided with two identical cameras both situated at forehead (top and bottom). They provide a 640x480 resolution at 30 frames per second by default. And there is a list of parameters which can be modified, including brightness, contrast, saturation, auto white balance, camera resolution, fps, etc. By adjusting these parameters, we can get a balance between image quality and processing performance.

**Bumper** A bumper is a contact sensor that helps us to know if robot is touching something, in this case the bumpers are situated in front of each foot and they can be used, for example, to know if there are some obstacles touching our feet. This sensor is very useful when the obstacle is too short to be detected by ultrasound sensors.

**Force Sensors** Nao has 8 Force Sensing Resistors (FSR) situated at sole of feet. There a 4 FSRs in each foot, then. The sensor value doesn’t change linearly and need to be calibrated. When no force is applied the sensor reading is 3000 and when the sensor reading is 200 means that it is holding about 3 kg. The sensors are useful when we are generating movement sequences to know if one position is a zero moment pose (ZMP) and its sensors can be complements with inertial sensors.

**Inertial Sensors** Nao has a gyrometer and an accelerometer. These sensors are two important devices when we are talking about motion concisely Kinematics and Dynamics. The output data enables an estimation of the chest speed and attitude (Yaw, Pitch, and Roll). The estimation helps us to know if the robot is in a stable position or in unstable one when robot is walking, and in lying position or in standing one when robot is static.

**Microphones** Nao is equipped with 4 omnidirectional microphones located around its head (front, rear, left, and right). The electrical band-pass of the microphone’s ways is: [300Hz - 8 kHz]. By calculating the sound energy and the position of each microphone, we can get an approximation of the direction of sound source.

**Loudspeakers** In order to play sound, Nao is equipped with a stereo broadcast system made up of 2 loudspeakers in both of its ears.

Nao’s head is equipped with an AMD Geode 500Mhz CPU motherboard based on x86 Intel architecture with 256MB SDRAM and one ARM7 at 60Mhz microcontroller located in the torso that distributes information to all the actuators module microcontrollers. Nao’s CPU is the main processor system in the robot and its limited computational capacity is shared by operating
2.1.2 Other hardware

The server and mobile phone are also necessary in the demo application. The server we adopted is Lenovo T400 (CPU 2.26GHz/Memory 2GB/Hard disk 320GB/Wireless and wired network card/Win 7 OS). Mobile phone is required to be able to access 3G or Wi-Fi network. We use Iphone 3GS.

2.2 Software

2.2.1 Telecom API

2.2.1.1 Location

The Web Location API enables you to retrieve the location of your users from their mobile phones in order to use the information in your web applications. The API uses the positioning systems of mobile operators, so your users don’t need to install any applications on their phones, and the positioning works for all types of phones. However, End-user integrity is very important. Positioning is allowed only with explicit permission from users. Ericsson’s web location API includes functionality to collect and manage end-user consent and to handle user cancellations of consent. The necessary end-user interaction is done via SMS or wap.

Here’s how it works:
The API lets a developer locate his users in two different ways:

1. In a mobile web application, when a user clicks on a link the location of the user is sent to the application which can respond with customized content (e.g. all nearby restaurants). This is called Active Positioning.

2. In any web application, the application sends a phone number to the Web Location service and the service responds with the location of the user. This is called Passive Positioning.

Since passive positioning is ideal for situations where your users do not access your application or service directly through their mobile phones, in our application only passive positioning is adopted. Therefore, the following focuses on the detail about passive positioning.

To retrieve a user’s location using Passive Positioning there are 4 steps needed to follow:

**Step 1:** Apply for an API key and an application identifier. API key will allow you to invoke the Web Location API from your web application. This is the key that uniquely identifies you as a registered developer using Web Location. The API key should be kept secret at all times. An application identifier is a short string that you enter when you sign up to Web Location. The application identifier identifies your application. Since it is used by Web Location in the SMS messages that communicate consent / cancellations of consent with the end-user an identifier that users understands belongs to your application should be picked.

**Step 2:** Request end-user consent. You initiate the request to create an end-user consent by sending the MSISDN (phone number) of the user you want to position to the Web Location service. Web Location will then send an SMS to the user to confirm that he/she agrees on being positioned. This is to ensure that your users are ok with being positioned and the show them that they are in control of by whom they can be positioned and for how long. The base URL for requesting the end-user consent has the following format:

http://mwls.labs.ericsson.net/mwlservice/resource/consent/create

Requests are standard HTTP POST methods, with parameters placed in the payload. The parameters needed are:

- **applicationID** This is the application id that you selected when signing up for the Web Location API key.
**devKey** This is the API key that you received when signing up for the Web Location API key.

**msisdn** The end-user MSISDN identifier (mobile phone number). The MSISDN should be formatted like a regular phone number starting with country code, e.g. “46703123456”.

**endDate** End of the valid period for the consent. The time and date from which this consent will no longer be valid. The date should be given on the format “YYYY-MM-DDThh:mm” where the time is in GMT.

**Step 3:** Check the status of the consent. Before making a request to position a user you could check if the consent has been created and is valid by sending a request to Web Location. The base URL for requesting a consent status has the following format:

http://mwl.labs.ericsson.net/mwlservice/resource/consent/status

Requests are standard HTTP POST methods, with parameters placed in the payload. The parameters needed are:

- **applicationID** This is the application id that you selected when signing up for the Web Location API key.
- **devKey** This is the API key that you received when signing up for the Web Location API key.
- **msisdn** The end-user MSISDN identifier (mobile phone number). The MSISDN should be formatted like a regular phone number starting with country code, e.g. “46703123456”.

**Step 4:** Request the location of the user. Send a request to the Web Location service with MSISDN of the user you want to position and the approximate location of the user’s mobile phone is returned. The base URL for requesting the location, using an end-user phone number, has the following format:

http://mwl.labs.ericsson.net/mwlservice/resource/location/getLocationByMsisdn

Requests are also standard HTTP POST methods, with parameters placed in the payload. The parameters needed are:

- **applicationID** This is the application id that you selected when signing up for the Web Location API key.
- **devKey** This is the API key that you received when signing up for the Web Location API key.
- **msisdn** The end-user MSISDN identifier (mobile phone number). The MSISDN should be formatted like a regular phone number starting with country code, e.g. “46703123456”.

### 2.2.1.2 SMS

The SMS Send & Receive Web API provides developers with the opportunity to develop SMS enabled web services. The API consists of a simple web interface for sending SMS messages to mobile phones, as well as an easy-to-use service identifier-based system for receiving messages. To be able to use the API you need to apply for an API key. This key is used to identify you as a developer when you're sending and receiving messages. When you sign up for an API key you will
get:
1. 1000 SMS messages that you can send for free
2. A service identifier on the SMS Send & Receive API shared short code, allowing you to receive messages sent from mobile phones
3. A shared phone number that can be used to send and receive messages internationally, as well as a shared short code that works in Sweden

When receiving, a service identifier in the beginning of incoming SMS messages tells the system that the messages are intended for a particular web service. Messages that start with this service identifier are then forwarded to a specified callback URL (i.e., the developer’s web service address) on a web server associated with the identifier. When the callback URL is requested, the web server receives the message contents along with information about its origin. The SMS callback interface assumes that the service at the callback URL is capable of handling incoming HTTP GET requests with parameters formatted like this:

http://URL?from=[MSISDN]&time=[TIMESTAMP]&message=[MESSAGE]&hmac=[HMAC]

When sending a message, the rules are much simpler than receiving. What is needed is to make a web request which is standard HTTP GET or POST, with parameters either placed in the URL or in the payload in the following manner:

http://sms.labs.ericsson.net/send?key=[KEY]&to=[MSISDN]&message=[MESSAGE]

2.2.2 Internet Enabler

Many websites have their own internet APIs which can help developers access to website’s services easily. The most famous and useful APIs are Google Earth APIs. They help web applications or websites possess the ability of distinguishing visitors’ location which makes applications much more user-friendly and more interesting. In this section, we introduce another two well-known Internet APIs which are incorporated in our prototyping - Google Map APIs and Facebook APIs.

2.2.2.1 Google Maps API

As is discussed above, Ericsson Location API will get the longitude and latitude of the target phone. Then we call Google Maps API to translate its coordinate into human-friendly location information such as city name or street name.

