Research Internship Report

25/04/2016 ~ 25/09/2015

Deep Learning and Reality Gap

Author:

FENG Siyao

A report submitted in fulfilment of the requirements
for the degree of Master in Computer Science

at

Telecom Paristech

September 2016

Supervisors

Scientific Supervisors:

Michele SEBAG

Head of Équipe A-O - LRI

Tel : 01.69.15.42.33 E-mail : sebag@lri.fr

Academic Supervisor:

Albert BIFET

Associate Professor in Big Data

Télécom ParisTech

E-mail : albert.bifet@gmail.com

Acknowledgements

The internship opportunity I had with TAO group was a great chance for learning and for research experience. It's here at TAO I strengthened my willing to future research careers. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am sincerely grateful for having a chance to meet so many wonderful people who led me through this five-month internship.

First and foremost, I would like to express my sincere gratitude to my supervisor Mme. Michele SEBAG, for the continuous support of my internship, for her patience, motivation, and immense knowledge. Her guidance helped me in all the time of research and writing of this report. I could not have imagined having a better superadvisor and mentor for my internship.

My sincere thanks also goes to M. Gaetan Marceau Caron, who provided me his knowledge and experience and precious time, only with his help I can get better work enviroment.

Last but not the least, I would like to thank all the TAO team members. Thank you for the fabulous leisure time we had together!

Contents

List of Figures

Chapter 1

Introduction

Deep learning is part of a broader family of machine learning methods based on learning representations of data. In the recent years, Artificial Neural Nerwork has achieved a success in the domains like Computer Vision, Natural Language Processing.

Research in this area attempts to make better representations and create models to learn these representations from large-scale unlabeled data. Some of the representations are inspired by advances in neuroscience and are loosely based on interpretation of information processing and communication patterns in a nervous system, such as neural coding which attempts to define a relationship between various stimuli and associated neuronal responses in the brain.[1]

In the field of neuroscience, the data of electroencephlogram data(EEG Data) are being used in the brain computer interface(BCI), for example, using the EEG data to control the wheelchair or the gamestick. But they are facing the problem that the EEG data has a high variability problem. The data is not only depends on the subject, but also depends on the session. Which is to say that the EEG data collected in the morning and in the evening from the same subject is different. So it asks to train a model depends on a subject and then to adjust the model for different session.

Domain Adaptation[2][3] is a field associated with machine learning and transfer learning. This scenario arises when we aim at learning from a source data distribution a well performing model on a different (but related) target data distribution. For instance, one of the tasks of the common spam filtering problem consists in adapting a model from one user (the source distribution) to a new one who receives significantly different

emails (the target distribution). Note that, when more than one source distribution is available the problem is referred to as multi-source domain adaptation.[4]

In this internship, I work under the supervision of Dr. Michele Sebag and Dr. Gaetan Marceau Caron at Laboratoire de Recherche en Informatique (LRI), Gif-sur-Yvette, France. We aim at using domain adaptation to eliminate the variability of the EEG data.

1.1 Outline

In the first part(chapter 2) the contexte and the objectif of this internship will be introduced, the detail of EEG data, domain adaptation will be explained.

chapter 3 will show the state of the art on the field of deep learning or more presicely the domain adaptation.

The following chapter(chapter 4) focuses on the framework of this internship, the program I used to train the model.

chapter 5 will give you the experiments I have done and the results achieved. Some comments will also follow the results.

The final chapter 6 will be the discussion of the experiments results of using domain adaptation on the EEG data to eliminate the variability problem. Then the perspective of the subject will be given.

Chapter 2

Context and Objective

In this chapter, I will first introduce the context of this internship which is the Brain Computer Interface (BCI) and the new deep learning technique Domain Adaptation. From the problems identified for the EEG data, we propose the approach by using Domain Adaptation to eliminate the variabilty problem in the EEG data.

2.1 Context

2.1.1 BCI

Brain Computer Interface Brain-computer interface is a method of communication based on neural activity generated by the brain and is independent of its normal output pathways of peripheral nerves and muscles. The neural activity used in BCI can be recorded using invasive or noninvasive techniques. The goal of BCI is not to determine a person's intent by eavesdropping on brain activity, but rather to provide a new channel of output for the brain that requires voluntary adaptive control by the user.[5] The potential of BCI systems for helping handicapped people is obvious. There are several computer interfaces designed for disabled people.[6] Most of these systems, however, require some sort of reliable muscular control such as neck, head, eyes, or other facial muscles. It is important to note that although requiring only neural activity, BCI utilizes neural activity generated voluntarily by the user. Interfaces based on involuntary neural activity, such as those generated during an epileptic seizure, utilize many of the same components

and principles as BCI, but are not included in this field. BCI systems, therefore, are especially useful for severely disabled, or locked-in, individuals with no reliable muscular control to interact with their surroundings.

But there's a main limitation for BCI is that the EEG data have a high inter-variability and intra-variability. EEG data are highly dependent on subject, so it requires to train the BCI on each subject. Also, it's highly dependent on session, EEG collected are varies from different time(like morning and evening), and even during a same work session, so it also requires to train on each subject for each session, which is difficult to realize and redundant. Thus we should find an approach to eliminate the subject-independent and session independent problem of EEG data.

2.1.2 Domain Adaptation

Top-performing deep architectures are trained on massive amounts of labeled data. In the absence of labeled data for a certain task, domain adaptation often provides an attractive option given that labeled data of similar nature but from a different domain (e.g. synthetic images) are available. The paper by [Ganin et al. 2015][7] propose a new approach to domain adaptation in deep architectures that can be trained on large amount of labeled data from the source domain and large amount of unlabeled data from the target domain (no labeled target domain data is necessary).

As the training progresses, the approach promotes the emergence of "deep" features that are

1. Discriminative for the main learning task on the source domain
2. Invariant with respect to the shift between the domains.

The paper[7] shows that this adaptation behavior can be achieved in almost any feed-forward model by augmenting it with few standard layers and a simple new **gradient reversal layer**. The resulting augmented architecture can be trained using standard back-propagation. Overall, the approach can be implemented with little effort using any of the deep-learning packages. The method performs very well in a series of image classification experiments, achieving adaptation effect in the presence of big domain shifts and outperforming previous state-of-the-art on Office datasets.

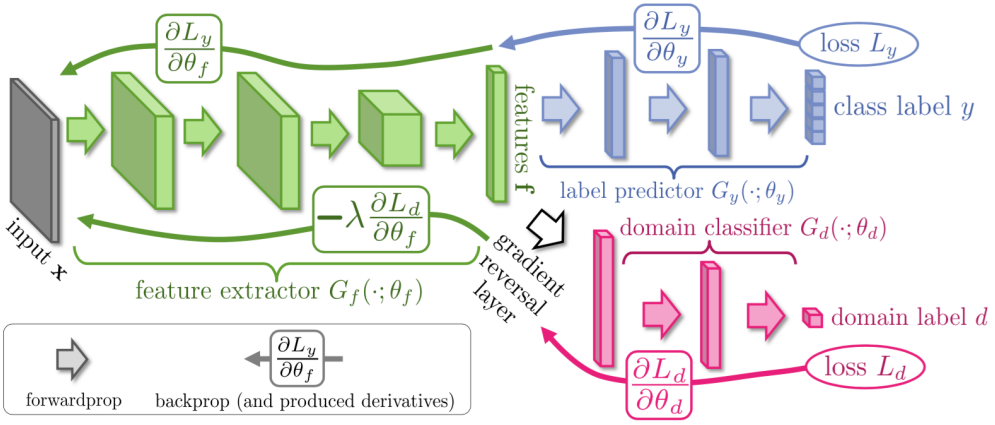

FIGURE 2.1: Domain Adaptation Model Architecture

This method involves two domains, the source domain and the target domain. For example, the source domain can be MNIST dataset (handwritten digit database) and the target domain can be the SVHN dataset (The Street View House Numbers). Formally, the source is described through a training set, made of (instance, label) pairs:

$$\mathcal{S} = (x_i, y_i), x_i \in X, y_i \in Y, i \in [1, n]$$

where X is the instance space (e.g. the vectors of pixel values) and Y is the label space (e.g. the digit numbers). The target is described through a test set made of instances:

$$\mathcal{T} = x'_i, x'_i \in X, i \in [1, n]$$

The general goal is to build a classifier h for the source domain and for the target domain **without** gathering the target doamin labels.

