Mobile icon-based phrasebook for travelers

Bachelor Thesis of:
Evghenia Derivolcova
Date: August 2008

University of Rousse **Delft University of Technology**

Acknowledgements

First of all I would like to thank my project leader professor Dr. drs. L.J.M. Rothkrantz of the KBS chair for the support, attention and great help that he provided for me during my stay in the Netherlands. I want to thank him for being understanding, good-heart, jocular and kind.
The thanks also belongs to chief assistant S. Smrikarova from the University of Rousse, who helped me to come in Delft for development my bachelors' diploma projects in Delft University of Technology and supported me during all the period of my stay here. I would also like to thank all guys from Man Machine Interaction group of TU Delft whom we were working with together in the lab. They were funny and friendly with me and my colleague Borislav, they shared their knowledge about life in Holland and they spend time together. Finally, I would like to express my gratitude to TU Delft for providing this possibility for foreign students to study and to develop their project in the university.

Abstract

The most problematic aspect in human communication today is a problem of common language. People these days like to travel a lot. They travel to see new places, to get new experiences, to meet new people, to get away from the daily routine or even for a business trip. No matter the reason we travel, everyone who travelled at least once knows the problems that might occur during the trip and of course, the first problem that traveller meets is *non*-understanding. People need one common language to speak with each other. Such a language like Esperanto was created. The goal for creating Esperanto was to create an easy and flexible language that would serve as a universal second language to foster peace and international understanding. This is a language with limited requirements that makes it easy to learn. Esperanto speakers are more numerous in Europe and East Asia than in America, Africa, and Oceania, and more numerous in urban than in rural areas. [3] But unfortunately Esperanto is not usable and not so many people speak this language yet. It looks like English is most spoken language today all over the world. But the problem with language understanding still exists. You can't be sure that people will be able to understand you and you will be able to explain them your needs during the trip. Because of all these reasons, we have to think about new concept of representing information, new way to describe knowledge that can be used and understood by people from different cultures, different ages, and different social groups. It should be the human natural way of communication.

Developed by: Evghenia Derivolcova
Application name: Mobile Icon-Based phrasebook for travelers
Submit date: 28.08.2008
Receiving University: Delft University of Technology
Project supervisor: Professor Leon Rothkrantz
Sending University: University of Rousse
Erasmus coordinator: Stoyanka Smrikarova

Table of content

Table of figures

1. Introduction

The main goal of this project is to create an application that gives the possibility to travelers to communicate using an international "language" - the icons. The best way to share information is using the icons, which is a visual and natural way of communication. With the development of modern mobile devices (such as PDAs) new possibilities for communicating in other languages appeared, such as using language translation software or a digital dictionary. It makes PDAs very favorable for our goal to create application for travelers. I take in consideration typical situations that might occur to a traveler: looking for accommodation, using transport resources, shopping, using medical services, going outside, society communication and etc. and also the fact that the communication can take place without the user knowing an international language, and even the fact that he/she may not be able to speak. The last fact exceeds the scope of using applications like ours. Possible users of the system can be people with speech problems (deaf and dumb people) or for example people learning foreign languages using the system to practice pronunciation (additional training for using the system will be needed). Furthermore the system can be used as a tool for creating notes or written instructions using different languages.

We have to create our own language, constructing it in a matter of using a new way of communicating, including the following levels: vocabulary, ontology, grammar rules, also taking care of the syntactical and semantic problems. The main idea is to permit the expression of ideas using only icons. My wish is to create the application using Java Technologies for PDA (Personal Digital Assistant) with a good looking user interface, with easy to understand icons, that can communicate with users in different languages (text and speech).

I am well aware of the fact that one can describe a situation in many ways, use different words, create sentences with different levels of difficulty and usually every spoken language is very flexible to allow it. That's why the human needs to have some restrictions, the application should offer simple basic sentences, created in advance, that will clearly represent the main idea and won't allow the user to build meaningless sentences.

The input data for the system is made up of basic, structured sentences and icons that should represent them.

For storing these resources we need a database, but keeping in mind that the application which we want to create will be run on PDA it has to be as simple as possible. In a more difficult variant of our system, the resources needed for PDA can be stored on the remote server and be accessed by transferring via internet communication (or another practicable kind of communication). Therefore this application can be a server – client side one.

In our prototype all the necessary data we will be stored locally and we will focus only on the client side.

For input information I decided to use an xml file that combines all the necessary data for creating sentences: basic sentence, reference to used icons (as icons are stored locally), ordered in a proper way to describe the used structure (graph).
The output data is a generated text (sentence or sequence of sentences) according to chosen language and speech reproducing in the same language.

1.1. Communication using icons

As I will use icons in my system to represent dictionary sentences, I would like to make some introduction to this topic and discuss the situation about using icons.

What we would like to achieve in this project is to get a structured World Model from real life and to realize it in graphical way. A World Model is composed of objects, characteristic features of the objects, and relations between the objects. Every observer has his own World Model, has his own view of the world, and it can arouse some difficulties with understanding ideas. An observer will look at the situation that he wants to describe to his interlocutor and he will form his own ideas of how to represent it. We have to offer the most convenient icons for the user to define his idea.

1.1.1. About icons. Usage of icons in communication

Icons offer a rich potential for communication across natural language barriers.

Just as any language, icon language is something that does need some training. Most people already have some training in recognizing icons however, because icons can be found anywhere.
In modern society everyone is familiar with icons, both in and out of work: for example, icons on the toilet door, iconic road signs and complex icons on electronic goods. From the everyday context of living to the packaging for the latest products, one can meet icons as a daily occurrence.
In the computer world, the use of icons has been an extension of their traditional uses but computer and related technologies offer the unique possibility of exploiting animation and interaction.
Furthermore, a computer interface language, which consists entirely of icons, would have many advantages. It would avoid the need of foreign language translation, it would assist those with language and learning difficulties, even the people who can’t speak, and it would help teaching new systems. Also, the use of pictures and gestures to convey our ideas is a basic form of communication that two people frequently resort to when they find they do not share a common language. Because you cannot speak does not always mean that you cannot write or maybe produce some kind of mark. This possibility is often ignored. It is a basic human need to make your own mark, and is of psychological importance not only to read or recognize something, but

also to have some fast personal means of having an input into your care or daily life. Ideally you need a way of expressing your thoughts and feelings, negative as well as positive.
All these ideas are based on a concept called communication. One should understand the meaning of the icon, remembering it if it is not the first time he/she saw it, and so on. [1a]

1.1.2. The history of icons

Icons are graphical symbols representing a concept or object in reality. The term icon has been adapted from the Russian word “ikona”, which is a religious painting or statue. Icons have been around for a very long time, e.g. in the middle ages complex iconic systems have been used to denote systems of astrological signs. It may even be argued that the ancient Egyptians were using icons as a language. They may not have called them icons, but they did communicate using graphics.
In the 1930s Otto Neurath developed the Isotype, a system for communication which uses stylized graphics within a two-dimensional syntax. Neurath’s work ranged from a very specific example of how a complex idea can be conveyed graphically, to a proposal for an international set of iconic images.
In the 1950s, Charles Bliss developed a set of atomic icons that represent basic objects in the world, and their features. These can be combined to form complex icons that map on to the set of words found in natural languages. *Figure 1* below shows how we can construct a symbol for telephone using: mouth-ear-language-electricity-telephone:

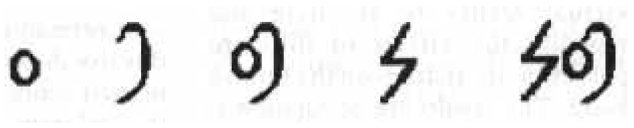


Figure 1. Symbols for telephone

The work of Bliss has some resemblance with the work of linguist Anna Wierzbicka, who claims to be able to describe any concept with using only 61 different words. The combination of these atomic words lead to a new concept, just as the atomic pictures of Bliss lead to a new concept. Although Wierzbicka does not use icons, the possibility of mapping her atomic words to atomic icons seems interesting.
The iconic languages were not all as successful as their developers might have hoped for, but they do show that there are distinct advantages in a communication based on graphical icons.

Our ability to learn or to recall the meaning of a sign seems to be greatly enhanced to the point where we may not need to be told what the sign represents or to explicitly learn its meaning. There might also be some advantage in the efficiency of using icons over natural languages in the sense that difficult concepts might be represented by only a small number of icons, as opposed to many words. Furthermore our ability to recognize icons does not depend on the natural languages we know, suggesting that iconic systems may be a way to overcome linguistic differences. Note that this does not imply that icons are also culturally independent. [4]

1.1.3. Icon design

An icon can be seen first by its perceivable form (syntax), second by the relation between its form and what it means (semantics), and third by its use (pragmatics). In general, the semantics of icons should be based upon their real-world domain.