Google Maps has a wide array of APIs that let you embed the robust functionality and everyday usefulness of Google Maps into your own website and applications, and overlay your own data on top of them. Google Maps API provides web services as an interface for requesting Maps API data from external services and using them within your Maps applications. These web services use HTTP requests to specific URLs, passing URL parameters as arguments to the services. Generally, these services return data in the HTTP request as either JSON or XML for parsing and/or processing which depends on your application. A typical Google Maps API Web Service request is generally of the following form:

http://maps.google.com/maps/api/service/output?parameters
where service indicates the particular service requested and output indicates the response format usually JSON or XML. To use Google Maps API, what you need to do is only to apply for a Google developer key, and for the sake of security, you must send the key to Google’s server as one of the parameters of the Web Service request.

2.2.2.2 Facebook REST API

The Facebook API is a Web services programming interface for accessing the members of the social network and performing other Facebook-centric functionality (log in, redirect, and update view). The API allows applications to use the social connections and profile information to make applications more involving, and to publish activities to the news feed and profile pages of Facebook, subject to individual user’s privacy settings. With the API, users can add social context to their applications by utilizing profile, friend, Page, group, photo, and event data. The API uses RESTful protocol and responses are localized and in XML format which enables you to interact with Facebook web site programmatically via simple HTTP requests.

Assuming that you have already registered an account of Facebook, to be able to manipulate via Facebook APIs, here are three steps to follow:

**Step 1:** Register your application at Facebook developer center. Before coding the application itself, you are going to need to get some application-specific information from Facebook to communicate with the Facebook Platform. Your first step is, therefore, to register the application with Facebook. When you do so, you’ll get an API key and secret key that you can plug into your code.

**Step 2:** Configure client library. Client library contains the interfaces to Facebook services which save lots of developer’s effort. Facebook officially supports client libraries for PHP (4 and 5) and Java. When you have an API key and secret key, you need to download the suitable language-specific Facebook SDK for your code, unzip the library, and place these files in your folders. Most of Facebook applications are Web application. Facebook supplies a well support for PHP. The developer just simply places PHP library in the server’s folder which is specified as the callback URL for your application.

**Step 3:** Get user’s authorization. This step is the most important and complicated one in the process. Facebook Platform uses the OAuth 2.0 protocol for authentication and authorization. In order to access or modify user’s data, you need to get proper authorization first. The abstract protocol flow is shown in figure below. After you obtain the access token for a user, you can perform authorized requests on behalf of that user by including the access token in your Facebook API requests.
The authentication consists of two levels. By default, your application can access all general information in a user’s profile, including his/her name, profile picture, gender, and friend list. If your application needs to access other parts of the user’s profile that may be private, your application must further request extended permissions. For example, if you want to incorporate a user's photos into your application, you would request the user_photos extended permission. During the authentication process, the user is presented with a UI similar with the one shown below in which the user can authorize your application to access that specific part of her profile.

When you finish the steps above, you can call Facebook APIs in your application. Similar with other REST APIs, the operation involves sending an HTTP request to Facebook and then waiting for an HTTP response. The responses from Facebook are JSON objects.
2.2.3 Robot

2.2.3.1 Naoqi

2.2.3.1.1 Overview

The software framework of Nao is called Naoqi which is developed by Aldebaran Robotics. Naoqi answers to common robotics needs: parallelism, resources, synchronization, events. In Naoqi as in other frameworks we'll find generic layers but it was created on Nao and fit to the robot. Naoqi allows homogeneous communication between different modules (motion, audio, video, etc.), heterogeneous programming (c/c++, python, urbi) and heterogeneous information.

Naoqi is designed as a distributed system where each component/module can be executed locally in robot’s on-board system, or executed among distributed systems meanwhile Naoqi Daemon is running in the main system. To implement remote procedure call (RPC), Naoqi defines two important terms:

**Broker** is an executable that listens to remote commands on an IP address and port. Naoqi can communicate with distributed components/modules via brokers.

**Proxy** is a pointer which is used to access to one module’s method by other modules (remote or local). To call a method from a module, we need create a proxy to the module.

This architecture simplifies the development of developer’s own module and reduces the coupling of different modules.

Naoqi is divided into three main layers, Naoqi Operating System, Naoqi Module Library and Device Control Manager (DCM). We shall talk about them more detailed in the coming paragraph.
Naoqi Operating System called OpenNao is an Open Embedded Linux Distribution modified to be executed on Nao onboard system. It is composed of the following core modules:

- **ALLauncher**: automatically loaded. Manage external library
- **ALPreferences**: manage xml preference file.
- **ALMemory**: shared memory of robot.
- **ALLogger**: manage log.
- **ALFileManager**: Manage robot files.
- **ALBase**: the kernel module of the OS

The main function is:

- Allocate or deallocate shared memory
- Manage programming in parallel, sequential or event driven calls manner
- API management to show or hide methods to other applications
- Catch exceptions
- Encapsulation of http communication for remote calls between computers
Naoqi Library consists of several modules which are optional and can be developed by users as well. These modules have implemented the minimum functions to interact with the robot, sensors and actuators, to develop our own behaviors and actions. The functionality allows us to design robot behaviors within an abstraction layer, which avoid us to access directly to the hardware. Otherwise, that abstraction only is useful in the first steps of our development because if we want a high performance then we have to adapt the implementation to Nao’s hardware architecture. That’s the job of DCM.

The last division is DCM. DCM like Naoqi Library is a set of libraries that allow us to control the robot. The difference between DCM and Naoqi Library is that DCM controls the robot directly sending function calls to Nao’s ARM controller. As well as, ARM is Nao’s hardware architecture part that controls and senses all motors and input devices except cameras, because they are connected to head’s on-board system. DCM is essential if we want generate a new sequence for
walking, or reach a position. But DCM has some disadvantages, the first one is that robot stability is disabled and that stability has to be guaranteed by the developer using some stability techniques of biped robots. And operating DCM is much more complicated than using Naoqi Library.

2.2.3.1.2 Modules
In most cases, Naoqi Library has supplied us all the necessary modules which facilitate lots of developing time. It is almost one-to-one correspondence between Library modules and hardware devices. To have a better understanding, we shall talk about several mostly used modules.

A. Memory
ALMemory is the robot memory module. Data in the memory has its own name as the identifier. Through the proxy of ALMemory, other modules can read or write data, or subscribe on data so as to be called periodically or only when data changes. Variable access is thread-safe. It uses read/write critical sections to avoid bad performance when memory is read. Memory will store an array of ALValue's. ALValue class is a complex variant object to store and operate data. ALValue is compatible with several data type, including boolean, int, double, float, char*, and binary.

![ALMemory diagram](image)

Figure 2.10 ALMemory

B. Motion System
ALMotion module was designed to facilitate the control of Nao's movements. This section will introduce more details about Nao's motions system, which are divided into four aspects - Commands, Interpolation, Support Mode description, and Balance Mode description.

Commands: ALMotion module offers two kinds of low level commands, Joint-space and Cartesian-space, by which you can conveniently control Nao's movement. Joint-space commands take joint's name as the target. Each joint can be controlled individually, or in parallel with other joints. The joints have been listed in the counterpart hardware section. To control a joint, you need to specify the name of the joint and the target angle in radians. The Motion Module also provides a set of functions to control the specific chains in a Cartesian space which are called Cartesian-space commands. To control the end of a chain in the 6 Dimensions of the Cartesian space...
Space, 6 values need to be defined: 3 for velocity (dx, dy, dz) and 3 for orientation (dwx, dwy, dwz).

![Diagram of Six Dimensions of the Cartesian Space](image)

In some cases when the chain is not redundant like Nao Arms, a 6 Dimension Task is impossible to achieve. However, in most cases only one velocity or orientation needs to be controlled and the user doesn’t care so much about the other dimension. So, some Axis Masks are defined in the Motion Module:

```
#define AXIS_MASK_X 1
#define AXIS_MASK_Y 2
#define AXIS_MASK_Z 4
#define AXIS_MASK_WX 8
#define AXIS_MASK_WY 16
#define AXIS_MASK_WZ 32
#define AXIS_MASK_ALL 63
#define AXIS_MASK_VEL 7
#define AXIS_MASK_ROT 56
```

**Interpolation:** ALMotion module has defined an interpolation algorithm which is the key difference from DCM module. To facilitate the use of joint-space commands, two methods - LINEAR and SMOOTH (3rd degree polynomial) exist which interpolate from one position to another. Thus, you don't need to provide a full stream of very small changes. The diagram below represents the Speed and Position for SMOOTH and LINEAR interpolations between positions '0' to '1' over 100 samples.
**Support Mode description**: The Support Mode defines which leg is on the ground and therefore which leg is responsible for realizing Center of Mass (COM) trajectory and Torso orientation commands.