The paper has proposed a model for the domain adaptation as Figure 2.1. The proposed architecture includes a deep feature extractor (green) and a deep label predictor (blue), which together form a standard feed-forward architecture. Unsupervised domain adaptation is achieved by adding a domain classifier (red) connected to the feature extractor via a gradient reversal layer that multiplies the gradient by a certain negative constant during the backpropagationbased training. Otherwise, the training proceeds in a standard way and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples). Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features.

During the learning stage, we aim to minimize the label prediction loss on the annotated part (i.e. the source part) of the training set, and the parameters of both the feature extractor and the label predictor are thus optimized in order to minimize the empirical loss for the source domain samples. This ensures the discriminativeness of the features $\mathbf{f}$ and the overall good prediction performance of the combination of the feature extractor and the label predictor on the source domain. At the same time, we want to make the features $\mathbf{f}$ domain-invariant which is we want the representation of the two domains to be similar.

At training time, in order to obtain domain-invariant features, we seek the parameters of the feature mapping that maximize the loss of the domain classifier (by making the two feature distributions as similar as possible), while simultaneously seeking the parameters of the domain classifier that minimize the loss of the domain classifier. So as we can see, the parameter λ of the Gradient Reversal Layer controls the trade-off between the two objectives that shape the features during learning.

2.2 Objective

After the problem identified of the EEG data using in the BCI (high inter-variability and high intra-variability) .the objective of this internship thus is to build a subject-dependent, session-dependent representations of the EEG data. The sought representation must achieve a lossy compression of the signal to enabling the reconstruction of the brainwave data in order to further use in the BCI, besides, the representation of the signals gathered along different sessions follows the same distribution, or distribution as similar as possible. There's also a difficulty is that this reconstruction must preserve the complex spatio-temporal-frequency structure of the EEG data.

Chapter 3

State of the Art

Recently, some scientist use deep learning network on the EEG data to do the emotion recognition, also some other domain adaptation methods have been proposed over the recent years, I will list the abstract of this papers in this chapter to give you the state of the art for this subject.

3.1 EEG-based emotion classification using deep belief networks

Abstract:[8] In recent years, there are many great successes in using deep architectures for unsupervised feature learning from data, especially for images and speech. In this paper, we introduce recent advanced deep learning models to classify two emotional categories (positive and negative) from EEG data. We train a deep belief network (DBN) with differential entropy features extracted from multichannel EEG as input. A hidden markov model (HMM) is integrated to accurately capture a more reliable emotional stage switching. We also compare the performance of the deep models to KNN, SVM and Graph regularized Extreme Learning Machine (GELM). The average accuracies of DBN-HMM, DBN, GELM, SVM, and KNN in our experiments are 87.62%, 86.91%, 85.67%, 84.08%, and 69.66%, respectively. Our experimental results show that the DBN and DBN-HMM models improve the accuracy of EEG-based emotion classification in comparison with the state-of-the-art methods.

3.2 EEG-Based Emotion Recognition Using Deep Learning Network with Principal Component Based Covariate Shift Adaptation

Abstract:[9] Automatic emotion recognition is one of the most challenging tasks. To detect emotion from nonstationary EEG signals, a sophisticated learning algorithm that can represent high-level abstraction is required. This study proposes the utilization of a deep learning network (DLN) to discover unknown feature correlation between input signals that is crucial for the learning task.The DLN is implemented with a stacked autoencoder (SAE) using hierarchical feature learning approach. Input features of the network are power spectral densities of 32-channel EEG signals from 32 subjects. To alleviate overfitting problem, principal component analysis (PCA) is applied to extract the most important components of initial input features. Furthermore, covariate shift adaptation of the principal components is implemented to minimize the nonstationary effect of EEG signals. Experimental results show that the DLN is capable of classifying three different levels of valence and arousal with accuracy of 49.52component based covariate shift adaptation enhances the respective classification accuracy by 5.55provides better performance compared to SVM and naive Bayes classifiers.

3.3 Domain Adaptation via Transfer Component Analysis

Abstract:[10] Automatic emotion recognition is one of the most challenging tasks. To detect emotion from nonstationary EEG signals, a sophisticated learning algorithm that can represent high-level abstraction is required. This study proposes the utilization of a deep learning network (DLN) to discover unknown feature correlation between input signals that is crucial for the learning task.The DLN is implemented with a stacked autoencoder (SAE) using hierarchical feature learning approach. Input features of the network are power spectral densities of 32-channel EEG signals from 32 subjects. To alleviate overfitting problem, principal component analysis (PCA) is applied to extract the most important components of initial input features. Furthermore, covariate shift adaptation of the principal components is implemented to minimize the nonstationary effect of EEG signals. Experimental results show that the DLN is capable of classifying

three different levels of valence and arousal with accuracy of 49.52component based covariate shift adaptation enhances the respective classification accuracy by 5.55provides better performance compared to SVM and naive Bayes classifiers.

3.4 Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach

Abstract:[11] The exponential increase in the availability of online reviews and recommendations makes sentiment classification an interesting topic in academic and industrial research. Reviews can span so many different domains that it is difficult to gather annotated training data for all of them. Hence, this paper studies the problem of domain adaptation for sentiment classifiers, hereby a system is trained on labeled reviews from one source domain but is meant to be deployed on another. We propose a deep learning approach which learns to extract a meaningful representation for each review in an unsupervised fashion. Sentiment classifiers trained with this high-level feature representation clearly outperform state-of-the-art methods on a benchmark composed of reviews of 4 types of Amazon products. Furthermore, this method scales well and allowed us to successfully perform domain adaptation on a larger industrial-strength dataset of 22 domains.

Chapter 4

Program

The approach we proposed for this subject is using domain adaptation to eliminate the variability problem in the EEG data. In this chapter, firstly I will present the structure of EEG data we used for this internship. Then I will present the adaption of the DANN structure to our case. Thirdly is defining the criteria for our Deep Neural Network.

4.1 EEG data

The EEG data we use is provided by Dr. Fabrizio De Vico Fallani from ICM in paris(Institut du Cerveau et de la Moelle Epinière). The EEG dataset contains 18 files. They are divided into two sessions from 9 subject.

The EEG signals refer to recordings of one minute during a state of no-task with closed eyes, or say "resting state". The data is collected by a helmet on the head. The helmet contains 56 electrodes all over the head, as shown in the Figure 4.1. The sampling frequency is equal to 200 Hz, so each file contains a matrix of 56 x 12000.

The preprocessing for the data is normalization. I have linearly normalize the sensor values to [0,1] by:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

where x_{min} and x_{max} are the minimum and maximum sensor value on the session.

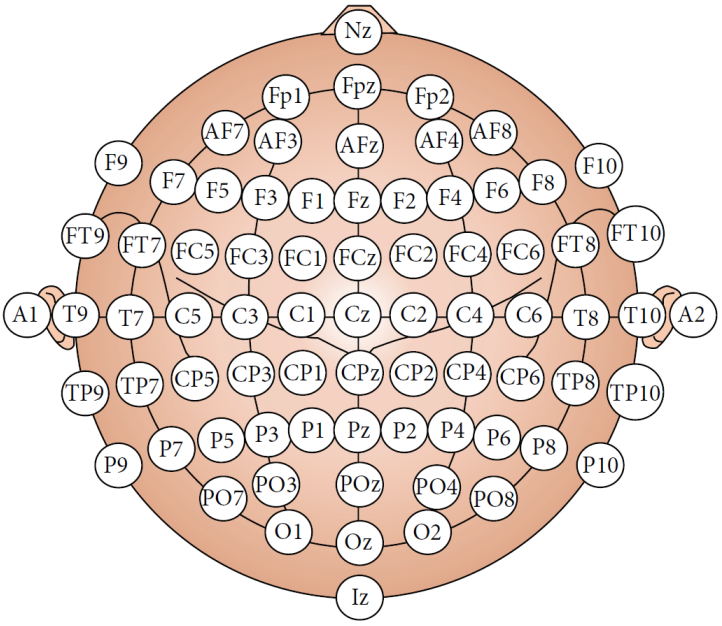

FIGURE 4.1: Location of the electrodes over the head of the subject

4.2 Architecture of Dann EEG

The principle for our adaption is that we treat session one of each subject EEG data as source domain in the Domain Adaptation, and the session two as the target domain. Once we using the DANN for training, the model will not be able to distinguish which session the input data belongs to, this is to say we can't tell the difference between session one and session two. So after reconstruction, the EEG data from two sessions become indistinguishable.