An ideal icon language wouldn’t need any explanation, the intuition of the user, based upon his life experience, and should be enough to immediately understand it. Of course, this is not a very realistic goal.

The challenge in designing icons is that they should be as easy as possible to learn, as easy as possible to remember, and as easy as possible to recognize. [4]
Therefore icons should have “good” design:

- *Simple,* familiar and self-explanatory.

Although we have decided that icons should not have to be intuitive; their perceiver should immediately know its significance in the context in which it is used. Otherwise people will find it difficult to understand its meaning.

- *Semantically unambiguous*.

All the icons in the interface should be easily differentiated with each other. They should not be easily misinterpreted nor contain cultural, racial or linguistic bias. They should not be culturally dependent.

- *Consistent.*

Related icons with the same set of concepts should be grouped. The way abstract concepts are expressed within these should be clear and cohesive as well. The icon elements should be used consistently throughout the design. Therefore, the users would be able to easily recognize and learn standard interface elements and icons across the interface.[5]

- *Graphically clear.*

Icons should be recognizable at all levels:

- The *Symbolic* level.
  A notebook and pencil together are symbolic for a simple text editor, a notebook and a fountain pen for a more advanced text editor. A car, a bucket and a brush symbolize an application to wash a car.

- The *Object* level.
  Objects often have an archetypal form and deviations from that form. You can have a classical watch or a swatch.

Each one of us has another image of a classical watch in his mind. You may have experienced being in a supermarket, looking for a certain product, while having the wrong image of it in your mind. It is hard to find then, while with the right image in mind it is spotted instantly. For this reason the artist should not follow his own image and paint it as he sees it in his mind's eye. He should bring it down to the essential elements.

- The *Shapes* level.
  You travel by train to a big city you do not know yet. You pass the industrial outskirts, you see many non-descriptive buildings, and forget them immediately. On the other hand, if you ever saw an image of the Empire State Building, you will always recognize it. Some shapes stand out, others do not. You will not mistake a Mondrian for a Monet.

- The *Assembly* level.
  Shapes together form an assembled shape. This shape too can stand out or not. Icon sets may enhance the assembly shape with an outline.

- The *Sizes* level.
  If images are too small all details are lost, and you can’t recognize the main idea of picture. [1]

1.1.4. Usage of icons in software development

Most of today's applications use icons, even though most developers don't bother to implement custom designed icons into their applications. Actually, settling with the operating system's default stock icons is not such a bad thing as some people might think. Computer users might sometimes have some problems adapting to new applications, especially if they have different interfaces than the applications already installed on the users' computers. What happens if you want to save and you're looking for a disk icon, but you can't find it because the developer decided to use a star icon? If so, there will be a poor communication between the user and the application, because the interface's icons are different from the ones the user is used to. Application developers should not fall into this trap just because they like some other icons and they don't want to use the same old default system icons, because it would do more bad than good. The default system icons are preferred because most applications use them, so users will learn to use your application a lot faster. The functions and commands will be easier to understand, because users will be able to identify the iconic symbols faster.

To have a good communication between the user and the application, developers must predict all the possible problems that the interface could cause to its users.

There are multiple benefits from using icons in an application interface. No matter whether an application uses the operating system's default icons or has custom-made icons designed especially for it, there must be some form of graphics in it, otherwise people might find it harder to use. This may happen because most of today's applications use icons to ease the learning of the application. Because most applications use similar icons in certain tasks - for example a printer icon for printing or a disk button for saving files - it's a lot faster for users to click on those buttons, therefore they will remember it next time they use that application, or any other application for that matter. Even if later on advanced users will choose to use shortcut keys on the keyboard instead of clicking with the mouse on buttons in the toolbars, it's really important that for starters they will use the toolbar, so icons will be very important in the communication with the application [2].

1.2. Project Definition

Taking in account that in the scope of a single thesis work not everything can be handled and developed, I tried to make a research of usage software for mobile devices in traveling aspect. This particular thesis work is focused on the topic definition research, interface and logical realization, and is defined as follows:

> "Define key aspects of topic research, design a system that is suited for iconic communication for a travelling and implement system prototype with understandable user interface. The system should be intelligent enough to assemble and express a correct and up to date world model of a traveler."

1.3. Problem Definition

<u>Problem definition</u>: There are problems that I met during researching for this project:

- Technology that I used for developing the user interface part is completely new and it is still under development. So, the difficulties of new technology are: not enough documentation; tutorials are based mainly on examples; the libraries (API) are always changing; some problems with supporting technology in existing tools.
- Another difficulty is founding a proper set of icons that can describe all sentences, all ideas clearly. As I was looking for free icons on the internet, I could find different types of icons, but they didn't cover all ideas of our application, all basic sentences we have. We need a unique icon set for our project.

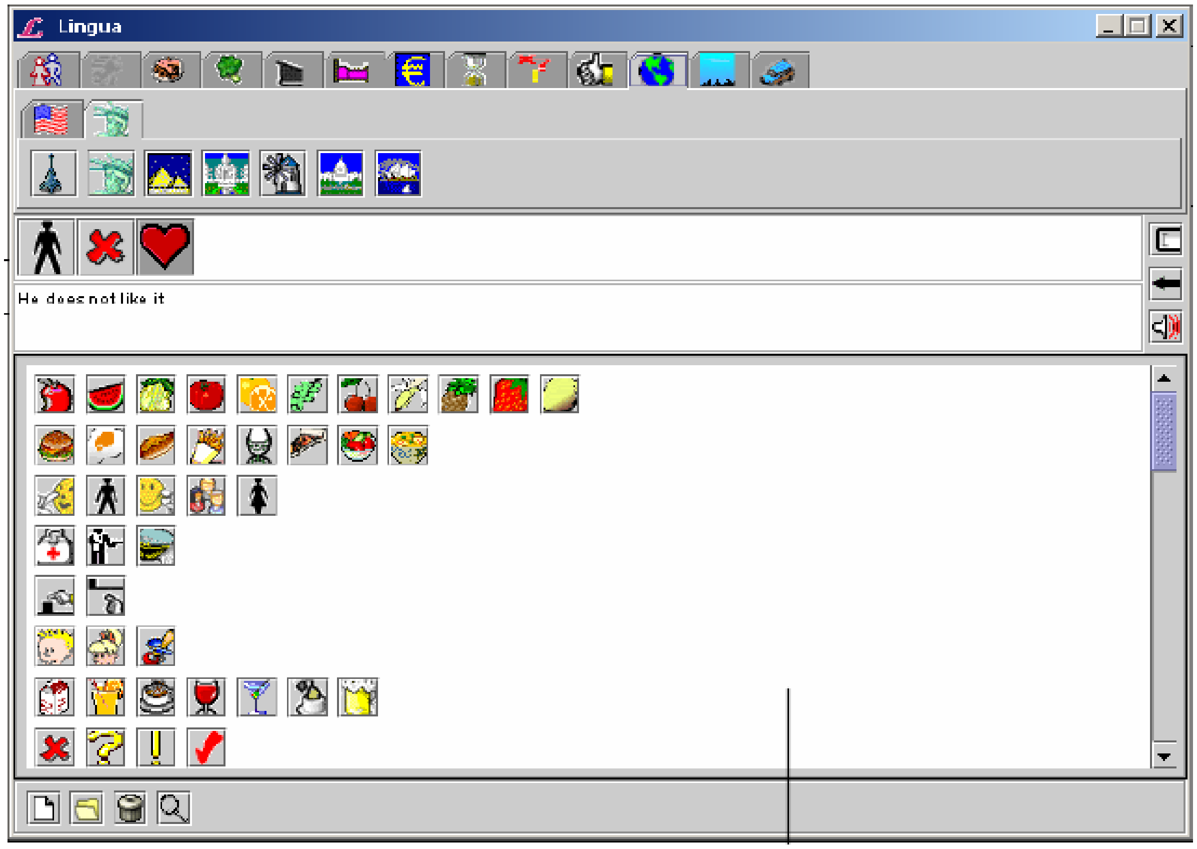
- Problem with specific world understanding. Everyone has his own perception of how to create sentence, how to represent it graphically, and some disagreements can happen during the system’s usage. That’s why the concept of dictionary has to be considered very carefully.