For the sake of memorization, define an enumeration SUPPORT_MODE like this:

```cpp
enum SUPPORT_MODE {
    SUPPORT_MODE_LEFT = 0,
    SUPPORT_MODE_DOUBLE_LEFT = 1,
    SUPPORT_MODE_RIGHT = 2,
    SUPPORT_MODE_DOUBLE_RIGHT = 3
};
```

Below is the Support Mode while Nao is walking.

**Balance Mode description**: The Motion module provides an open loop stabilizer algorithm which controls the COM and the Torso Orientation of the robot using the legs. Three models are
available. For the sake of memorization, define an enumeration BALANCE_MODE like this:

```c
enum BALANCE_MODE {
    BALANCE_MODE_OFF = 0,
    BALANCE_MODE_AUTO = 1,
    BALANCE_MODE_COM_CONTROL = 2
};
```

**BALANCE_MODE_OFF** In this mode, the open loop of stabilization is disabled. So, only Joints or Cartesian-space commands are available.

**BALANCE_MODE_AUTO** This mode is the Automatic Mode. The robot tries to keep both projection of the COM in the center of Robot's Support Polygon which depends on the Support Mode. In this Mode, Joint-Space and Cartesian-Space commands can be used to control the non-support leg. In Double support, no leg is free for such control. Torso Orientation commands are also available. This mode is very useful to create stable motion in the single support Mode with the other leg. In addition, this mode yields very natural human motion when arms are controlled.

**BALANCE_MODE_COM_CONTROL** In this mode, all the functions of motion are available: Joint Space, Cartesian Space, torso Orientation and com position. As above, joint space and Cartesian space commands cannot be used on supporting legs.

C. **Main Sensors**

The ALSonar module and ALInertial module are extractors respectively for the sonar and inertial devices. Since several other modules may need the result of the sensor and may start and stop it at any time, the use of sensor work with subscriptions. The modules enable other modules to subscribe and unsubscribe to them and then writes the result in ALMemory. Sensor modules will run whenever at least one module is subscribed to it.

ALSonar needs only one parameter: the period of acquisition of each actuator (an integer in milliseconds). If several modules are subscribed to ALSonar, the smallest period will be chosen. Because of hardware constraints, the period shall be more than 240 milliseconds. 500-millisecond is recommended. The result of the acquisition is stored in ALMemory module with the key: "extractors/alsonar/distances". The value is an array of two values: the left sonar's acquisition and the right one.

Likewise, ALInertial also needs only the period of acquisition of each actuator. The result of the acquisition is stored in ALMemory module with the key: "extractors/alinertial/position". The value is a string among "back", "front", "right", "left", "up", "down".

D. **Audio System**

Nao's audio system makes itself speak as well as listen, including ALAudioPlayer, ALTextToSpeech, ALAudioSourceLocalization and ALSpeechRecognition.

**ALAudioPlayer**: This module allows the robot to play WAV and MP3 files which are uploaded to Nao's flash memory. In our demo, Nao plays background music while it is dancing.

**ALTextToSpeech**: This module, as is implied by the name, allows the robot to speak. It sends
text string to a text-to-speech engine which parses the string, and then speaks it out.

**ALSpeechRecognition:** The ALSpeechRecognition module allows the robot to recognize words in a list that defined by users. ALSpeechRecognition works as an extractor, which means that after Nao detects someone is speaking, the recognized words are stored in the ALMemory module and the result can be read by any other modules. The structure of the result is an array: [(string) word, (float) confidence]. ALSpeechRecognition module uses the signal from the front microphone. The vocabulary and the language can be changed on the fly among English, French and more in the future.

**ALAudioSourceLocalization:** ALAudioSourceLocalization module calculates the sound energy of each microphone. The higher the energy, the closer the microphone is to the source of sound. Therefore, the microphone with highest energy indicates the direction of the sound. Like ALSpeechRecognition, the results are also stored in ALMemory, and you can subscribe to the results. However, this module is pretty vulnerable to noise. Especially, the sound made by fans inside Nao’s head disturbs the rear microphone a lot, so we usually ignore the value of rear one. As per experiments, the nearer the source is, the better the performance. Less than 20cm is recommended.

**E. Video System**

Figure 2.15 Architecture of Video System [3]

Video system incorporates Video Input Module (VIM) and Generic Video Module (GVM). The VideoInput module (VIM) is designed to give an access to the video stream to every module related to vision-access provided by Nao’s camera, by a simulator on a computer, by a file recorded from one of the preceding source and running either on Nao or a computer, or later by other sources. It is designed to provide either direct access to raw data from the video source without any copy, or converted data in the selected resolution and color space. This data can be accessed locally by directly pointing to it, or remotely through an ALValue encapsulation. Finally, when a module requires the same kind of data than a preceding one, this module will access it directly if the data is still up-to-date. This means that the VIM manages data mutualization in order to save processing power.

A Generic Video Module (GVM) represents the module developed by users themselves. GVM
needs to work on a specific image format to perform its processing. So it registers to the VIM that will apply transformations on the stream to the required format. If this format is a native one for video source, a direct raw access can be asked. Once your GVM is registered to the VIM, you can get a video buffer. Regardless of the kind of buffer requested (raw buffer or converted one), there are 2 possibilities: either your module works on the local system (the robot) or on a remote machine. If your module works locally, the memory addresses are shared. So a fast method to get the video buffer is to use the “getImageLocal” or “getDirectRawImageLocal” methods. If your module is remote, the memory addresses are not shared. So it’s useless to get the pointer to the video buffer. Your module is able to get the data contained in the video buffer through using the “getImageRemote” or “getDirectRawImageRemote” methods.

2.2.3.2 Choregraphe

Choregraphe is a cross-platform, user-friendly, powerful developing tool that allows you to edit Nao’s movements and behaviors. The recommended programming language is python, although Urbi is available as well. Choregraphe can develop both simple and complicated application running on Nao. Figure 2.16 shows the worksheet of Choregraphe. The worksheet is divided into 3 major panels:

- The box library
- The Flow diagram
- The 3D Nao simulator

![Figure 2.16 Panels of Choregraphe](image)

**Box library panel** displays a list of boxes ordered in categories according to the nature of the boxes and their functions. You can drag and drop any of these boxes onto the flow diagram.
to enrich a behavior. All the boxes of this library have been created with Choregraphe. You can also create your own boxes to implement a certain function. A box is a fundamental object in Choregraphe as everything you can create with this software rests on boxes.

**Flow diagram panel** is the place where you “compose” Nao’s behaviors. To create a behavior, you need to grab the boxes from the Box library and drop them onto the flow diagram. You can also create a new box from scratch by right-clicking on the diagram screen and pressing Add a new box. The diagram panel allows you to connect a box to either another box, or to the behavior entry. The flow diagram follows a multilevel graphical logic to organize the links between the boxes and structure a behavior. As a matter of fact, to have a more readable flow diagram, it is possible to group boxes linked to each other into a unique one and therefore create multilevel boxes.

**3D Nao panel** is a 3D representation of the robot used to create and simulate behaviors. This 3D model can either be a representation of a simulated Nao (if you are connected to a simulator) or the virtual mirror of the real Nao robot you are working with (if you are connected to the real robot IP address). Press the mouse buttons to zoom and move around the virtual model. You can also select any part of the robot (head, arms, body and legs) by simply clicking on it. This action will open a motion window like this one:

![Figure 2.17 Motion window of arms](image)

This window allows you to modify the joints values of each limb. You can see the result of
your actions on the virtual Nao model. The simulator can easily test the correctness of your defined movement and avoid dangerous factors as many as possible.