The base of our DANN is auto-encoder. An autoencoder, autoassociator or Diabolo network[12] is an artificial neural network used for unsupervised learning of efficient codings.[13] The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data.[14]

Considering that the input data dimension is 56, so in the input layer and output layer of our Neural Network, the size will be 56. In this internship, which is as a first approach to solve the problem, we use a simple structure with 1 hidden layer like 56 - n -56, where n varies from 1 to 56.

Above is the base structure of our neural network which is an auto-encoder. Then we need to add the gradient reversal layer which connected to the hidden layer. So the structure is shown in Figure 4.2

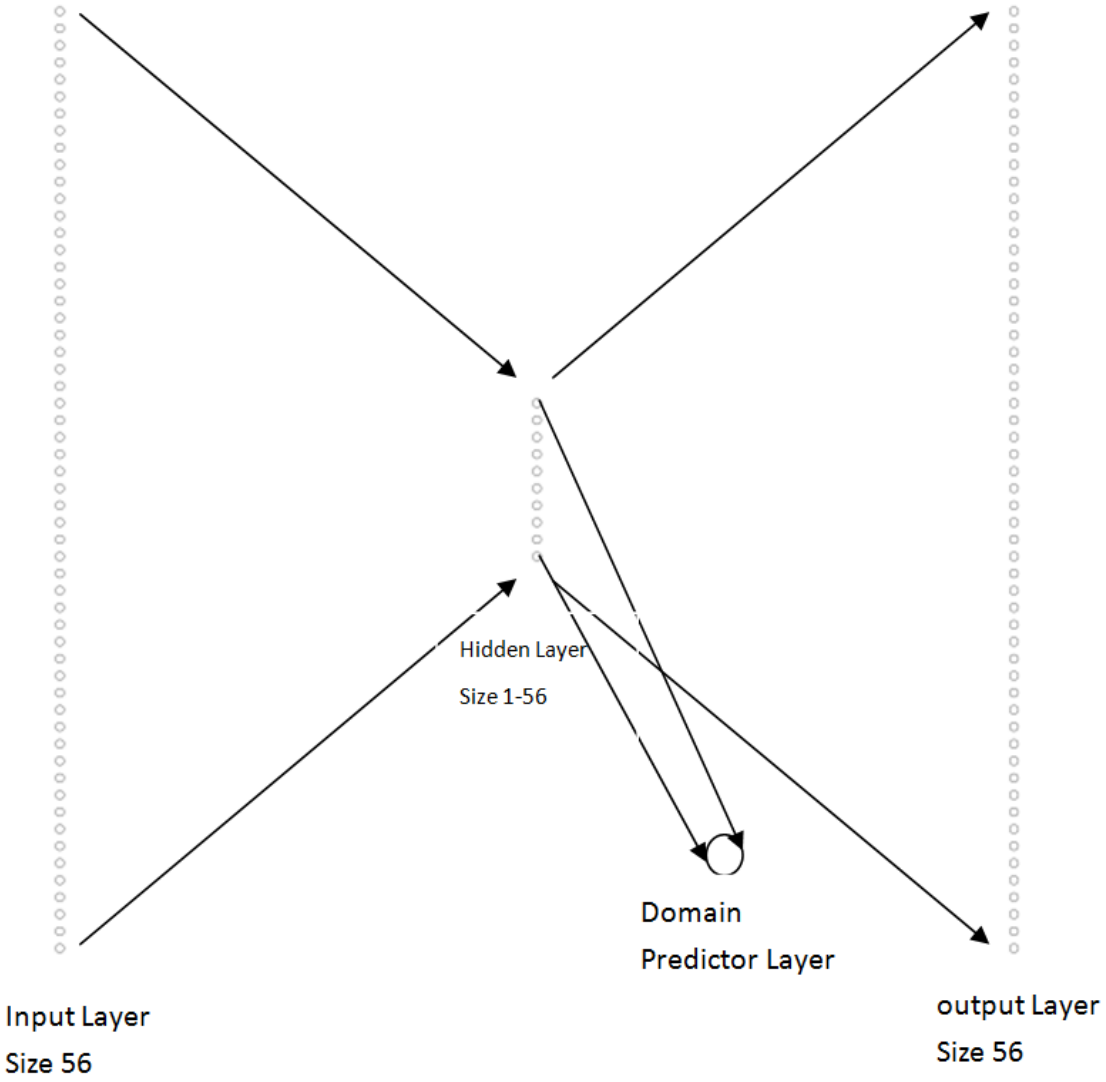

FIGURE 4.2: Structure of DANN for EEG data

At every neuron, the activation function will using **Sigmoid**.

4.3 Criteria

For our neural network, there are 2 purposes:

1. Minimize the reconstruction error of the EEG data.

2. Maximize the domain(session) prediction error, so that the sessions cannot be discriminated, even with the best classifier.

Considering the requirements above, two criterion functions will be defined in the program.

1. For the reconstruction loss of EEG, we want to be able to reconstruct the initial EEG data, so we choose the MSE cost function(mean square error) like:

$$loss(x, \tilde{x}) = \frac{1}{n} \times \sum_{i=1}^{56} |x_i - \tilde{x_i}|^2$$

2. The second criterion is session-invariant. We will assess this criterion by measuring the BCE(Binary Cross Entropy) criterion of the domain(session) predictor. So the loss function will be:

$$loss(o, t) = -\frac{1}{n} \sum (t[i] \times \log o[i] + (1 - t[i]) \times \log(1 - o[i])$$

where o is original prediction and t is target prediction. This criterion will give the entropy of the error, so when the result have a lot surprise, criterion will be large, otherwise the criterion will be 0.

4.4 Framework

To realize this neural network, I have using the **Torch** framework. Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

In the torch, there's already a **gradient reversal layer** prepared to use, so we can easy connect the auto-encoder with the gradient reversal layer to build our neural network.

In the program, I have defined three separate neural network, which are:

1. **Encoder**, the structure is 56 - n ($n \in [1, 56]$), this layer is used like feature extractor in the DANN, it also serves as a conpression part.

2. **Decoder**, the structure is n - 56 ($n \in [1, 56]$), this layer is used like label predictor in the DANN, it also serves to decode the data to get the initial data.

3. **Gradient reversal Layer**, the structure is n - 1 ($n \in [1, 56]$), this layer is used to predict which domain the instance is belonged to. With the help of negative gradient for this layer, we can maximize the error to make the two session indistinguishable.

Then in the training phase, the EEG data will be used as input data for Encoder, the output of Encoder will then be used as the input data of Decoder and Gradient reversal layer.

Chapter 5

TR-069 Client

TR-069 Client is implemented by Orange at 2008, but is a generic version for all potential devices. The first part of my internship is to make TR-069 Client for Homelive Box. To be able to provide a generic TR-069 module which is easily portable on different devices, it satisfy the following points:

- Written in ANSI C
- Small memory footprint
- Provide generic API to access device specific modules
- Provide Makefiles to build the binary
- Provide system traces on module activity

On a CPE, there are two modules that are mentioned all along this document and which are responsible of the CWMP:

- **TR-069 Agent**: This agent is responsible of the CWMP sessions. It initializes them with the inform message, realizes the ACS command(s) and execute some feedback command (notably after a firmware upgrade).
- **TR-069 Server**: This is a small HTTP server which listens of the WAN interface for an ACS solicitation (in TR-069, it is known as "connection request"). On a valid connection request, the TR-069 server contacts the TR-069 agent for starting a CWMP session with the ACS.

5.1 Architecture of TR-069 Client

The TR069 Generic Agent is composed of several modules and interfaces. Some modules are generic and can be ported with no modification. Other modules are platform specific (specific libraries usage, specific device API to get/set values, ...) and must implement services declared into generic interfaces.