2. Related work

2.1. System Lingua

An icon-based communication interface on a PDA research and application "Lingua" was developed by Siska Fitrianie.

The Lingua system is designed for multilingual use. To be able to speak other languages, we had to separate the Lingua system from the text-to-speech synthesizer and the language translator. The Lingua application only interprets the selected sequence of icons into a meta language.



He does not like it

Figure 2. Lingua System's Interface

The Lingua application consists of four parts: Syntax Analyzer, Next Icon Predictor, Inference Engine and the interface.

- *Syntax Analyzer*
  The Syntax Analyzer receives a sequence of selected icons from the user interface. It parses the sequence from left to right. The icon database defines the terminal symbol of

each icon. The terminal symbol consists of the words or phrases that an icon represents. The analyzer uses this information to dire a rule that fits the sequence best. The parser takes every icon from the sequence and matches it with every terminal symbol according to the grammar rules. The algorithm stops if a sentence is formed and the matched grammar rule is fired. If a sentence is not yet formed and there is not any syntax error, then the system will list all next possible terminal symbols from all possible grammar rules. Application "Lingua" uses BNF grammar for creating correct syntax. All syntax rules are described in the xml.

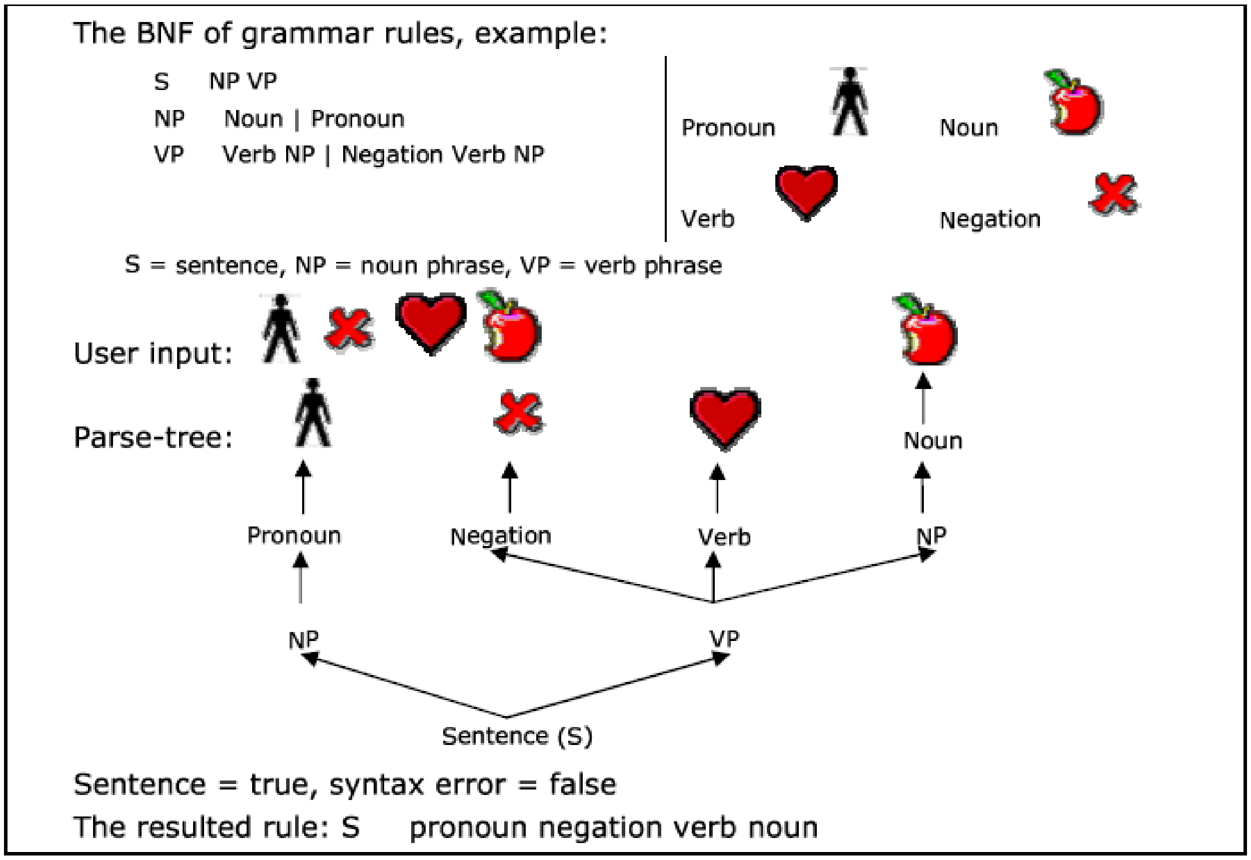

The BNF of grammar rules, example:
S NP VP
NP Noun | Pronoun
VP Verb NP | Negation Verb NP
Pronoun
Noun
Verb
Negation
S = sentence, NP = noun phrase, VP = verb phrase
User input:
Parse-tree:
Noun
Pronoun
Negation
Verb
NP
NP
VP
Sentence (S)
Sentence = true, syntax error = false
The resulted rule: S pronoun negation verb noun

Figure 3. Lingua BNF grammar

- *Next Icon predictor*
  The predictor receives the selected sequence of icons. It also receives the next possible terminal symbols from the Syntax Analyzer. The predictor calculates and ranks the probability of every possible next icon. The Lingua interface receives the ranked icons and displays them based on their concepts. The display is refreshed each time the user adds a new icon to the sequence.

- *Inference Engine*
  The inference engine receives selected sequence of icons and grammar rules. Based on input data and using algorithms this module creates complete sentences.

- *Interface*
  The main difficulties in this phase were designing a large amount of the icons. System has five hundreds icons in its vocabulary divided into thirteen concepts. [5]

Advantages: As a result of my observation and partition testing of the system, it provides high level grammar correctness of built sentences, has a big scope of thematic, give freedom for user to create sentences he wants.

Disadvantages: One of the confusing things is that the system allows users to create meaningless sentences, because only grammar rules are applied.
Because of this reason in our project I'll try to apply another concept in order to restrict users in choice.

3. Global Design

In this chapter I will try to explain the main idea of this project as clearly and as detailed as possible. I will describe the model overview and feasible ways to realize the purposes of the system, the steps of creating the application and more detailed explanation about the program design of the system.

3.1. Overview

My goal in this project is to develop and describe an application which should communicate with users in an understandable way using universal visual language – using icons. The user doesn't need to care about grammar rules or lexical difficulties etc. Everything he should do is to try to express his idea with the icons given to him.

One of the problems that might occur when you give freedom to user to create his own sentences using icons is the nonsense (for example: *"You and dog and spoon like oranges"*. The grammar rules are kept correctly but in general the sentence has no meaning). To avoid this problem I decided to restrict users with sentences he/she can create. Therefore the first specialty of this project is that the user can't construct various sentences he wants. Only the sentences built in the program model can be used. The user will only see those icons on the GUI model that respond to words and needed for creating basic sentences our application offers.

Another feature of the system is that it should be able to communicate with users in different languages. On the *Figure 4* the structure of modules that are needed to realize a multi - language system is shown. To be able to "speak" in some language, the system needs to use a proper text-to-speech library that will allow speech, and the xml file with the described program model in this language has to be prepared in advance. When the language is changed, the xml and text-to-speech library have to be changed as well. Also the GUI module should support the specific language's encoding, to be able to show the text.

All information needed for the application is given in the XML file format. This information is processed and converted to a structured graph that contains information about the used icon and lexical meaning of it (represented as a word for given language).