Choregraphe simplifies our development to a great extent, but it is not perfect. The main drawback is that the demo written by Choregraphe is essentially a python application which needs to be interpreted by the Python Interpreter module on the fly. Therefore, the real-time performance is not so good if the demo needs hard RT requirement. For instance, in motion case, if we desired to do a set of movements with ALMotion module, it does not guarantee that joints values reach positions that there are determined in method call. Also, if movements were executed as developer determined, joints motion times are not also guaranteed. This means, if the developer wishes that robot arrive to a fixed position in a determined time, the robot can arrive to the position several seconds late. In our application, the real-time requirement of motion is soft, so the motion related behavior running on Nao is totally developed by Choregraphe.

2.2.4 Server

2.2.4.1 PHP \(^{(5)}\)

The major technology used to develop this server-side application is PHP. PHP is an open source tool to develop dynamic web sites. Current version of PHP is 5.3. PHP has a long list of features. Some of them are given below:

- PHP is licensed under GNU/GPL which allows the users to use it free of cost for commercial uses. The development cost of PHP is also less.
- PHP has any excellent ability to work with a lot of open source database management systems like MYSQL, MySQLi, firebird, Postgre SQL. In this application MySQL is used.
- PHP applications are secure, reliable and efficient.
- PHP has a big community consisting of experienced professionals all over the world and getting support is very easy.
- PHP can do anything that any other web development language can do. It is easy to write complex code in PHP. Learning PHP is also very easy.

2.2.4.2 MySQL \(^{(6)}\)

In our demo, we use database to share data like images and the content of receiving message. As a great friend of PHP, MYSQL is one of the best database solutions. MYSQL abbreviated as My Structured Query Language, is an open source database management system. MYSQL is written in C and C++. It has the ability to work on different platforms. It is the most popular database management tool. It is designed, developed and supported by MYSQL AB. Here are some features of MYSQL out of a rich features list:

- MYSQL is an Open Source Database Management System.
- Getting support is also very easy.
- MYSQL has a high performance.
- MYSQL provides excellent security features. It has powerful data encryption and decryption functions.
- MYSQL is compatible with several other technologies like PHP, Java, C++, Servlet, and
Servlets are the Java platform technology of choice for extending and enhancing Web servers. It provides a component-based, platform-independent method for building Web-based applications. Today servlets are a popular choice for building interactive Web applications. Here are some features of Servlets:

- Servlets are server and platform independent.
- Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. In my application, MySQL is used.
- Servlets can also access a library of HTTP-specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability, and crash protection.
- Servlets is free of charge, so the cost of development is very low.

Apache Tomcat is an open source software implementation of the Java Servlet and JavaServer Pages technologies. It is intended to be a collaboration of the best-of-breed developers from around the world. Apache Tomcat powers numerous large-scale, mission-critical web applications across a diverse range of industries and organizations. One of the most attractive features is it is also free.

The Apache HTTP Server Project is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server that provides HTTP services in sync with the current HTTP standards. Apache HTTP Server has been the most popular web server on the Internet since April 1996, and the latest released version is Apache HTTP Server 2.2.15. Since it supports PHP very well, Apache HTTP Server is a good complementary choice to Tomcat.

OpenCV is an open source computer vision library available from http://SourceForge.net/projects/opencvlibrary. The library is written in C and C++ and runs under Linux, Windows and Mac OS X. There is active development on interfaces for Python, Ruby, Matlab, and other languages. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. One of OpenCV’s goals is to provide a simple-to-use computer vision infrastructure that helps people build fairly sophisticated vision applications quickly. The OpenCV library contains over 500 functions that span many areas in vision, including medical imaging, security, camera calibration, stereo vision, and robotics. Two main sets of function are used in our application:

**XML and video file functions:** To simplify the file management, OpenCV ships with a set of functions to operate on video and xml files, including both reading and writing.

**Face Detection and Recognition:** face detection and recognition is such a common need that it is worth having a baseline technique that works fairly well in the computer vision library.
OpenCV implements the Haarclassifier face detection algorithm and PCA face recognition algorithm which works well in our applications.
Chapter 3

Use Cases

3.1 Overview

We define some use cases to help Nao interact with humans and try to exert Nao’s most potential. The architecture of use cases is shown in figure 3.1. We use the typical client-server architecture which is more familiar to us. Nao asks for service or call remote functions from Server. Mobile phone receives video stream via its web browser.

As is shown in Figure 3.1, there are two types of local applications. The first type like “Dance” and “PlayTaichi” are purely local which just depends on the Naoqi library. And the other one like “SMS” and “Location” are remote which means they not only use Naoqi library, but also call remote functions. Each remote application has a corresponding application running on Server.

Server is responsible for calling Web Service APIs, running remote modules, and streaming video to mobile phone. Server has three components which contain different applications - Nao Remote Module, Web API, and Streaming service. Figure 3.2 has a closer look at the architecture. The applications “Location”, “SMS”, and “Facebook” in Nao Remote Module are written in C++. Nao will use these applications to call web APIs such as Google Maps and Ericsson location APIs. However, we have introduced that web APIs usually support JSON or XML return. C++ doesn’t have a good support to cope with XML and JSON format. Fortunately, PHP dose parse XML and
JSON results very conveniently. Therefore, we build a website on Server so called “Web API” component. When Nao calls remote functions, they send HTTP request to our “Web API” site, and then “Web API” runs its PHP functions to call web APIs, parse the return, and store correct results into database. Here, we incorporate database within server in order to share the results among all components. No matter PHP or C++ have a good support to database. In this way, we can add more use cases to use the results and don’t need to change existing applications. What we need to do is just to query the data in the corresponding tables. The last remote module application is “Take Photo” which is used to save the pictures taken by Nao’s camera. “Streaming Service” is a web service built on Server which will stream those pictures saved by “Take Photo” to mobile phones that possess a web browser like safari, opera, or Firefox.

The following section will talk about all the developed use cases in detail.

3.2 MySQL Database

Database is used to share data among the corresponding applications in different components. Different application shares different tables. Figure 3.2 shows the database structure.

The first table is called “FaceRecog”. The information of the person who Nao can recognize his face is stored in FaceRecog. When Nao calls FaceRecog function and recognizes the person in front of it, the function will return the face id and then find his name via the number. In this way, we don’t need to cope with complex String type in C++, and we can easily add more faces in the future. Here is the code:

```sql
CREATE TABLE FaceRecog
(face id, face name)
```

Facebook table is used to share the image path between Facebook PHP page and Facebook remote module. Facebook remote module saves the picture taken by Nao’s camera and stores the image path into Facebook table. PHP page query the table in some period and if there is any new item inserted, PHP page will find the image via the path and upload the picture to Facebook. Here is the code:

```sql
CREATE TABLE Facebook
(
    image_id INT AUTO_INCREMENT,
    image_path TEXT,
    PRIMARY KEY (image_id)
);
```

Location application actually uses two tables. Locnumber table stores person’s name and phone number. When you ask Nao to locate some person, it will try to find his number in this table. Location table which stores the coordinate attained from Ericsson Location API is used to share the result between Location PHP page and Location remote module. Here is the code:

```sql
CREATE TABLE locnumber
(
    id INT AUTO_INCREMENT,
    number TEXT,
    name TEXT,
    time TEXT,
    PRIMARY KEY (id)
);
CREATE TABLE location
(
    id INT AUTO_INCREMENT,
    latlng TEXT,
    location TEXT,
    time TEXT,
    PRIMARY KEY (id)
);
```

Mobile table stores text messages and web command received from mobile phone. Here is the code:

```sql
CREATE TABLE Mobile
(
    id INT AUTO_INCREMENT,
    sender VARCHAR(20),
```
3.3 Nao local application

Nao local application is running on Nao’s ARM computer. It implements speech recognition, dancing, playing Taichi and calling various remote APIs. We develop local part in Choregraphe. As is shown in figure 3.3, the local application has three layers.

The first three boxes is the initialization layer. In this layer Nao enslave its motors and initialize its position. Nao can detect whether it is standing, if no, it will stand up automatically and say “I’m ready” to inform users the initialization is finished.