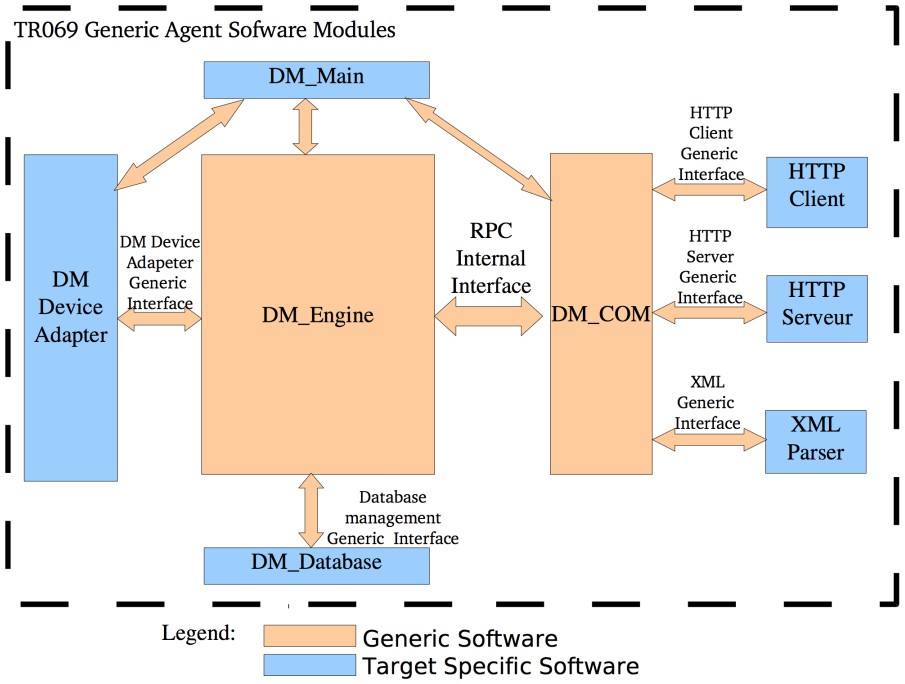

FIGURE 5.1: TR-069 Structure

For each module:

- **DM_ Engine** : In charge of the TR-069 logic,
- **DM_ Com** : Handles the HTTP, SOAP and SSL Protocol with the ACS (Auto Configuration Server),
- **DM_ DeviceAdapter** : Adaptation layer between the DM˙Engine and the device system. It allows the implementation of the RPC commands (Get Parameter Value, Set Parameter Value, Reboot, Download, ...),
- **DM_ Database** : Stores the data using an available storage solution on the device (or thanks to simple file storage),
- **HTTP Client** : Sets up / releases the HTTP connection with the ACS and sends / receives SOAP messages. The HTTP Client also handles SSL protocol,

- **HTTP SERVER** : Perform ACS Connection Request response,
- **XML Parser** : Decode / encode SOAP messages.

The figure below shows the file tree of the project, TR-069 Client defines all functions and APIs at each module. In consideration of easily ported to other devices, it put all target-related implementations in the folder *dm_target_implementation*.

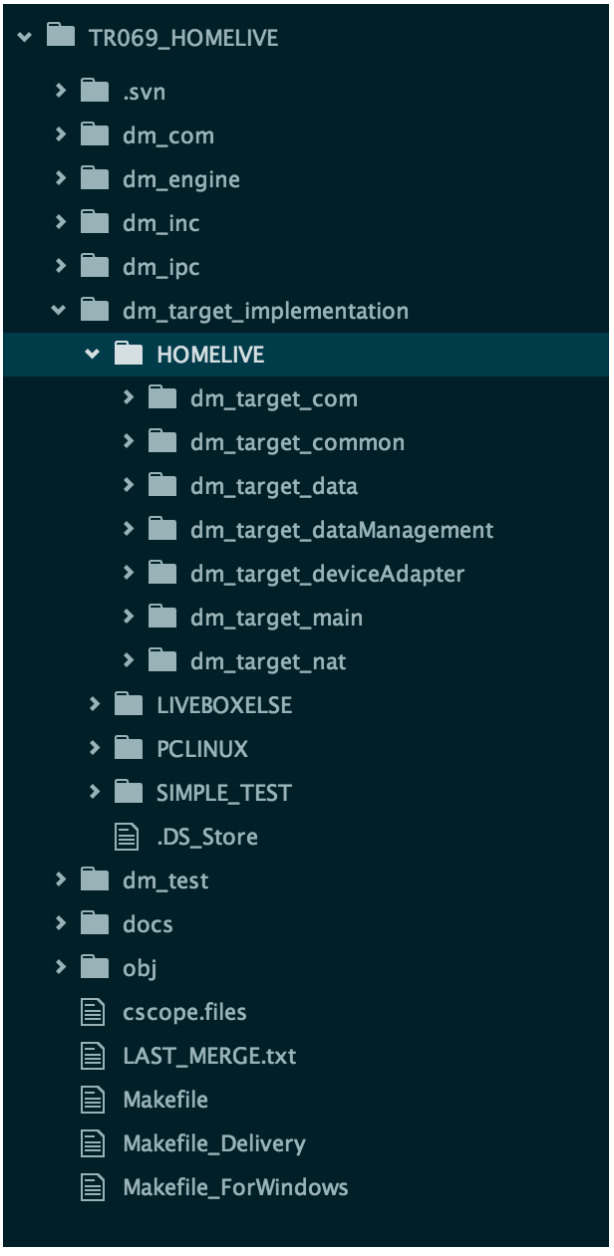

FIGURE 5.2: TR-069 File Tree Structure

In the folder *dm˙taget˙implementation*, different target has been defined, you can implement the APIs and Modules for different devices, and you specifie the target to compile in compilation, like:

```
$ make Target=TargetName TRACE_LEVEL=7 DEBUG=Y
$ make clean Target=TargetName
```

A csv file must be created for each new platform. This CSV file describes the data model parameters used by TR069 Agent and the default values for the TR069 specifics parameters. This target dependent CSV file is stored into *dm_target_implementation/MYNEWDEVICE/dm_dm_target_data/dm_csv* directory.

The CSV file is only read during the first system start up (or after a factory reset) and is used to generate the TR069 database (i.e. parameters.data and parameters.data˜ files)

5.2 Problems and Solutions

When ported the TR-069 Client to Homelive, we have encountered several problems before it can perfectly run on Homelive Box.

5.2.1 IP Address Error

The first problem is IP Address Error. At every start up of the TR-069 Client, the module will have to detect the IP address of the box, and use it to build connection between STUN server and CPE. But when executing, the TR-069 client can't detect the IP address, so I have located the code which are responsible for IP detection,

```
static const char* ETH_INTERFACE = "eth0";
struct ifaddrs *myaddrs = NULL, *ifa = NULL;
struct sockaddr_in *s4 = NULL;
int  status;
/* but must be big enough for an IPv6 address (e.g. 3ffe:2fa0:1010:ca22:020a:95ff:fe8a:1cf8) */
char buf[64];
memset((void *) buf,  0x00, sizeof(buf));

status = getifaddrs(&myaddrs);
if (status == 0)
{
   for (ifa = myaddrs; ifa != NULL; ifa = ifa->ifa_next)
   {
      if ( (ifa->ifa_addr != NULL)
         && ((ifa->ifa_flags & IFF_UP) != 0)
         && (ifa->ifa_addr->sa_family == AF_INET) )
```

```
        {
            s4 = (struct sockaddr_in *)(ifa->ifa_addr);
            if ( (inet_ntop(ifa->ifa_addr->sa_family, (void *)&(s4->sin_addr), buf, sizeof(buf)
                && (strcmp(ifa->ifa_name, ETH_INTERFACE)==0) )
            {
                ipAddress = strdup(buf);
                break;
            }
        }
    }
}
```

In general, the first Internet address should be *eth0*, but in Homelive Box, if you verify with command

```
$ ifconfig -a
```

The default Internet interface is "br-lan", which is specific in OpenWRT system, which is bridged Virtual Network Interface, used to make multiple virtual or physical network interfaces act as if they were just one network interface (quasi the opposite of VLANs).

So the solution is to change the macro definition of *ETH_INTERFACE* from *"eth0"* to *"br-lan"*. After the modification, the program can success finding the IP address.