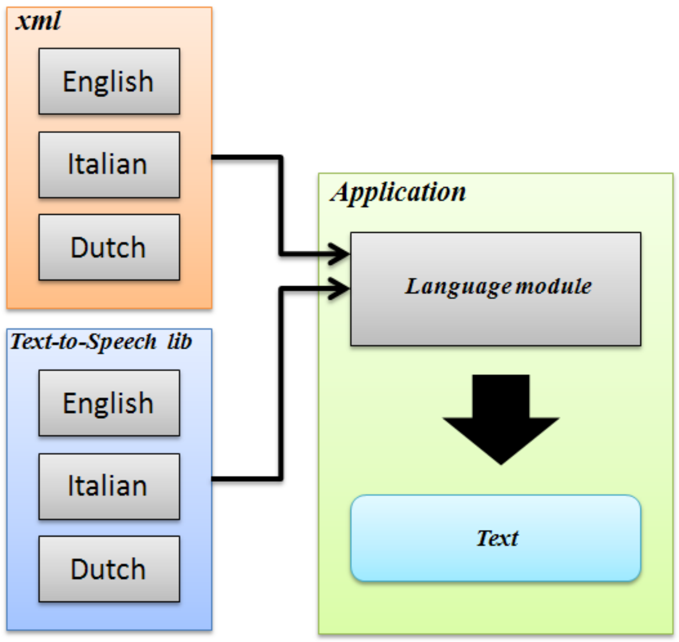


Figure 4. Multi-language system

There is another possible way to realize a multi-language approach and it works by using Resource Bundle [1] (in Java). In this case, the XML file will only contain the ID for given the icon (instead of its lexical meaning). The developer should create the property-file for every possible language that will keep the icon's lexical meaning in the given language according to the icon ID (for example: *id_1 = food; id_2=people*). When the language will be changed the system's local will be changed as well, and system will load another property-file according to the new language.[2]

In the *Figure 5* an explanation of how modules are located and how information passes through them is given. In this case the XML parser is separated in the external module, and the serialized file is presented as the output format, that is the input data for my system. The point here is to take the parser out of the application's scope as it is "heavy" resource for mobile devices. But separating the parser's part from the main module makes it obligatory to use a serialization process. That's why we bring the deserialization part inside of the application's scope. We use the temporary type data to transfer data in between. (As is shown on the picture, I just have to prepare the file and send it to the mobile device as a resource for our application). This version of my system uses this approach.

[1] Resource bundles contain local-specific objects. When your program needs a locale-specific resource, a String for example, your program can load it from the resource bundle that is appropriate for the current user's locale. In this way, you can write program code that is largely independent of the user's locale isolating most, if not all, of the locale-specific information in resource bundles.

[2] This is not implemented in my system, it is described here only as one of possible opportunities for implementation.

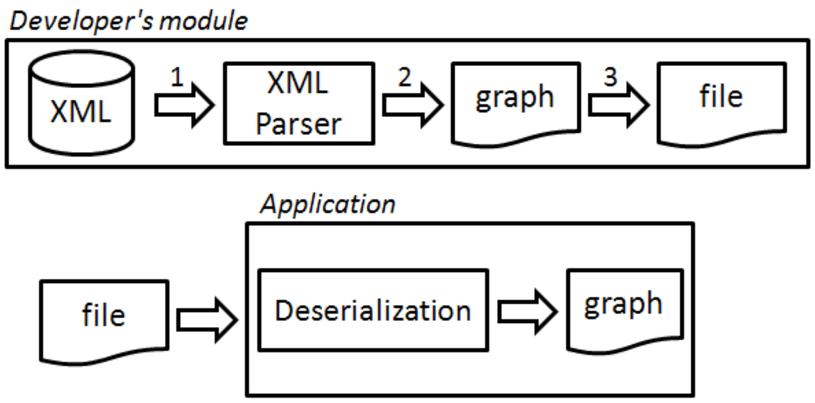


Figure 5. Application using serialized file

On the *Figure 6* I wanted to show another possible scheme of application where the XML parser is included into the system module. In this case the serialization/deserialization process is not used; we avoid using temporary data type format as serialized file. In this case XML represents the input data for the application's module where it is parsed to the tree and directly used by the GUI model.

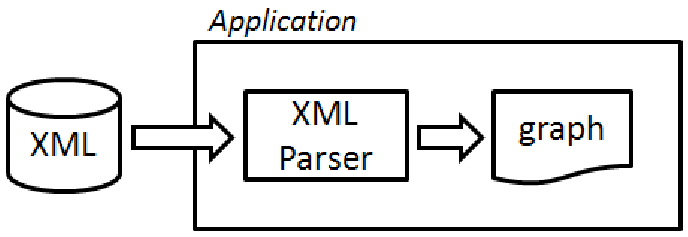


Figure 6. Application with included parser

In this case the application will need more recourses because every time the user will change the main category of dictionary, a new xml will be parsed (if this category used for the first time) instead of the deserialization process being performed in the previous case (*Figure 5*).

Structure of data.

My idea was to build the graph structure that includes all necessary information for our application. All categories and words in the dictionary application should be ordered into the graph. Every node of this structure contains information: grammar type (verb, noun, pronoun etc.), grammar value, sentence's type (positive, negative, or question) image, reference to other possible words for the next choice (set of next possible icons in user interface). Once the graph with all the important data is created – all we need is to work with it, to go down and up through this graph. When the user chooses some path in the graph, the sentence is generated automatically, because every structure's node has its grammatical meaning (value); it depends on the chosen path, the required sentence is created in the right grammar sequence.

In the case that we have a big and complex hierarchy for the dictionary, we can escape describing a huge xml. Instead we can describe main categories as every category is the root for a sub-graph represented as a separate xml file. And when the user will make his first choice of category, only those xml files will be used (see **Error! Reference source not found.**). After he/she will chose another category – another xml will be processed and new sub-graph will be created and added to the already existing one. This concept will help us to save some of the device's memory.

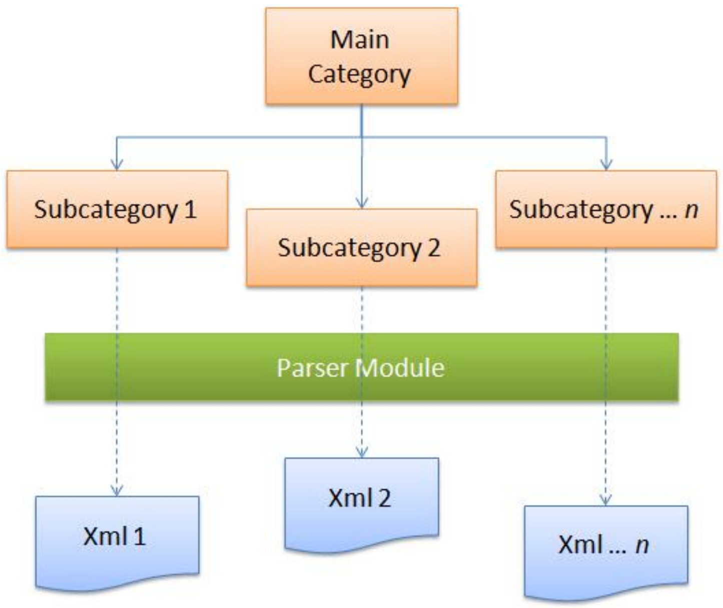


Figure 7. Categories using separate XML files

3.2. Plan

In this chapter I will define a number of steps to create our application. They are: definition of basic sentences that the system needs to offer to the user; creating the model that should order sentences in a structured way; definition of a program model for representing model; definition data to XML file format; processing xml file and creating GUI representation of this data.

3.2.1. Define basic sentence set

The first thing I should do is to think about traveling aspects and to see which problems and questions can meet the usual traveler, to select the most important from them and to define them as sentences[3]. The sentences have to be simple and well described in order to help the user to express his/her idea clearly and to avoid troubles during the work with the system.

Persons with a specific function and in a specific context will use the same sentences more frequently than other. Via interviewing and recording there will be a database with most frequently used sentences. For example it can be expected from a tourist that he needs information about how to go to the center, restaurant, stations and etc.

I defined the main categories that can be useful for one traveler. Some of them are:

1) Money
   - Bank
   - Exchange
   - Cash-dispenser
2) Transport
   - Train
   - Bus & Tram
   - Taxi
   - Plane
3) Border crossing
   - Passport
   - Customs
4) Hotel
   - Reservation
   - Rate
5) Shopping
   - Information
   - Price
   - Clothes and shoes
   - Consumer electronics
6) Food and drink
   - Restaurant
   - Drink
   - Meal
7) Health
   - Doctor
   - Dentist
   - Pharmacy
   - Hospital
8) Sport
   - Game

   Playing sport
9) Communications
   - Internet
   - Post Office
10) Where to go
   - Walk
   - At night
11) Society
   - Feelings
   - Opinions
   - Invitation
12) Location.

[3] In the application I will use international language – English. To make system available to use another language, described steps have to be performed for this language according its specifics.

- Service

The next step is to select key-words in every sentence that can help to split sentence by meaning and representing it with icon concepts.