The second layer is the daemon layer. There are two boxes in this layer - FaceDetect and AudioReco. These two programs are running all the time like human’s eyes and ears. Just as the name implies, FaceDetect initialize the camera and detect faces. If faces show up in front of Nao, Nao will call the remote Face Recognition API and get the name of the face (if exists in database). AudioReco initialize the list of recognized words which are defined by programmers and the microphones locate in either side of Nao’s ears.

In our demo, AudioReco takes a list of commands which help users control Nao, including SayHello, Dancing, FaceTracking, Location, SMS, WhatCanNaoDo, Taichi, Facebook, and Goodbye. Each command corresponds with a box. All these boxes belong to the third layer. Each of the commands is corresponded to one robot scene. The outputs of AudioReco and FaceDetect are
changing in different scenes. We defined a variable RobotScene to store the number of the current scene:

- 1<--- SayHello
- 2<--- Taichi
- 3<--- WhatCanNaoDo
- 4<--- Goodbye
- 5<--- FaceTracking
- 6<--- Dancing
- 7<--- Location
- 8<--- Facebook

### 3.3.1 Robot Scenes

As is shown in Figure 3.3, there are many boxes with different names which indicate their function. In this section, we introduce the key code of the boxes one by one.

**Stiffness:** This box set the stiffness of all motors to 1, i.e., enslaves the motors, and then the step motors can work regularly. Here is the code snippet enslaving Nao’s head:

```python
self.chain = "Head"
self.duration = 1.0 # 1 second
ALMotion.gotoChainStiffness(self.chain, 1.0, self.duration, 1)
```

**Standup:** The robot stands up, if it is lying on its front or on its back. Nao also stands up from a sitting position. This box can be found in the box library in Choregraphe.

**Initiation:** Initiate the global variables. Here is the code snippet:

```python
ALMemory.insertData("StartAudioRecognition", 1, 0) # open audio recognition
ALMemory.insertData("StartFaceRecognition", 0, 0) # close face recognition
ALMemory.insertData("RobotScene", 0, 0) # set RobotScene to “listening to commands” which is default value
ALMemory.insertData("faceRecogResult", 0, 0) # the variable stores the result of the remote module FaceRecog
ALMemory.insertData("soundPositionToHead", "", 0) # this variable stores the position of the sound source
ALMemory.insertData("faceDirectionH", "", 0) # this variable is used by FaceTracking
ALMemory.insertData("faceDirectionV", "", 0) # this variable is used by FaceTracking
```

**FaceDetect:** This box works in SayHello and FaceTracking scenes, and it closes in other scenes to save CPU and memory resources. FaceDetect starts face detection extractor and subscribe to faces value. Depending on whether it sees a face or not, it will give the number of faces seen through onFaceDetected, or stimulate onNoFace. When it sees a face, it reads the value of “faceRecogResult”. If “faceRecogResult” is not zero, it means Nao remember the name of the person. It is to be noted that “faceRecogResult” is produced by the remote module FaceRecog
which will be discussed later. Here is the code snippet:

```
person = 0
count = 0
while person == 0 and count < 1000:  # the recognizing needs some time, so we use a counter here.
    person = self.memory.getData("faceDetectionResult",0)
    count = count + 1
```

**Face result:** If Nao remembers the person, this box will speak out the name of the recognized person. Here is the code snippet:

```
self.tts = ALProxy('ALTextToSpeech')  # get the proxy of 'ALTextToSpeech'
```

```
id = self.tts.post.say(name)  # speak the name.
```

**AudioReco:** This is the voice recognition box. Its recognizing words are different in different scenes. If RobotScene is “listening to commands”, the list of words is “nice to see you nao”, “see you next time”, “Can you play Taichi”, “what can you do”, “You are good!”, “you are great”, “Start face Tracking”, “tell me where I am”, and “show us your dance”. If RobotScene is “SayHello”, the list of words is “raise your head”, "look at right", "look at left", "bow your head", “That’s OK”, “You are good!”, and “you are great”. If RobotScene is “FaceTracking”, Nao can recognize "You are good!", "You are great!", "That's great", and “Stop Tracking”. Here is the code snippet:

```
if RobotSceneNum == 0:  # listening to command
    if(len(p) > 1):  # recognized some words exist
        if(p[1] >= self.threshold):  # the trust is higher than threshold
            self.onWord(p[0])  # activate output of the box
    elif RobotSceneNum == 1:  # say hello
        if(len(p) > 1):
            if(p[1] >= self.threshold):
                self.onWordScene1(p[0])  # activate output of the box
    elif RobotSceneNum == 5:  # face tracking
        if(len(p) > 1):
            if(p[1] >= self.threshold):
                self.onWordScene5(p[0])  # activate output of the box
else:  # other scenes
    pass  # do nothing
```

**Say hello:** this scene is activated by saying “nice to see you” to Nao. After hearing the command, the FaceDetect and AudioReco are started at once. Meanwhile, we make a proxy to ALAudioSourceLocalization module to get the sound energy received by each microphone and calculate the direction of the sound source, i.e. your face. We defined a function to process the energy of microphone data and give Nao the instructions about how to turn its head. Here is the code:

```
def processMicrophoneData(self, pDataName, pValue, pMessage):
    # get the energy of each microphone except the rear one which is disturbed too much by noise
    self.p.append(self.memoryProxy.getData("ALAudioSourceLocalization/D/Energy", 0));
```
self.p.append(self.memoryProxy.getData("ALAudioSourceLocalization/1/Energy", 0));
self.p.append(self.memoryProxy.getData("ALAudioSourceLocalization/2/Energy", 0));

if (min(self.p) > self.threshold_energy):  # the energy should be larger than a threshold.
    newpos = ""
        newpos = "raise your head"  # sound comes from top
        newpos = "look at right"     # from right
        newpos = "look at left"      # from left
    if(newpos != ""):  
        self.onSoundDetected(newpos)   # activate the output to turn Nao’s head

At the same time, you can tell Nao where the sound is by saying "raise your head", "look at right", "look at left", and "bow your head". As soon as Nao receives an instruction, Nao turns its head according to it. After Nao’s camera turns to your face, Nao will work as what we have described in FaceDetect module.

**Taichi/Dancing/Goodbye**: These three scenes are very similar. All of them consist of a sequence of movements with background music.

To play music, first we upload the MP3 or WAV file to Nao’s flash memory. Then, make a proxy to the module ‘ALAudioPlayer’ and call playFile method. The code snippet is:

```python
self.player = ALProxy('ALAudioPlayer')
id = self.player.post.playFile(MusicFile)
```

As introduced above, with the help of Nao 3D panel, it is very easy to make various movement sequences. What we need to think is how to balance the load of motors, particularly the leg motors. This can be done by adjusting the Support Mode and Balance Mode of Nao.

Two functions are available to change or get the Actual Support Mode during Nao’s dancing or Taichi show:

```python
void setSupportMode(int pSupportMode);
int getSupportMode();
```

Two functions are available to change or get the current Balance Mode:

```python
void setBalanceMode(int pBalanceMode);
int getBalanceMode();
```

While Nao is dancing or playing Taichi, we need to change the support mode and balance mode properly according to the movement, in case it falls down or breaks its motors. It is also necessary to choreograph very carefully.