5.2.2 STUN Initialization

When using the default data model provided, TR-069 Client can't build connection with STUN server. By looking into the log file, the problem aims to no STUN data model has been settled. The solution is to add the items which describe STUN parameters as follow:

```
ManagementServer.UDPConnectionRequestAddress;STRING;0;0;1;2;1;0;;0;0;0
ManagementServer.UDPConnectionRequestAddressNotificationLimit;UINT;0;1;0;0;0;0;;0;0;0
ManagementServer.STUNEnable;BOOLEAN;0;1;0;0;1;0;;1;0;0
ManagementServer.STUNServerAddress;STRING;0;1;0;0;1;0;;161.105.161.211;0;0
ManagementServer.STUNServerPort;INT;0;1;0;0;1;0;;3478;0;0
ManagementServer.STUNUsername;STRING;0;1;0;0;0;0;;test;0;0
ManagementServer.STUNPassword;STRING;0;1;0;0;0;0;;1234;0;0
ManagementServer.STUNMaximumKeepAlivePeriod;INT;0;1;0;0;0;0;;400;0;0
ManagementServer.STUNMinimumKeepAlivePeriod;UINT;0;1;0;0;0;0;;10;0;0
ManagementServer.NATDetected;BOOLEAN;0;0;1;2;1;0;;0;0;0
```

5.2.3 Memory Alignment Error

After the modifications above, every time when running the TR-069 Client, after 2 seconds, the program crash and report *bus error*, a bus error is a fault raised by hardware, notifying an operating system (OS) that a process is trying to access memory that the CPU cannot physically address: an invalid address for the address bus, hence the name. In modern use on most architectures these are much rarer than segmentation faults, which occur primarily due to memory access violations: problems in the logical address or permissions.

Because the TR-069 is generated by cross-compiling, some platforms (in our case MIPS64) can only read or write ints from addresses that are an even multiple of 8 bytes, otherwise they segfault. Even the ones that can handle arbitrary alignments are slower dealing with unaligned data (they have to fetch twice to get both halves), so the compiler will often pad structures to align variables. Treating structures as a lump of data that can be sent to disk or across the network thus requires extra work to ensure a consistent representation.

In the source code, there are *struct* defined like:

```
typedef struct _DM_ENG_Parameter
{
    bool writable; //1 byte
    char* name;  // 4 bytes
  int minValue; // 4 bytes

  ....

  struct _DM_ENG_Parameter* next;
} __attribute((packed)) DM_ENG_Parameter;
```

In a 32bits processor, the first three elements of the structure will be aligned like Figure 5.3. But when using the attribute (packed), the memory will become Figure 5.4 to save the memory space.

Under the Homelive architecture, it can only read data from address that is multiple of 4, so when the program tries to read value of name, the address is 0x0001 which is not an valid address, it reports the *bus error*. To solve this bus error, the easiest way is to delete the attribute *packed*, and then the compilator will automatic shift the address to meet the requirements.

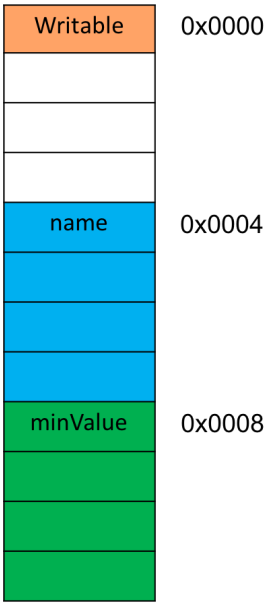

FIGURE 5.3: Memory Algnement under 32bits system

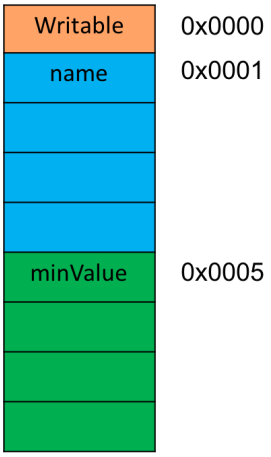

FIGURE 5.4: Memory Algnement under 32bits system with packed

5.3 RPC methods

The next step is to verify and implement all the RPC methods, after verification:

RPC Method	Result
Connection Request	Yes
Get RPC Methods	Yes
Set Parameter Values	Yes
Get Parameter Values	Yes
Set Parameter Attributes	Yes
Get Parameter Attributes	Yes
Get Parameter Names	Yes
Add Object	Yes
Delete Object	Yes
Reboot	No
Downloaded	Yes
Factory Reset	Yes
Schedule Inform	Yes
Upload	Yes

When we call *reboot* from ACS, the TR-069 turns out only reboot the program but not the whole box. To repair this, after located the reboot function, we added one line at the end of the function to let the program call the box to reboot after 5 seconds (which is for the TR-069 to shutdown all the process).

```
/*Reboot system after 5 seconds*/
system("(sleep 5 && reboot) &");
```

Chapter 6

Z-Wave

Homelive use a new Smart Home communication technology – Z-Wave. TR-069 Client should be able to retrieve the data in Luup and synchronize with TR-069 data model and ACS.

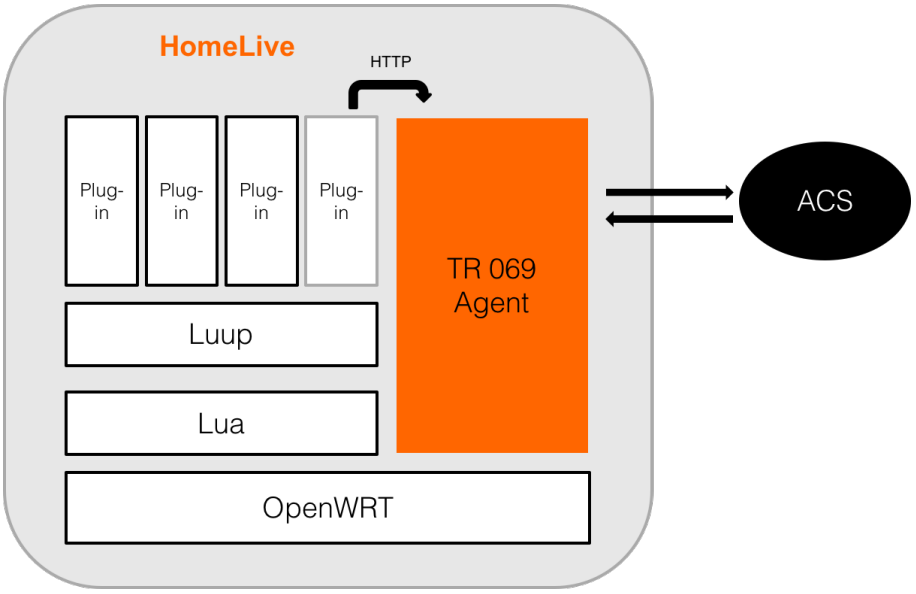

FIGURE 6.1: Homelive Structure

6.1 Protocol Z-Wave

Z-Wave is a standardized protocol for automation wireless habitat solution. It was designed by the Danish company *Zen-Sys* which was later bought by the US company

Sigma Designs in 2008. *Sigma Designs* and the Japanese company Mitsumi provides the Z-Wave chips.

Z-Wave equipment manufacturers are gathered in the Z-Wave Alliance. Founded in 2005, it promotes the protocol and ensures interoperability between devices. Interoperability is true on two levels: Radio layer and application layer. Certified equipment receive the Z-Wave logo (see Figure 6.2). More than three hundred companies have since joined this alliance.

FIGURE 6.2: Z-Wave Protocol Logo

The Z-Wave radio protocol is optimized to communicate with low bandwidth (between 9 and 40 kbps) and applied on stand-alone power supply or mains-powered, as opposed to Wi-Fi, which is intended for exchanges broadband.

Requirements\Protocol	**Z-Wave**	**Zigbee**	**En-Ocean**
Reliability	Yes	Yes	No
Security	Yes	Yes	No
Radio waves Reduction	Yes	Yes	Yes
Simplicity of use	Yes	-	No
Better pricing	Not yet	Not yet	No
Capitalization	Yes	-	Yes
Interoperability	Yes	No	Yes

6.2 Data Model Z-Wave

Orange has defined its own Z-Wave data model, which is divided into three parts:

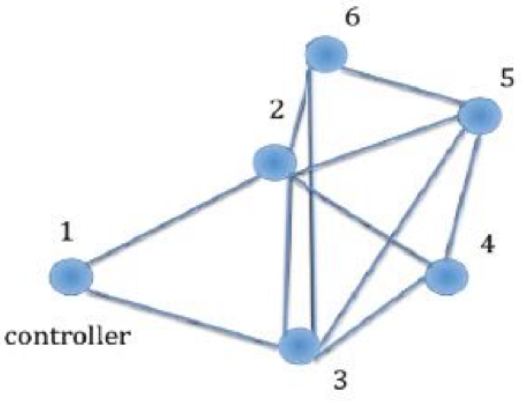

FIGURE 6.3: A Z-Wave Mesh Network Example

- The part *Z-Wave Network* defines the network topology
- The part *Z-Wave Structure* defines the set of information that can be recovered on the Z-Wave network to identify the devices.
- The part *Z-Wave Device* contains architecture and device configuration.