For example a few sentences from category *Society* -> *Feelings:*

- <I> am (not) happy. (key: society, feelings, (-), <person>, mood, happy)
- Are <you> happy? (key: society, feeling, (?), <person>, mood, happy)
- <I> am (not) tired. (key: society, feelings, (-), <person>, body, tired)

For example a few sentences from category *Society* -> *Opinion:*

- Do <you> like it? (key: society, opinion, (?), <person>, like, it)
- I like it. (key: society, opinion, person, like, it)
- I think it is beautiful. (key: society, opinion, it, beautiful)

In these sentences the "< person>" means that every option of persons can be applied (I, you, he, she etc.); (not) – means that the same sentence can be negated; and (?) shows that the sentence is a question.

3.2.2. Designing model

After I split basic sentences to key-words, I have to try to order them in a structured way. For this reason I have to create a model for every category of dictionary (keeping some aspects together). The model I create should arrange data by visual representations and logical meanings at the same time.

In the *Figure 8* a model for the category "Society" of my dictionary is shown. The model is represented as graph structure.

Every element in this graph is shown as a rectangle; all properties for a given node – with oval figures. It means, that if we have to choose some value for a person (person is a node of graph) and it has a list with properties that can be chosen from the user (I, you, he, etc.). All values from the property list have a common grammar type and respond to the same grammar rules.

A model will help us to describe data in an XML file and program realization as well.

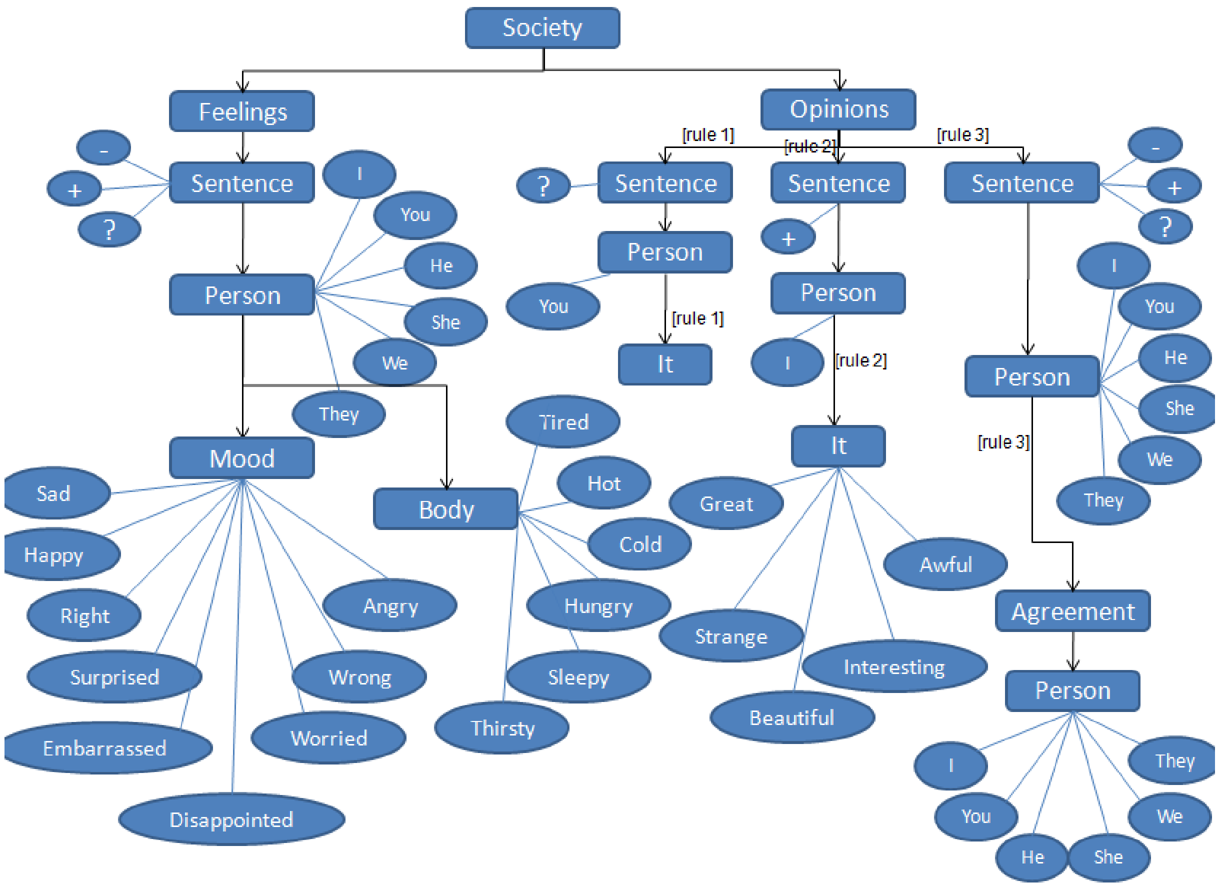


Figure 8. Ontology model

Problem definition: It is difficult to make a structure which should respond to a few aspects at the same time. These aspects are: program representation of the data (how it will be better to process information in the graph), the user's view point (in which order to present words to user to make it easy to understand) and the logical aspect (the sentence will be created from graph's nodes and the order they are arranged in in the graph is still important).

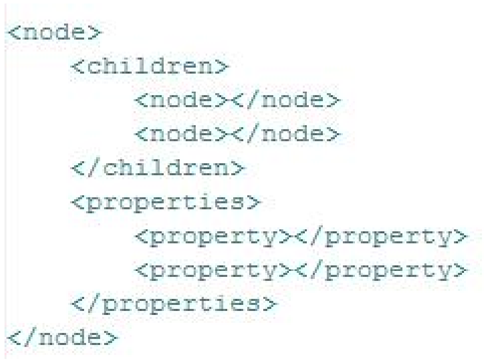
3.2.3. Define program model for data representation

Now I have to describe program components that will obtain (represent) my model in program way. At this step I start to think about the implementation part of our project, because I need to define programming structures which will represent information inside of the system. To see classes that I'll need for the graph see the explanation in the Implementation chapter (*Figure 18*).

3.2.4. Create xml file

The next step is to describe the data for the application based on the created model. Using the advantages of the XML file format I can define a tag's name that will give me a hint about its content.

With <node> I tag the element of the model (graph) that can contain children from the same type. Children nodes for current element are described in the <children> tag. With the <property> tag I mark every possible value for the current node. Usually they are more than one (zero or more than one), that's why I use another tag <properties> that represents set <property> tags. There are some cases in my model when the graph should be created according specified rules (you can see it



```
<node>
    <children>
        <node></node>
        <node></node>
    </children>
    <properties>
        <property></property>
        <property></property>
    </properties>
</node>
```

Figure 9. XML file structure

in the *Figure 8*). Due to these reasons I use additional tags like <rules>, <rule>, <extras>, <extra>. These tags will be interpreted by a parser for creating specific paths in the graph.

3.2.5. Process xml file

To be able to use the data described in the XML I need to parse it to the programming object. For this reason I will implement a SAX parser (using Java API) that will read my xml file step by step and will build the graph gradually. The scheme of the parser is shown on *Figure 10.*

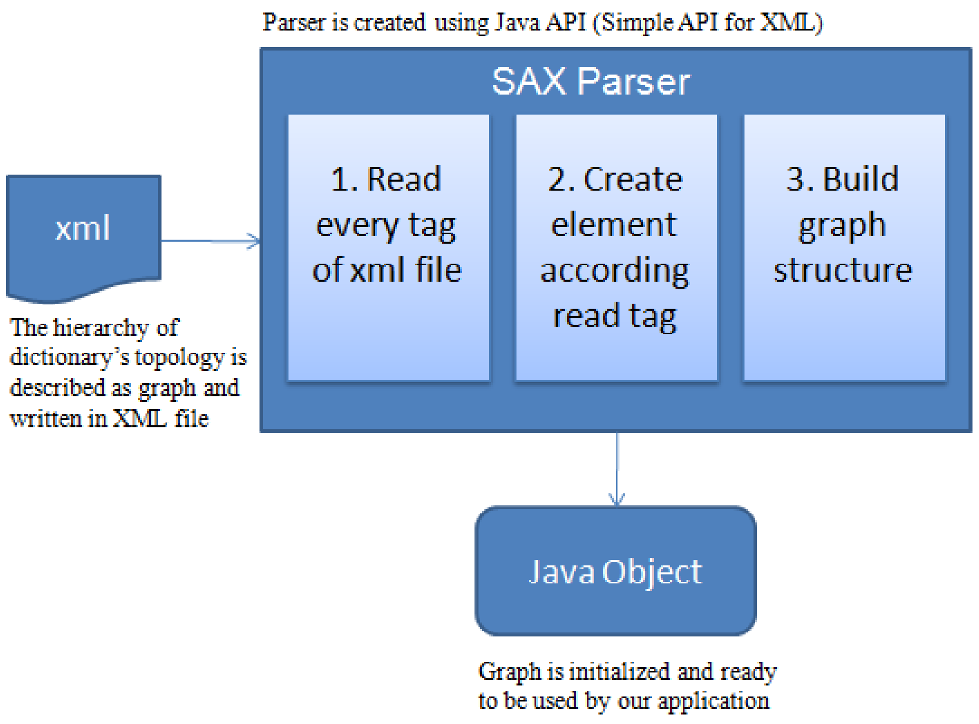

Parser is created using Java API (Simple API for XML)
SAX Parser
xml
1. Read every tag of xml file
2. Create element according read tag
3. Build graph structure
The hierarchy of dictionary's topology is described as graph and written in XML file
Java Object
Graph is initialized and ready to be used by our application