**FaceTracking**: this scene is activated by saying “Start Tracking”. FaceTracking is implemented by subscribing to two ALMemory variables, "faceDirectionH" and "faceDirectionV". After Nao finds
your face in the image captured by Nao’s camera, the remote module FaceRecog outputs not only the name of the people, but also the offset of the face’s position to the image center. “faceDirectionH” represents the horizontal offset, and “faceDirectionV” is the vertical one. Nao will turn its head with the change of “faceDirectionH” and “faceDirectionV”. In this way Nao can track person’s face. Here is the code snippet:

Subscribe to ALMemory. Set the period and the callback function.

```python
self.memoryProxy.subscribeOnData("faceDirectionH", self.getName(), "", "processFaceTrackH");
self.memoryProxy.subscribeOnDataSetTimePolicy("faceDirectionH", self.getName(), self.period);
self.memoryProxy.subscribeOnData("faceDirectionV", self.getName(), "", "processFaceTrackV");
self.memoryProxy.subscribeOnDataSetTimePolicy("faceDirectionV", self.getName(), self.period);
```

The horizontal callback function is defined like this:

```python
def processFaceTrackH(self, pDataName, pValue, pMessage):
    # read the value of horizontal coordinate
    directionH = self.memoryProxy.getData("faceDirectionH", 0)
    if pValue in ['left', 'right']:
        # after reading the value, clear it
        self.memoryProxy.insertData(pDataName, "", 0)
        # turn head according to the value
        self.TurnHead(pValue)
```

The vertical callback function is defined like this:

```python
def processFaceTrackV(self, pDataName, pValue, pMessage):
    # pValue takes the value of “faceDirectionV”
    # read the value of vertical coordinate
    directionH = self.memoryProxy.getData("faceDirectionV", 0)
    if pValue in ['up', 'down']:
        # after reading the value, clear it
        self.memoryProxy.insertData(pDataName, "", 0)
        # turn head according to the value
        self.TurnHead(pValue)
```

**Location/SMS/Facebook:** In these scenes, Nao is supposed to call their correspondent remote methods. Here is the local part code snippet:

For Location module,

```python
# create a proxy to location module where “location” is the name of remote module.
self.locProxy = ALProxy("Location")
```
# get the coordinate of the target,
self.coordinate = self.locProxy.getLatlng("Location", number)

# call the remote API getLocation. This function returns the name of the city according to the coordinate
self.location = self.locProxy.getLocation("Location", float(latitude), float(longitude))

For SMS module, the code of sending message is like this:

# create a proxy to SMS module where “SMS” is the name of remote module
self.smsProxy = ALProxy("SMS")

# call the remote API sendSMS. This function returns “ok”, if works properly.
self.sms = self.smsProxy.sendSMS("SMS", "ok")

For Facebook module,

# create a proxy to Facebook module
self.fbProxy = ALProxy("facebook")

# take picture using Nao’s camera
self.fbProxy.saveImageRemote("photos\uploadPic", "jpg")

# call uploadPic function to upload the picture to Facebook
self.fbProxy.uploadPic("Hello Facebook ","Nao&Nao&uploadPic.jpg")

3.4 Web API application

As is required, in order to develop Ericsson Labs and Facebook application, we first register a Call-back URL and build a website handling incoming HTTP GET/POST requests. The web server we adopted is Tomcat 6.0 handling Java Servlets code and Apache HTTP Server 2.2 handling PHP code.

3.4.1 SMS

In our application users can control Nao’s movement by text message and Nao will send a confirm message to the user. The format of text is defined like this:

<table>
<thead>
<tr>
<th>Service identifier</th>
<th>Command</th>
<th>Parameter</th>
</tr>
</thead>
</table>

- Service identifier, as explained before, is the short code allowing you to receive messages sent from mobile phones.
- Command tells Nao what to do next. It may take several parameters.

Here is the command list:

**Move type:**
- walk straight -> walk straight 0.5m
- body turn left -> body turn left
body turn right -> body turn right
head turn left-> head turn left
head turn right-> head turn right
crouch-> crouch
stand up-> stand up

Dance type:
taichi-> play Taichi
pop dance-> Nao start to dance

Interaction type:
Hello-> say hello to Nao and Nao will welcome you.

Facebook type:
uploadPic -> upload picture to Facebook

Once received a message, the message is stored in the database. Meanwhile, Nao is polling to check if there is new item added in database. If yes, Nao parses the command and execute, otherwise, Nao does nothing.

3.4.1.1 Implementation
The key parts are to define the callback function to receive text message and to call the Ericsson Labs SMS API to send confirm message. We use Servlets technology to implement the server part. For receiving, we define a class MessageReceivedServlet which extends HttpServlet class. MessageReceivedServlet overrides the doGet method. First, it extracts the useful information from the request. Then, it makes a connection with database and stores the result.

```java
public class MessageReceivedServlet extends HttpServlet {

    // define the database parameters
    public static final String DBDRIVER = "org.gjt.mm.mysql.Driver";
    public static final String DBURL = "jdbc:mysql://localhost:3306/nao";
    public static final String DBUSER = "root";
    public static final String DBPASS = "root";
    public static final int STORE_NUM = 10;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {

        // Extract request parameters according to SMS Send & Receive API.
        String from = req.getParameter("from");
        String time = req.getParameter("time");

```
String message = req.getParameter("message");
String hmac = req.getParameter("hmac");

// database resources
Connection ct = null;
Statement sm = null;
ResultSet rs = null;

// catch the exception
try {
    Class.forName(DBDRIVER);
    ct = DriverManager.getConnection(DBURL, DBUSER, DBPASS);
    sm = ct.createStatement();

    // insert item
    sm.executeUpdate("INSERT INTO messages (source, msg, time) "
    + "VALUES (" + from + "," + message + "," + time + ")");

    // delete unnecessary items (maximum STORE_NUM)
    rs = sm.executeQuery("select * from messages");
    if (rs.last()) {
        if (rs.getRow() > STORE_NUM) {
            sm.executeUpdate("DELETE FROM messages WHERE messageId < " + rs.getString(1));
        }
    }
} catch (Exception ex) {
    System.out.println("database error: " + ex);
} finally {

    // close all the unnecessary resources
    try {
        if (rs != null) rs.close();
        if (sm != null) sm.close();
        if (ct != null) ct.close();
    } catch (SQLException ex) {
        System.out.println("Exception: " + ex);
    }
}

// Reply back to the SMS Send & Receive server.
res.setContentType("text/html");
ServletOutputStream respToServer = res.getOutputStream();
respToServer.println("Thanks for the message!");
For sending, we define HttpHelperUtil class with sendSynchronousHttpGetRequest method to send HTTP GET request. SendSynchronousHttpGetRequest takes only one parameter indicating the URL and return the response from server.

```java
public class HttpHelperUtil {
    public static String sendSynchronousHttpGetRequest(String url)
        throws MalformedURLException, IOException {

        // Opens a connection on the URL
        HttpURLConnection connection = (HttpURLConnection) new URL(url).openConnection();

        // Creates a reader on the input stream of this connection
        BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getInputStream()));

        // Retrieves HTTP response
        StringBuilder result = new StringBuilder();
        try {
            String inputLine = null;
            while ((inputLine = reader.readLine()) != null)
                result.append(inputLine);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                reader.close();
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }
        return result.toString();
    }
}
```

3.4.2 Location

As is explained above, as long as the target mobile phone accepts the consent, we can send a HTTP POST request to Ericsson Labs server, and then the server will locate the phone and return the longitude and latitude. In our application, the user can send text message to get other’s location or ask Nao directly by speech. However, although the accuracy is acceptable, the main problem is that the result is not direct enough. In order to get a more user-friendly and useful location information, we use Google Maps Service to interpret the longitude and latitude to the
name of place. The result will be stored into Location table in Nao database.

### 3.4.2.1 Implementation

There are two steps to follow:

First, we make a HTTP POST request with the target mobile phone number to Ericsson Location server, extract the longitude and latitude from the response, and store the value into the database. The step is implemented in PHP

```php
<?php

// Create a request for the end-user location
$url = "http://mwl.labs.ericsson.net/mwlservice/resource/";
$msisdn = 'phone_number';
$fields = array(
    'msisdn' => $msisdn,
    'devKey' => 'DuLaTs7n8gyC8sDcgVbgHsHsyM7u1V6qlnreern',
    'applicationID' => 'EricNao' );
$create_url = $url . 'location/getLocationByMsisdn';

// Make the request
$response = http_post_fields($create_url, $fields, NULL, array("timeout"=>10), $info);

// Extract the body from the response and print it
.HTTP_obj = http_parse_message($response);

// Output the results
.print("Response = ", $HTTP_obj->body);

// Database info
$dbhost = "localhost:3306";
$dbuser = "root";
$dbpass = "root";
$dbname = "nao";

// Connect to mysql
$dbconn = mysql_connect($dbhost,$dbuser,$dbpass)or die('Error, connect db failed');
mysql_select_db($dbname,$dbconn)or die('Error, select db failed');

// Make sure the result contains the coordinate. If it's correct, insert the value into database
$result = mysql_real_escape_string("latitude", $dbconn);
if ($result == "latitude") {
    mysql_query("INSERT INTO messages ("source", "msg", "time") VALUES
```
After we get the response from Ericsson server, we make a HTTP GET request with the longitude and latitude to Google Maps server. The response format is XML. We can easily find out the name of the place, and store the value into the database.

```php
// get the coordinate from GET request
$latlng = $_GET['latlng'];

//Create a request for the end-user location

// make a HTTP request and save the response into $output
$output = file_get_contents($url);
$result=json_decode($output);

......
// the database related code is similar with the first step. Here we omit it.
......
```

3.5 Nao Remote Module

Nao can directly call the functions of remote modules which are always run in PC. There are two benefits to do so. Firstly, some code can only be run in a PC or other platform, so remote module can help to fix this problem. Secondly, for the sake of performance, we sometimes need to run large-computation function or procedure in a more powerful PC. When we meet with this situation, we can wrap this code in a function of remote procedure, and call the function from Nao. In our prototype, we developed three remote modules - Location, Facebook, and Face Recognition. The first two work as an interface to call the web service API. And the last one helps to save the CPU and the memory of Nao. In this section, we will discuss remote module in detail.