6.3 Update Z-Wave in TR-069

In order to retrieve the Z-Wave data from the Homelive Box, a http request to Homelive firmware can return a file (JSON or XML) which contains all the parameters that generated from lower layer. Then we should write a program to parser the file and save it in the TR-069 Client database.

6.3.1 HTTP Request

In addition to sending requests using standard UPnP, we can also do most things using a simple HTTP requests. Use the built-in URL data˙request, and pass the following on the URL: `http://ip_address:3480/data_request?id=user_data&output_format=xml`.

This returns the configuration data for Vera, which is a list of all devices and the UPnP variables which are persisted between resets as well as rooms, names, and other data the user sets as part of the configuration.

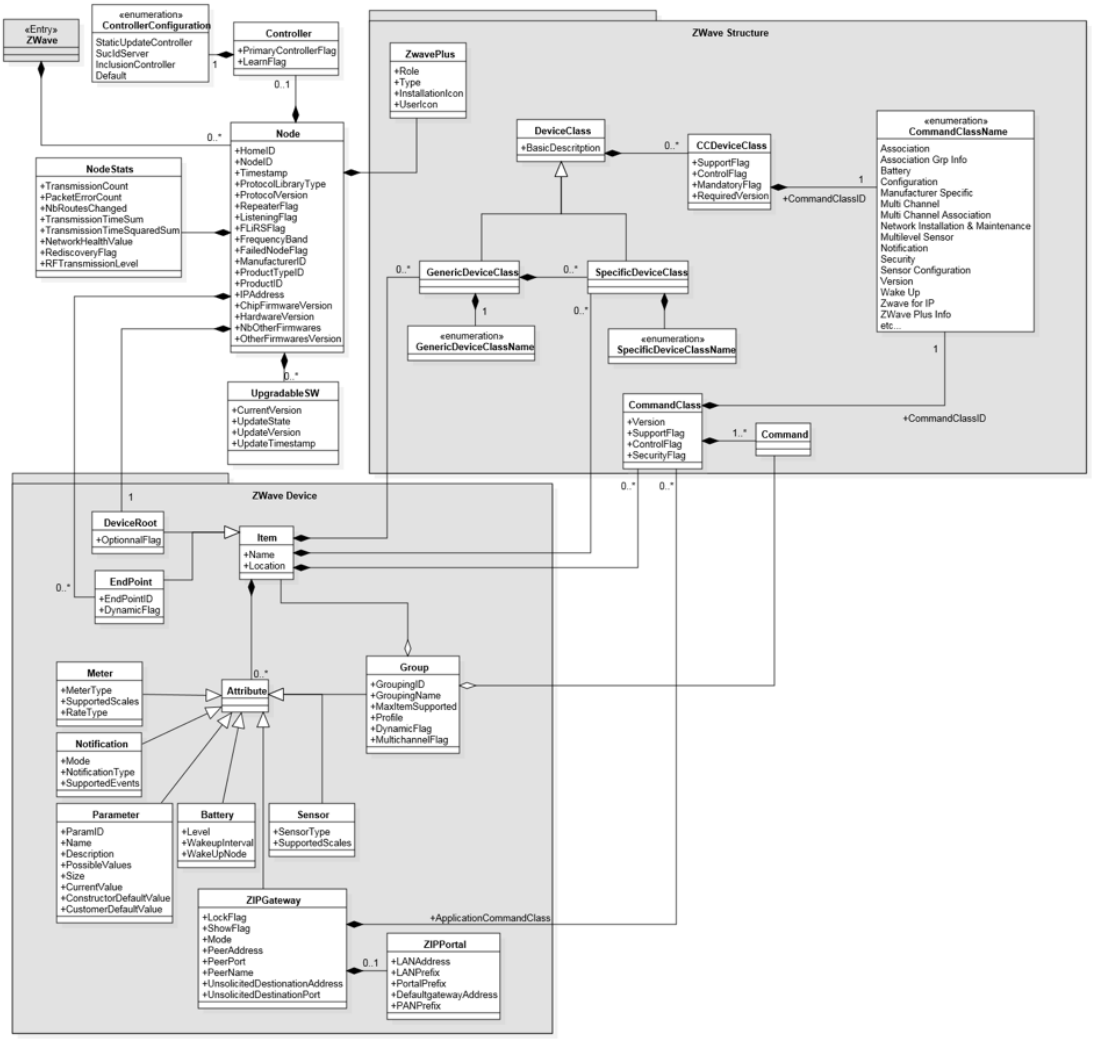

FIGURE 6.4: Z-Wave Node Data Model

In our case, we should retrieve the data of *status*, so we change *user˙data* to *status*. The http request is being called in the Homelive, so we can replace it with the local address: `http://127.0.0.1/`. And in consideration of parser, we set the output format to *JSON*. The request url become:

`http://127.0.0.1:3480/data_request?id=status&output_format=json`

We can call this http request by using the API already provided in TR-069 Client:

```
int DM_HttpGetFile(IN httpGetFileDataType * httpGetFileDataPtr)
```

After this, we will have the JSON file in format of Luup data model, the next step is to parse the file.

6.3.2 JSON Parser

JSON parser is not a default library of language C, so we have to find a JSON parser. Fortunately, in the Luup firmware, there is a JSON parser library already included, which is *json-c*. My program is based on *json-c* library and use a iterative structure to parse the JSON file we retrieved from Luup(See Appendices 2).

After parse the JSON file, we will have a huge name\value pairs, a two-dimension table is declared to saved them in buffer with dynamic memory management.

The next step is to find the correspondence between Z-Wave data model and Luup data model.

6.3.3 Simple file Storage System

TR-069 Client use simple file storage system as its data base. The structure is as following:

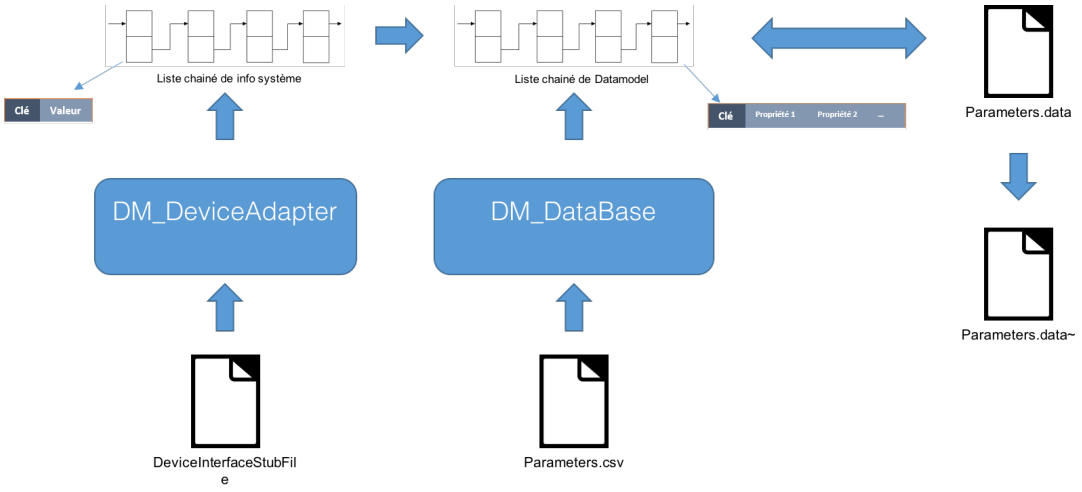

FIGURE 6.5: TR-069 Simple File Storage System

When the program starts, there are two modules are in charge of manage the database. First is the module *DM_ DeviceAdapter*, it reads the configuration file *DeviceInterfaceStubFile*, which contains the system parameters(Manufacturer, ProductClass, SerialNumber, etc..), and it will create a linked list to store the name\value pairs.

After that, module *DM_ Database* will read the data model file *parameters.csv* and build a linked list from it. In the *parameters.csv*, there are name\value of the data model and

also 10 properties (Datatype, Writable, Notification type, etc..). When finish building the long linked list, it will read the value from the short linked list(*DeviceInterfaceStubFile*) and copy them into the long linked list.

In the runtime, a file called *parameters.data* will be created, and TR-069 will store all the data model in this runtime file in format of CSV (Comma-separated values) as the storage base.

The backup file is called *parameters.data˜*, before every modification made to data model, the *parameters.data* will save all the information into *parameters.data˜* as a backup and update its own value.