Figure 10. SAX Parser

As a result of this process I have a graph object that can be used and processed. Keeping in mind that the parser is separated in an additional module, I need to convert and store this graph as a temporary data type. This can be done through converting the graph to a binary stream and storing it in the file (serialization in Java). This file should be added to main application sources and be installed on the mobile device. When the application will be run, the file containing serialized graph object will be read, deserialized and delivered to GUI model to be processed.

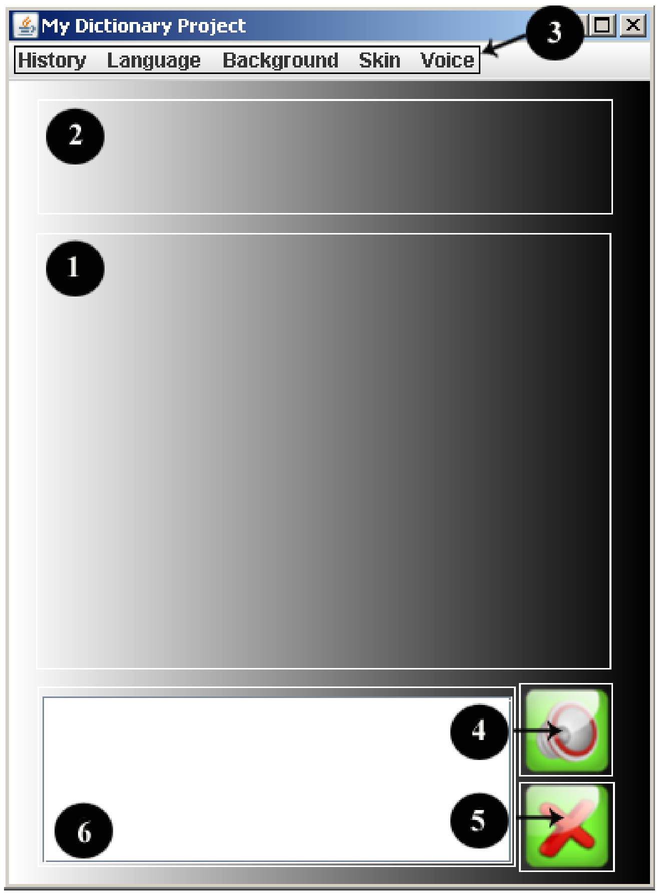
3.2.6. Create user interface.

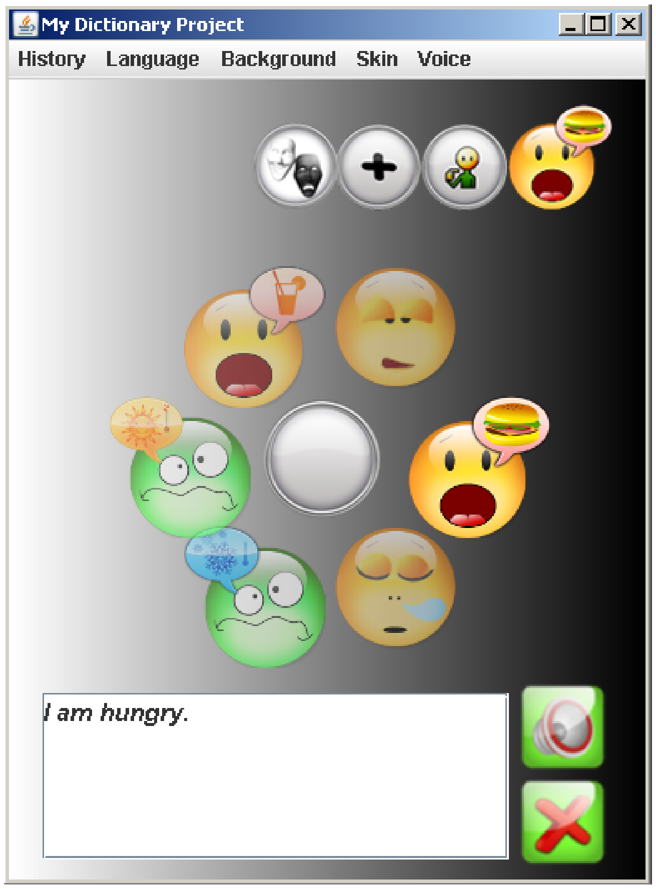
In this step I need to think about creating a good looking user interface. The implementation part will include only binding the GUI model with the graph and the interaction between them.

In *Figure 11a* main user interface is shown. The main window's areas that are marked with numbers:

1) Main area for representing next possible group of icons. Every time the icon is selected the new possible options will be calculated and shown on this area.
2) Previously selected icons list are kept in this area. Each time the user will select an icon, it will be automatically be added to this list. We need to keep all chosen paths in the graph to give the user the chance to go back in the graph.

3) Application menu. The menus are grouped according to their functionality. The menu allows the user to make changes in the currently used language, voice, and preview background of the interface. These options are described in more detail in the use-case diagram.
4) Button for text presentation. By using the button on the bottom panel the user can activate the current text-to-speech synthesizer. The synthesizer will read the translation of the resulted text that is displayed in the text area aloud. If the sentence is not created and text area is empty, the synthesizer will do nothing.
5) Button to clear the text area's content.
6) Text area for representing the created text. Every time an icon is selected the program code will "convert" the icon into the word and add it to the current sequence of the user's input words.





I am hungry.

Figure 11. a) User Interface form model b) User Interface form prototype

In the *Figure 11b)* a simple example of a prototype of the application is shown. The generated text is "I am hungry", the path of selected icons is "Feelings" ->"Positive" -> "I" -> "Hungry". "Feelings" in this case is category, "Positive" shows the sentence's type - positive, "I" is selected from the category representing people, "Hungry" represents the body's property.

4. Used Technologies

4.1. XML

I had chosen the XML data format for storing input information. It is useful for storage information, for describing structures data and it is easy to work with.

4.1.1. Why XML?

XML is a markup language for presenting information as structured documents. Structured information contains both content (words, pictures, etc.) and some indication of what role that content plays. The language has been developed from SGML (Standard Generalized Markup Language, ISO 8879) as an activity of the World Wide Web Consortium (W3C). It is very lightweight, as opposed to a traditional database, and can be edited relatively easy. XML allows developers to easily describe and deliver rich, structured data from any application in a standard form. All XML files should be well-formed, meaning that they should obey certain grammar rules of XML.

The primary usage of XML is in a data-centric model. In a data-centric model, XML is used as storage or interchange format for data that is structured. It appears in a regular order and is most likely to be machine processed instead of read by a human.

XML also provides an expedient way to describe structured data thus making it important as a data storage and interchange format. XML provides many advantages as a data format over others, including:

1. Built in support for internationalization due to the fact that it utilizes unicode.
2. Platform independence.
3. Human readable format makes it easier for developers to locate and fix errors.
4. Extensibility in a manner that allows developers to add extra information to a format without breaking applications that where based on older versions of the format; tags can be defined in specific application;
5. Large number of off-the-shelf tools for processing XML documents already exist.[4,10]

4.1.2. XML Parsers for Java. SAX Parser

As XML is created for storing structured data, the necessity of translating data to program representation appears. Efficient parsing of XML documents is more and more critical as XML gets adopted more widely. It is important to have an efficient way to parse XML data if you want to read and manipulate data stored in the XML document. Several types of XML parsers are available. [4] Which one is right for your situation is dependent on your system’s goal.