3.5.1 Location

In section 3.2, we introduce how to call Ericsson Location and Google Maps API in PHP. However, Nao doesn’t support PHP code directly. Fortunately, we can build Location module as the interface to use the PHP function. The way in which Location module works has two steps:

**Step 1:** Location module sends a HTTP GET request with the target mobile phone number to the PHP page in our server. As soon as our server receives the request, it calls Ericsson Location API to fetch the longitude and latitude of the target, and return the value to
Location module.

**Step 2:** Location module call Google Maps API with the coordinate. Google Maps Service will return the location information in JSON. Our module will parse the result and tell Nao the name of the city.

We implemented this part by the Windows network library in C++.

Here is the core code of this step:

```cpp
// this function send a HTTP GET request to urlAddress
bool location::SendURLRequest(const std::string& urlAddress)
{
    HINTERNET hSession=NULL, hConnect=NULL, hRequest=NULL,
    hOpenUrlHandle=NULL;
    hSession = InternetOpen( TEXT("HTTPGET"),INTERNET_OPEN_TYPE_DIRECT,NULL,NULL,0);
    hOpenUrlHandle = InternetOpenUrlW(hSession, (LPCWSTR)urlAddress, NULL, 0, INTERNET_FLAG_RAW_DATA,0);
    return true;
}

// this function can be called by Nao to get the coordinate of the target. pNum is the target phone number
AL::ALValue location::getLatlng(const std::string& pNum )
{
    if(!SendURLRequest(url)){
        std::cout << "LocationModule say : no response" << std::endl;
    }else{
        // get the location information from the database.
        locationInfo = dbGetLatlng();
    }
    return locationInfo;
}

// this function can be called by Nao to get the name of the city where the target stay. pLat and pLng is the coordinate of the target
AL::ALValue location::getCityName ( const AL::ALValue& pLat,const AL::ALValue& pLng )
{
    if(!SendURLRequest (url)){
        std::cout << "LocationModule say : no response" << std::endl;
    }else{
```
3.5.2 Face Recognition

FaceRecog module starts when Nao finishes initiation. This module registers itself in Naoqi and then communicates with Nao via a broker. The recognition algorithm we used is a well-known PCA-based face recognition method 'Eigenface'. This approach treats face recognition as a two-dimensional recognition problem, taking advantage of the fact that faces are normally upright and thus may be described by a small set of 2-D characteristic views. Face images are projected onto a feature space (“face space”) that best encodes the variation among known face images. The face space is defined by the “Eigenface”, which are the eigenvectors of the set of faces. Before start FaceRecog module, we need take a training set of face images and calculate the eigenfaces. Then, save the eigenfaces and its corresponding number in an XML file.

FaceRecog module works in the following steps:

- Fetch images captured from Nao’s either top forehead or bottom forehead camera. And load the eigenfaces file which is prepared before.
- When a new face image is encountered, calculate a set of weights based on the input image and the M eigenfaces by projecting the input image onto each of the eigenfaces.
- Determine if the image is a well-known face by checking to see if the image is sufficiently close to any eigenface.
- Return the number of the eigenface which matches the new face and save the result in ALMemory “faceRecogResult”.

3.5.2.1 Implementation

This section shows the core code to implement FaceRecog module. In this module we use the functions of OpenCV2.0 library.

Fetch the images from Nao’s camera:

```cpp
locationInfo = dbGetLocation();
}
return locationInfo;
}

3.5.2 Face Recognition

FaceRecog module starts when Nao finishes initiation. This module registers itself in Naoqi and then communicates with Nao via a broker. The recognition algorithm we used is a well-known PCA-based face recognition method 'Eigenface'. This approach treats face recognition as a two-dimensional recognition problem, taking advantage of the fact that faces are normally upright and thus may be described by a small set of 2-D characteristic views. Face images are projected onto a feature space (“face space”) that best encodes the variation among known face images. The face space is defined by the “Eigenface”, which are the eigenvectors of the set of faces. Before start FaceRecog module, we need take a training set of face images and calculate the eigenfaces. Then, save the eigenfaces and its corresponding number in an XML file.

FaceRecog module works in the following steps:

- Fetch images captured from Nao’s either top forehead or bottom forehead camera. And load the eigenfaces file which is prepared before.
- When a new face image is encountered, calculate a set of weights based on the input image and the M eigenfaces by projecting the input image onto each of the eigenfaces.
- Determine if the image is a well-known face by checking to see if the image is sufficiently close to any eigenface.
- Return the number of the eigenface which matches the new face and save the result in ALMemory “faceRecogResult”.

3.5.2.1 Implementation

This section shows the core code to implement FaceRecog module. In this module we use the functions of OpenCV2.0 library.

Fetch the images from Nao’s camera:

```cpp
// Check that a video module has been registered.
if (!fRegisteredToVim)
  return false;
ALValue results;
results = (fCamProxy->call<ALValue>("getImageRemote", fGvmName));
if (results.getType() != ALValue::TypeArray && results.getSize() != 7)
  return false;
const int size = results[6].getSize();
// You can get some image information that you may find useful.
const int width = (int) results[0];
const int height = (int) results[1];
const int nbLayers = (int) results[2];
const int colorSpace = (int) results[3];
```
const long long timeStamp = ((long long)(int)results[4])*1000000LL + ((long long)(int)results[5]);
const int seconds = (int)(timeStamp/1000000LL);

// Set the buffer we received to our IplImage header.
flplImageHeader->imageData = (char*)(results[6].GetBinary());

Load eigenfaces file and assign value to global variables:
CvFileStorage * fileStorage;
int i;

// create a file-storage interface
fileStorage = cvOpenFileStorage( "facedata.xml", 0, CV_STORAGE_READ );
if ( !fileStorage ){
  cout<"Can't open facedata.xml\n";
}
nEigens = cvReadIntByName(fileStorage, 0, "nEigens", 0);
nTrainFaces = cvReadIntByName(fileStorage, 0, "nTrainFaces", 0);
*pTrainPersonNumMat = (CvMat *)cvReadByName(fileStorage, 0, "trainPersonNumMat", 0);
eigenValMat  = (CvMat *)cvReadByName(fileStorage, 0, "eigenValMat", 0);
projectedTrainFaceMat = (CvMat *)cvReadByName(fileStorage, 0, "projectedTrainFaceMat", 0);
pAvgTrainImg = (IplImage *)cvReadByName(fileStorage, 0, "avgTrainImg", 0);
eigenVectArr = (IplImage **)cvAlloc(nTrainFaces*sizeof(IplImage *));
for(i=0; i<nEigens; i++){
  char varname[200];
  sprintf( varname, "eigenVect_%d", i );
  eigenVectArr[i] = (IplImage *)cvReadByName(fileStorage, 0, varname, 0);
}

// release the file-storage interface
cvReleaseFileStorage( &fileStorage );

Projecting the input image onto each of the eigenfaces
float * projectedTestFace = 0;

// allocate nEigens memory unit to save the projected values
projectedTestFace = (float *)cvAlloc( nEigens*sizeof(float) );

// project the test image onto the PCA subspace. cvEigenDecomposite is in OpenCV's library
cvEigenDecomposite(          
  image,
where projectedTestFace is the eigenfaces distance of the input image.

Find the number of the nearest eigenface:

```c
int i, iTrain, iNearest = 0;
for(iTrain=0; iTrain<nTrainFaces; iTrain++)
{
    double distSq=0;
    for(i=0; i<nEigens; i++)
    {
        float d_i = projectedTestFace[i] -
                    projectedTrainFaceMat->data.fl[iTrain*nEigens + i];
        distSq += d_i*d_i; // Euclidean
    }
    if(distSq < leastDistSq)
    {
        leastDistSq = distSq;
        iNearest = iTrain;
    }
}
```

// if the least distance is still larger than a certain distance, the image are not saved in face database.
```c
if(leastDistSq > LEAST_DIS_SQ)
    iNearest = -1;
```

where iNearest represents the number of the nearest eigenface.

### 3.6 Interaction with Mobile Phone

#### 3.6.1 Video streaming

Video streaming is the prerequisite of building remote control panel of Nao. It is kind of web service application based on the AJAX and base64 en/decoding technique. This application implements that mobile phones or devices are able to display the video captured by Nao. However, the mobile devices have to be equipped with a web browser which supporting Base64 decoding and javascript, e.g., Firefox, Opera, and Safari. Video streaming works in the following steps:

- Nao’s camera captures images continuously and save every fps*videolength images into an AVI file. Fps and videolength are defined by users which depend on the network
condition.

• The video file is converted to a string array of Base64 encoded images and the array is saved into an XML file.

• The server supplies two web service methods which can be called by mobile phone browsers, getXMLBase64ClipInfo and getXMLBase64Frames. GetXMLBase64ClipInfo can get the video basic info like fps and length. GetXMLBase64Frames is used to get the video frames’ code one by one.

• Browser initiates an XML HTTP request to the web service method GetXMLBase64ClipInfo to fetch the clip info of the video, i.e., fps and length. Then send XML HTTP request to GetXMLBase64Frames periodically to get the frames. The period is set according to the fps of video.

• Decode the Base64 array and repaint the images on the screen.

3.6.1.1 Implementation

In this section we show the key code to implement the application.

In the server the core code is how to save video file and encode the video files. Here is the code snippet:

Save video file:

```c
// create the video writer with the basic settings, fps, name, file type, etc.
CvVideoWriter* writer = cvCreateVideoWriter( videofilename, CV_FOURCC('D', 'I', 'V', 'X'), videofps, cvSize(image->width, image->height ), isColor);
frameCounter = 0;
while( frameCounter < videofps*videolength ){
    if( !videostreamImg )
        videostreamImg = cvCreateImage(cvSize( image->width, image->height ),
                                        IPL_DEPTH_8U, 3);
    cvCopy( image, videostreamImg );
    cvWriteFrame( writer, videostreamImg);
    frameCounter++;
}
```

// this is an essential step. Otherwise, the video file can’t be converted.
cvReleaseVideoWriter(&writer);

Image encoding:

MediaDetClass type is used to get the video info:

```c
MediaDetClass mediaClass = new MediaDetClass();
_AMMediaType mediaType;
```

Output the video info:

```c
outFrameRate = mediaClass.FrameRate;
mediaType = mediaClass.StreamMediaType;
```
VIDEOINFOHEADER videoInfo = (VIDEOINFOHEADER)Marshal.PtrToStructure(mediaType.pbFormat, typeof(VIDEOINFOHEADER));
outClipSize = new Size(videoInfo.bmiHeader.biWidth, videoInfo.bmiHeader.biHeight);
outStreamLength = mediaClass.StreamLength;
mediaType = mediaClass.StreamMediaType;

Initiate image buffer:
int bufferSize=0;
byte* ptrRefFramesBuffer=null;

// Get the size of a frame in order to dedicate enough memory to store the Bitmaps in our pointer
mediaClass.GetBitmapBits(0.0, ref bufferSize, ref *ptrRefFramesBuffer, outClipSize.Width, outClipSize.Height);

// Dedicate memory to the pointer
ptrRefFrameBufSize = System.Runtime.InteropServices.Marshal.AllocHGlobal(bufferSize);
ptrRefFramesBuffer = (byte*)ptrRefFrameBufSize.ToPointer();

Get one frame:
//Get the Bitmaps
mediaClass.GetBitmapBits( currentStreamPos, ref bufferSize, ref *ptrRefFramesBuffer, outClipSize.Width, outClipSize.Height);
vdeoBitmaps.RotateFlip(RotateFlipType.Rotate180FlipX);

// Save the Bitmaps somewhere in the (managed) memory
vdeoBitmaps.Save(memory, System.Drawing.Imaging.ImageFormat.Jpeg);

Convert the frame to Base64 string:
string strFrame = System.Convert.ToBase64String(memory.ToArray());

Release pointer and clean up memory:
System.Runtime.InteropServices.Marshal.FreeHGlobal(ptrRefFrameBufSize);
memory.Close();
memory.Dispose();
return strFrameArray;

For browser decoding, the most important code is how to make the XML HTTP request and decode the Base64 string in javascript.
Make XML HTTP request:

```javascript
request= new XMLHttpRequest();

// define the request listener
request.onreadystatechange = ReqChangeListtener;
var requestTail = "strFileName="+videoFileName+".avi";

// get the XMLBase64ClipInfo. If get frame, change "getXMLBase64ClipInfo" to "getXMLBase64Frames"
AJAXVideoPlayer.request.open("POST", "/videostreamtest/Service.asmx/getXMLBase64ClipInfo","HTTP/1.1");

// set the POST request header
request.setRequestHeader("Host","flyinheart827.gicp.net");
request.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
request.setRequestHeader("Content-Length", requestTail.length.toString());
request.send(requestTail);
```

Decode Base64 string:

```javascript
// VideoScreen is the image object of HTML where the frames will be shown.
// It is defined like this,
// <img id=VideoScreen" class=imgVideScreen" style= "display:inline;" src= "alt="view" />
VideoScreen.src ="data:image/gif;base64," +strBase64ImageData;
```
Conclusion

It is a trend that all electronic devices are on internet and mobile network in the future. We can call our refrigerator to set its temperature to 0°C. Only sending text message is enough to pay all your bills. Your glasses like a screen will show you what you want to watch! The scene in scientific film will come up in reality. Nowadays we had taken a first step in accomplishing our dream. This work introduces GSMA OneAPI standard and one of its implementations - Ericsson IPX. A prototype was built on Nao robot platform using Ericsson Labs API and other proven web services API like Google Maps APIs and Facebook APIs.

The Architecture of our prototype is a kind of client-server system. Nao Local applications call remote functions which are running on Server. Server consists of PHP website, database, C++ remote application and C# video stream service. PHP website is used to make a HTTP request to web APIs server, parse the return, and store the proper results into database. Database works as a shared memory between web API applications and remote applications. C++ remote applications connect Server and Nao. Those remote applications register corresponding functions at Nao’s main broker, and then Nao can call these functions like local functions. Remote applications will query the latest inserted item from corresponding database table and transfer the result to Nao local application. Video stream service will publish the picture taken by Nao’s camera to internet at a changeable frame rate. Frame rate changes with the network condition.

Our use cases try to bring out the best in Nao and make Nao more interesting. You can interact with Nao, such as let him tell you where you or your friends are, dance for you, or upload what he is watching to Facebook.

During the prototype development several challenges have been faced. The followings are considered the most noteworthy:

- Share data among different programme languages. Our prototype used PHP, C++, Python, and Javascript to develop applications. How to integrate all of them is the trickiest problem at the beginning. Fortunately, all of them have a good support to database like MySQL. MySQL operates as a “shared memory” among those applications. Our application won’t be called frequently, so we don’t need to consider the “shared memory” speed and race-conditions too much.

- Another noteworthy problem caused by programme languages is that it is easy to find we miss some lib file or forget to install some module. Because not all of the necessary modules are installed as the default settings, we should download it and reinstall it, such as PHP http module. And we also need to manually add some lib file to our IDE, in order to support the programme.

Further developments for this work would most probably be extensions to use cases. Artificial intelligence can let Nao think by himself and response. We shall also add more use cases using other web service APIs, such as twitter. With the improvement of Nao’s operating system Naoqi, we shall try to transplant our programme from current version to the new one. More local modules will be added, and much faster response speed of speech recognition will be available.
Bibliography