6.3.4 Synchronization Data model Luup and Z-Wave

The data model of Luup and Z-Wave have different in approach but equally contains the data we need. For each data in Z-wave data model, we should find the correspondence in Luup data model. For example, in Luup there is a data called *ZWave.devices.*{*i*}*.states.15.variable WakeupInterval*, the correspondence we found in Z-Wave is *Device.Zwave.Interface.*{*i*}*.WakeUpInterval*. To synchronize the two data model, the value of *WakeInterval* in Luup should be given to Z-Wave *WakeInterval*. We have studied 100 parameters in Z-wave, and there are 27 are directly accessible from Luup data model, 26 can be calculated from Luup data model. For instance, the translation is finished for those all.

Appendix A

Installation of OpenWRT and TR-069

This appendix will introduce how to getting and building OpenWRT toolchain, compile TR-069 Client and run the client on the Homelive Box.

A.1 Getting and building OpenWRT toolchain

A.1.1 Get OpenWRT Source

First to do is create your OpenWRT folder and get the source code of OpenWRT using git or SVN:

```
$ tsocks svn co -r44360 svn://svn.openwrt.org/openwrt/trunk/ OpenWRT
```

or

```
$ tsocks git clone git://git.openwrt.org/openwrt.git
```

We choose the specific version 44360 because OpenWRT added unsupported libraries after this.

A.1.2 OpenWRT Buildroot Configuration

Then is to run configure interface to personalize your OpenWRT image:

```
$ make menuconfig
```

It will show a interface like below:

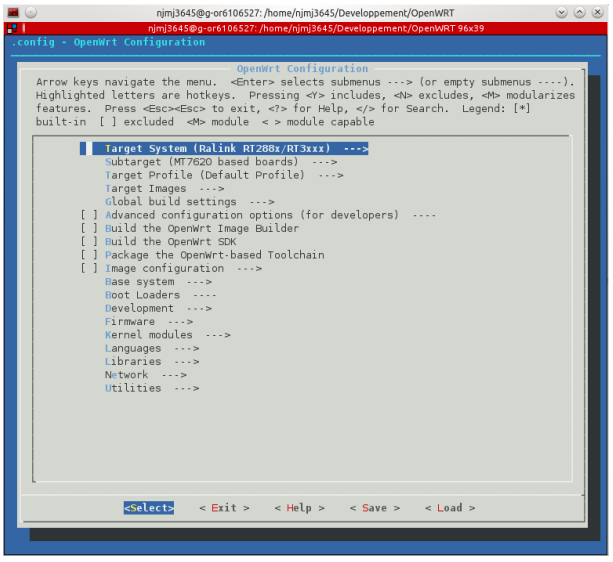

FIGURE A.1: OpenWRT Menu Config Interface

- In the **Target System** section, select **Ralink RT288x/RT3xxx**,
- In the **Subtarget** section, select **MT7620 base boards**.
- In the **Libraries** section, select **libcurl**.
- In the **Libraries/SSL** section, select **libopenssl**.
- In the **Base system** section, select **lipthread**.

Then save the new configuration and exit menuconfig.

A.1.3 Build the OpenWRT Toolchain

```
$ tsocks make
```

If there are errors during the compilation. After solving the problem, you can continue the compilation process by using commands like:

```
$ tsocks make package/install
```

or

```
$ tsocks make package/libs/curl/install
```

The compiling process may take a while. When finished, your OpenWRT toolchain is ready to use.

A.2 Building TR-069 Client

A.2.1 Get the source code

Getting the source code by using the SVN of Orange:

```
$ tsocks svn co https://www.forge,orange-labs.fr/svnroot/tr069agent
```

Before using OpenWRT toolchain, there are some path to set:

```
$ export STAGING_DIR=$HOME/path/to/openwrt/staging_dir
$ export PATH=$PATH:$STAGING_DIR/toolchain-mipsel_24krec+dsp_gcc-4.8-linaro_uCli
```

Then, the compiling command is:

```
$ make Target=HomeLive CC=$STAGING_DIR/toolchain-mipsel_24kec+dsp_gcc-4.8-linaro
```

You can find exactable file *cwmpd* in ./obj.

A.3 Connection to the Homelive Box

A.3.1 Port status checking

First change your network to Livebox LAN. Then use nmap to find the Homelive address:

```
$ sudo nmap -sP 192.168.1.0/24
```

Then check if the Homelive HTTP and SSH port are open:

```
$ sudo nmap -Pn 192.168.1.10 //Homelive address
Starting Nmap 6.00 ( http://nmap.org ) at 2015-08-20 10:43 CET
Nmap scan report for pc5.home (192.168.1.10)
Host is up (0.0019s latency).
Not shown: 996 closed ports
PORT    STATE    SERVICE
22/tcp  filtered ssh
53/tcp  open     domain
80/tcp  filtered http
443/tcp filtered https

Nmap done: 1 IP address (1 host up) scanned in 40.73 seconds
```

To open the ssh and http port permanently, you can put a script in /root and add the following line to contab:

```
*/2 * * * * /root/permissions.sh >/dev/null 2>&1
```

This line will execute the script every 2 minutes.

A.3.2 Connect to Homelive

We will use SSH to connect Homelive box:

```
$ sudo ssh root@192.168.1.10:/root
```

A.4 Running cwmpd on Homelive

Copy the *cwmpd* bin file and *parameters.csv*, *DeviceInterfaceStubFile* using scp.

A.4.1 Library Dependencies

List of the needed libraries for cwmpd:

```
$ objdump -p cwmpd | grep NEEDED
NEEDED               libcurl.so.4
NEEDED               libpthread.so.0
NEEDED               libpolarssl.so.7
NEEDED               libgcc_s.so.1
NEEDED               libc.so.0
```

We need to get the correct version of libpolarssl: libpolarssl.so.3. A temporary hack is to define a symbolic link from libpolarssl.so.7 to libpolarssl.so.3. On the HomeLive:

```
$ cd /usr/lib
$ ln -s libpolarssl.so.3 libpolarssl.so.7
$ ls -l libpolarssl.so*
lrwxrwxrwx    1 root     root            16 Dec  3 10:49 libpolarssl.so -> libpo
-rwxr-xr-x    1 root     root        223675 Feb 21  2014 libpolarssl.so.1.2.9
lrwxrwxrwx    1 root     root            20 Dec  3 10:49 libpolarssl.so.3 -> lib
lrwxrwxrwx    1 root     root            16 Jan 13 16:55 libpolarssl.so.7 -> lib
```

You can run cwmpd using:

```
$ ./cwmpd -p path/to/parametercsvfile/folder
```

Appendix B

JSON parser in C

This appendix will explain how to implement the JSON parser in C language using library JSON-c.

B.1 Objective of JSON Parser

The objective of our JSON parser is to parser a JSON file generated by Luup HTTP Request. The website Online JSON Viewer can be used to visualize the JSON file. Below is a JSON example of our JSON file named **status.json**:

In order to import the data into TR-069, the JSON format should be convert to Z-Wave format which is like:

```
ZWave.devices.1.id 1
ZWave.devices.1.states.1.id 208
ZWave.devices.1.states.1.service urn:micasaverde-com:serviceId:ZWaveNetwork1
ZWave.devices.1.states.1.value 1
ZWave.devices.1.states.2.id 209
ZWave.devices.1.states.2.service urn:micasaverde-com:serviceId:ZWaveNetwork1
ZWave.devices.1.states.2.variable UseMR
ZWave.devices.1.states.2.value 1
ZWave.devices.1.states.3.id 210
ZWave.devices.1.states.3.service urn:micasaverde-com:serviceId:ZWaveNetwork1
ZWave.devices.1.states.3.variable LimitNeighbors
ZWave.devices.1.states.3.value 0
ZWave.devices.1.states.4.id 211
ZWave.devices.1.states.4.service urn:micasaverde-com:serviceId:ZWaveNetwork1
ZWave.devices.1.states.4.variable LastDongleBackup
```

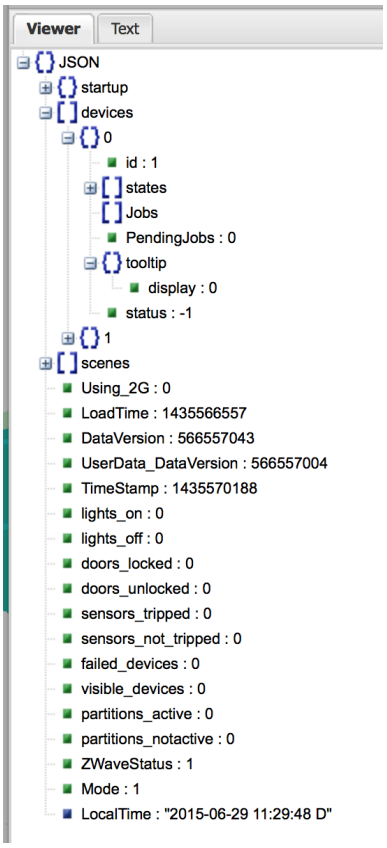

FIGURE B.1: JSON File in JSON Viewer Online

First part is the name and second is the value.