- **DOM (Document Object Model) Parsing.**
  DOM parser creates a tree structure in memory from an input document that allows the developer to access the XML document randomly. Using DOM is straightforward. The DOM APIs allow modification of the nodes, such as appending a child or updating or deleting a node.

  The XML document has to be parsed at one time; partial parsing is not possible. Loading the whole document and building the entire tree structure in memory can be expensive, especially when the document is large. It can be not effective if you will process huge XML because of a lot of memory is needed.

- **SAX (Simple API for XML) Parsing.**
  SAX is an event-driven push model for processing XML. The parser doesn't create any internal representation of the document. Instead, it calls handler functions when certain events (defined by the SAX specification) take place. These events include the start and end of the document, finding a text node, finding child elements, and hitting a malformed element.

DOM vs. SAX.

Both parsers have very different approaches for processing the information. DOM specification provides a very rich and intuitive structure for housing the XML data, but can be quite resource-intensive given that the entire XML document is typically stored in memory. SAX parser can scan and parse gigabytes worth of XML documents without hitting resource limits, because it does not try to create the DOM representation in memory. [11, 12]

As a result of the discussion above I prefer to use the SAX parser in the application we try to develop due to the reason that PDAs have restrictions of memory that applications can use.

4.1.3. Serialization and Deserialization

Serialization is the process of taking an object and converting it to a format in which it can be transported across a network or persisted to a storage location. Java provides classes to support writing objects to streams and restoring objects from streams. Only objects that support the java.io.Serializable interface or the java.io.Externalizable interface can be written to/read from streams.

[4] I will discuss the XML parsers available for Java.

Deserialization is the process of using the serialized state information to reconstruct the object, to extract the data from a series of bytes in order to get object back into its original state.

Why do we need to use serialization? The process of serialization allows an object to be serialized, shipped across the network for remoting or persisted in a storage location. The application needs to use serialization for transferring data from an external module that contains the parser to the main application's module (*Figure 5*). The xml is parsed into the graph in the external module; that's why the graph is serialized into the file that can be transferred between modules. Receiving this file the application has to be able to read the information and to convert it to the original object (graph) in order to use it. The deserialization process is used for this purpose. [13]

4.2. Java

Java technology is an object-oriented, platform-independent, multithreaded programming environment. It is the foundation for Web and networked services, applications, platform-independent desktops, robotics, and other embedded devices. Running across all platforms - from servers to cell phones to smart cards - Java technology unifies business infrastructure to create a seamless, secure, networked platform for your business.

4.3. JavaFX Script Language

I was looking for the proper technology for creating a demo application. It should be a technology that will give the opportunity for creating a good looking user interface. I found a completely new technology that is still under development – JavaFX, scripting language, based on Java. It makes it easy to develop rich and responsive graphical user interfaces for internet applications, desktop applications and for mobile devices as well. Most mobile devices initially have a java platform installed that allows running java-based products on it. This language gives a lot of possibilities for developers to manipulate GUI-components like rotations, shadows, animations etc..

About JavaFX.

JavaFX Script is one of a family of JavaFX products from Sun Microsystems. It is a scripting language that runs on top of Java™ Platform, Standard Edition 6 (Java SE) and makes it easy to code sophisticated user interfaces. The language is highly portable and can run on any system that supports Java technology, without local installation. It uses underlying Java technologies to let you create GUIs of any size or complexity easily.

JavaFX Script is a statically typed language, which means that the data type of every variable, parameter, and method return value is known at compile time. JavaFX Script is also a declarative programming language: it describes what the application is like rather than how to create it. The algorithm that determines how to display the application on the screen is left to the support software (Swing's Java 2D APIs). Because of these traits, JavaFX Script is well suited for GUI creation.

The main purpose of JavaFX technology is developed for building and delivering the next generation of rich Internet applications for desktop, mobile, TV, and other consumer platforms. To bridge the gap between user experience design and programming logic, JavaFX technology will provide a suite of tools and authoring solutions that enable unprecedented collaboration between designers and developers.

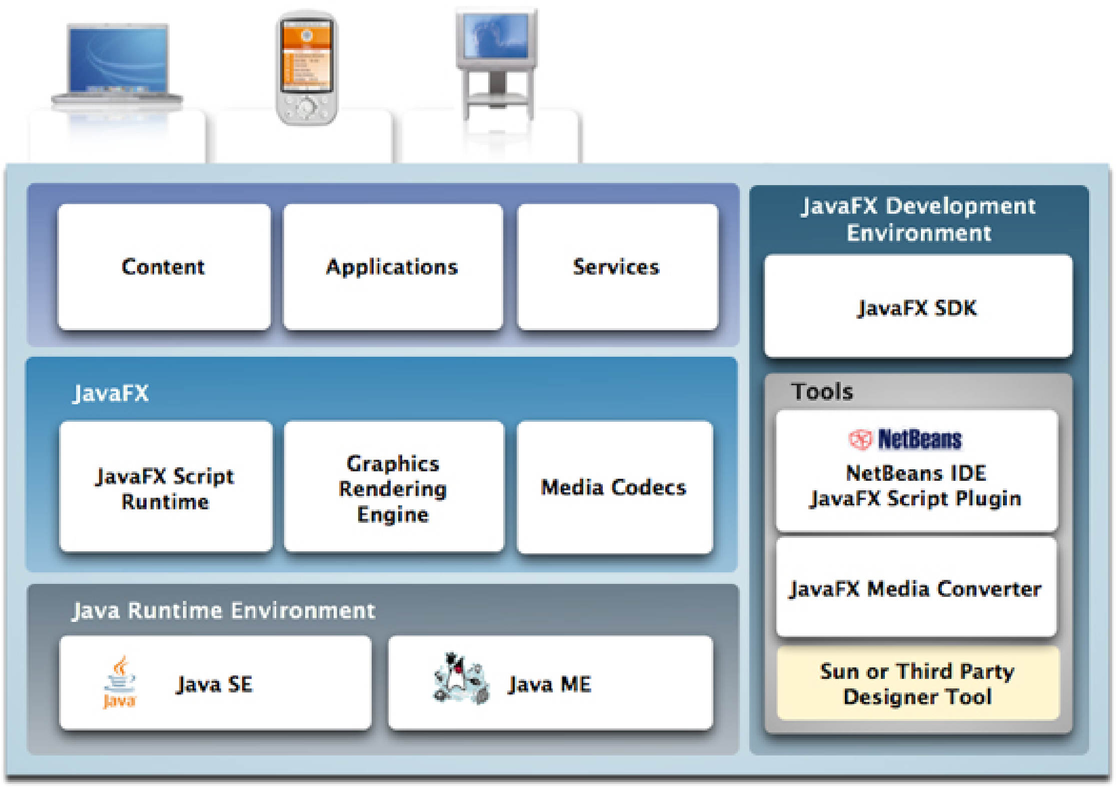


Figure 12. JavaFX overview

Sun Microsystems is bringing JavaFX technology to mobile devices – JavaFX Mobile product is expecting to be released soon, that will provide a platform for PDAs, smartphones and feature phones. It is understood that Sun will distribute JavaFX Mobile as a binary operating system to device manufacturers who will brand the interface to differentiate their product. [7]

4.4. XML, Java, JavaFX Integrating

As described above, JavaFX is compatible with Java language. The developer can mix and use syntaxes of both languages together according his needs. The XML is read and parsed using Java API for parsing (SAX parser) into a Java object. This object (in my case it is the graph structure) will be used later in the GUI model and will be processed with JavaFX Scripting language. Converting data from program object to byte-stream and back to original state is performed also using Java API. That’s why there is no problem integrating both of these languages.

5. Implementation

In the ideal case we would like to have a system that will communicate with users in different languages and that has text and speech in appropriate (current for the system) language as a result. The interaction between user and system can be implemented as easy and clearly as is possible in order to allow comfortable use of the system by the user.

5.1. Project architecture

In the *Figure 13* I tried to show the main flow of information via program modules and processes. As mentioned above, the initial point of the process is the XML file with all necessary information that our application will need. The xml is parsed to graph program object and after this is serialized into the byte stream and stored in the file. From this moment the file represents the input data for our application that has to be included into the system module directory for installation on the mobile device.

As our application knows which format for initial data to expect, it reads and deserializes the data from the file into the graph (original state of the data). After the graph is restored, it can be used by the interface module for interaction with the user. As a result of graph processing the text and speech is created and presented to the user.