B.2 JSON Format Characteristic

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

An object is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by: (colon) and the name/value pairs are separated by, (comma).

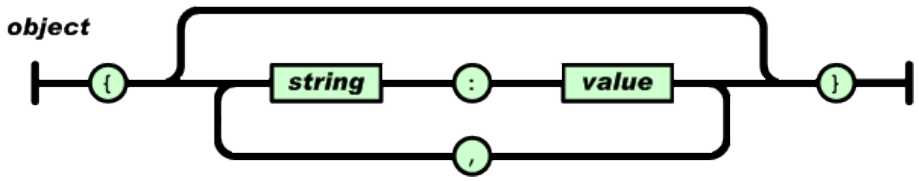

FIGURE B.2: JSON Object Structure

An array is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by, (comma).

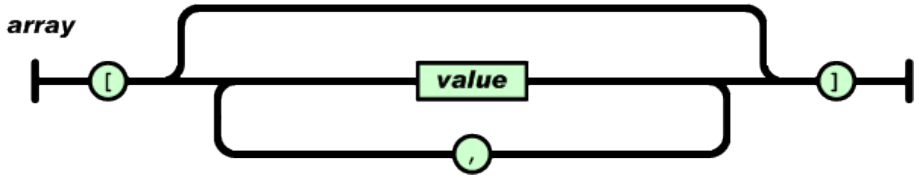

FIGURE B.3: JSON Array Structure

In the array case, the index must be added after the last level (e.g. ZWave.devices.1.states.2.id 209). The parser works in an iterative way.

B.3 Implementation of JSON Parser

At first, we set a buffer to store the path for each parameter. Then use the json-c library function *json_object_from_file* to save all the file content into a JSON object:

```
char path_json[70] = "ZWave.";    //set the buffer to save the name of parameter, also as th

struct json_object* jobj = json_object_from_file(fichierLu);  //fichierLu set to the file pa

_jsonParser(path_json, jobj);  //jobj is the json object will be parsed
```

_jsonParser is the private function to parse the JSON object. There is a essential json-c library function called *json_object_object_foreach(jobj, key, val)*, it allows to traverse each object in the JSON object. Before that, the current path should be added into a string.

```
/*Parsing the json object*/
void _jsonParser(char *path, json_object * jobj)
{
   enum json_type type;

   char parse_path[100];                          //define the local variable for path
   strcpy(parse_path,path);
```

```
    json_object_object_foreach(jobj, key, val)   /*Passing through every array element*/
    {
       type = json_object_get_type(val);
       switch (type)
       {
          case json_type_boolean:
          case json_type_double:
          case json_type_int:
          case json_type_string:  _printJsonValue(parse_path, val, key);
                                  break;

          case json_type_object:  sprintf(&parse_path[strlen(parse_path)], "%s.",key );
                                  jobj = json_object_object_get(jobj, key);
                                  _jsonParser(parse_path, jobj);
                                  break;

          case json_type_array:   _jsonParseArray(jobj, key, parse_path);
                                  break;
       }
    }
}
```

If the object parsed is a value, jump to the *_printJsonValue* function to save the value in buffer table. If it's a object, redo this function. If it's an array, go to the *_jsonParseArray* function.

```
/*Parsing the json array*/
void _jsonParseArray( json_object *jobj, char *key, char *path)
{
   void _jsonParser(char *path, json_object * jobj); /*Forward Declaration*/
   enum json_type type;

   char array_path[100];                          //define the local variable for path
   strcpy(array_path,path);
   json_object * jvalue;


   json_object *jarray = jobj; /*Simply get the array*/

   if(key) {jarray = json_object_object_get(jobj, key); /*Getting the array if it is a key value pa

   int arraylen = json_object_array_length(jarray); /*Getting the length of the array*/
   int i;

   sprintf(&array_path[strlen(array_path)],"%s.",key);
```

```
    sprintf(&array_path[strlen(array_path)],"x.");   //fill the path with x. for instance

    for (i=0; i< arraylen; i++)
    {
       jvalue = json_object_array_get_idx(jarray, i); /*Getting the array element at position
       type = json_object_get_type(jvalue);

       if( i <= 9 )
          sprintf(&array_path[strlen(array_path)-2], "%d.",i + 1 );  //replace the 2 last cha
       else
          sprintf(&array_path[strlen(array_path)-3], "%d.",i + 1 );  //replace the 3 last cha


       if (type == json_type_array)
          _jsonParseArray(jvalue, NULL,array_path);
       else if (type == json_type_object)
          _jsonParser(array_path, jvalue);
       else
          _printJsonValue(array_path, jvalue, NULL);
    }
}
```

The main difficulty of parse a array is to add the index. The solution is to add .x at every beginning of Array parse, and then replace the .x according to different object type.

```
/*At the end of each iteration, write the name/value pair to a 2D table*/
void _printJsonValue( char* path, json_object *jobj, char *key)
{
  enum json_type type;

  type = json_object_get_type(jobj); /*Getting the type of the json object*/

  char* value_buffer;
  value_buffer = (char*)malloc(150 * sizeof(char));

  char* paramKey;
  paramKey = (char*)malloc((strlen(path)+strlen(key)) * sizeof(char));

  strcpy(paramKey,path);         //build the JSON object name string
  strcat(paramKey,key);

  switch (type)
  {
      case json_type_boolean: sprintf(value_buffer,"%s",json_object_get_boolean(jobj)? "true
                              sdataParameterList[indiceValueStruct] = DM_ENG_newSystemParame
                              break;
```

```
        case json_type_double:  sprintf(value_buffer,"%lf",json_object_get_double(jobj));
                                sdataParameterList[indiceValueStruct] = DM_ENG_newSystemParameterValu
                                break;
        case json_type_int:     sprintf(value_buffer,"%d",json_object_get_int(jobj));
                                sdataParameterList[indiceValueStruct] = DM_ENG_newSystemParameterValu
                                break;
        case json_type_string:  value_buffer = strdup(json_object_get_string(jobj));
                                sdataParameterList[indiceValueStruct] = DM_ENG_newSystemParameterValu
                                break;
    }
    free(value_buffer);
    indiceValueStruct++;
  }
```

On the *_printJsonValue* function, each element in the buffe table is a TR-069 *System Parameter Value Struct* using the TR-069 Client API default function *DM_ENG_newSystemParameterValueStruct*.

Bibliography

[1] Bruno A Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[2] John S Bridle and Stephen Cox. Recnorm: Simultaneous normalisation and classification applied to speech recognition. In *NIPS*, pages 234–240, 1990.

[3] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

[4] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from multiple sources. *Journal of Machine Learning Research*, 9(Aug):1757–1774, 2008.

[5] Jonathan R Wolpaw, Niels Birbaumer, William J Heetderks, Dennis J McFarland, P Hunter Peckham, Gerwin Schalk, Emanuel Donchin, Louis A Quatrano, Charles J Robinson, Theresa M Vaughan, et al. Brain-computer interface technology: a review of the first international meeting. *IEEE transactions on rehabilitation engineering*, 8(2):164–173, 2000.

[6] Ingrid Wickelgren. Brain researchers try to salvage estrogen treatments. *Science*, 302(5648):1138, 2003.

[7] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[8] Wei-Long Zheng, Jia-Yi Zhu, Yong Peng, and Bao-Liang Lu. Eeg-based emotion classification using deep belief networks. In *2014 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2014.

[9] Suwicha Jirayucharoensak, Setha Pan-Ngum, and Pasin Israsena. Eeg-based emotion recognition using deep learning network with principal component based covariate shift adaptation. *The Scientific World Journal*, 2014, 2014.

[10] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22 (2):199–210, 2011.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

[12] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[13] Cheng-Yuan Liou, Jau-Chi Huang, and Wen-Chie Yang. Modeling word perception using the elman network. *Neurocomputing*, 71(16):3150–3157, 2008.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.