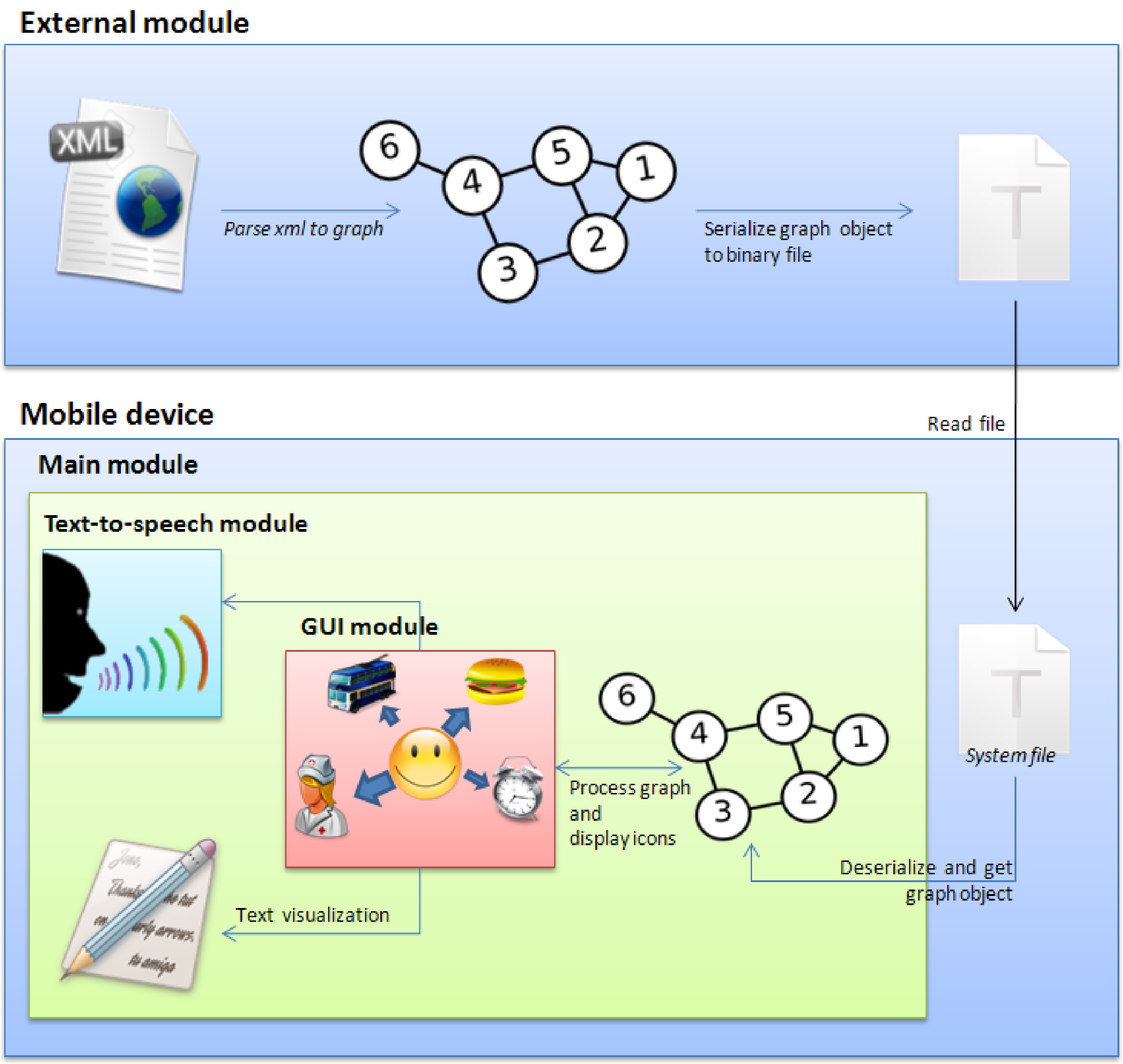


Figure 13. Project Architecture

In the *Figure 14* the interaction between actor (traveler in our case) and our system is shown. The user selects an icon on his mobile device, the next possible icons are calculated and sent back to the main window. In the same time the chosen word is put into the sentence's holder container into the proper slot (*like subject, prefix, infix, verb, object* and etc.) according to the grammar type of the word. This way the sentence is created step by step and validated via the grammar rules.

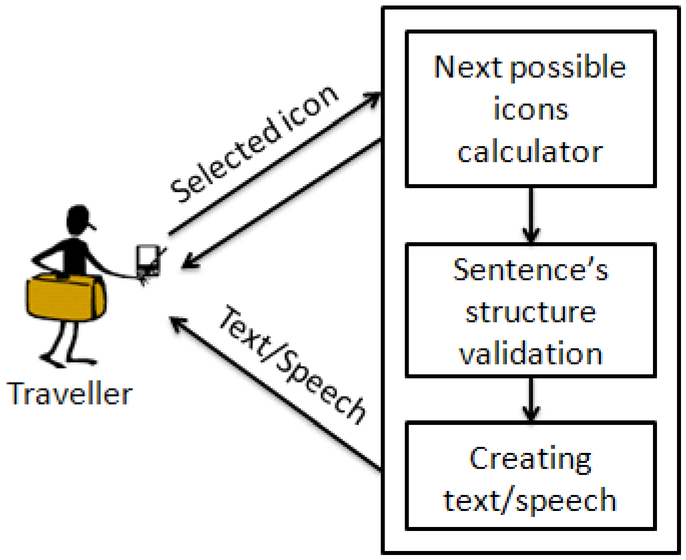

Selected icon
Next possible icons calculator
Sentence's structure validation
Creating text/speech
Text/Speech
Traveller

Figure 14. Actor-System interaction

5.2. UML

The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.

5.2.1. Use-case diagram

A use-case diagram is used to identify the primary elements and processes that form a system. The primary elements are termed "actors" and the processes are called "Use Cases." The Use Case diagram shows which actors interact with each Use Case. Each Use Case is named using a verb phrase that expresses a goal the system must accomplish.

Use Case relationships.

After we've defined the system, actors, and Use Cases, we need to associate each user with the system features they need to perform their jobs.

- ***Association relationship***. The association represents the fact that the actor communicates with the Use Case. This is the only relationship that exists between an actor and a Use Case. An Association relationship is drawn as a filled line.

- ***Includes relationship.*** An Includes relationship is used to indicate that a particular Use Case must include another use case to perform its function.
  When one Use Case delegates another the dependency is drawn as a dashed arrow from the "using" Use Case to the "used" Use Case and labeled with the <<include>> word. It can be used in two cases: a) another Use Case may already exist to perform the task that is needed; b) a number of Use Cases may need to perform the same task and rather than write the same logic multiple times, the common task is isolated into its own Use Case and reused by, or included into, each Use Case that needs it.
- ***Extends relationship.*** An Extends relationship says that one Use Case might need help from another Use Case. Somewhere in the logic of the Use Case that needs the help is an extension point, a condition test that determines whether or not the call should be made. There is no such condition in an include dependency. The Extends dependency arrow points from the extension Use Case (the one providing the extra help) to the main Use Case that it is receiving help and is drawn as a dashed arrow with label <<extends>>.

  In contrast, the Includes dependency says that one Use Case will *always* call the other Use Case. The other contrast between the two relationships (Includes and Extends) is the direction of the dependency arrow.
- ***Generalization relationship.*** Inheritance is a key concept in object-oriented programming and tells us that one object has, at the time of its creation, access to all the properties of another class, besides its own class. Thus, the created object incorporates all those properties into its own definition. The same idea, applied to actors and to Use Cases in a Use Case diagram, is called generalization and often goes by the nickname, an "is a" relationship.
  To model generalization, the UML uses a solid line with a hollow triangle. The triangle is always on the end near the item that is being inherited. [8,9]

Let's define the main actors and main actions for our application.

We have only one actor – the traveler. Extending the application with new functionality we can add new actors. There are use-cases for travelers shown in the *Figure 15*. It shows all possible actions for our users.

1. He/she can change the language of generating speech and text. This is the main function that our system has to allow its users. [5] If the user will choose this option after he/she builds half a sentence, the main GUI-form would be refreshed and restored to the start point that will allow the user to change the main category again. This is necessary because the system has to change the using library for text-to-speech module and to use a new xml file with the proper language. After this the application is working as usually.
2. The first step that the user has to perform to build a sentence is to choose a main category. After the system is run, there are a few icons on main view area that represent the

[5] The current prototype of application doesn't allow this possibility to use multiply languages, because text-to-speech libraries usually are made by commercial organizations and can't be used for free, but in ideal case it should be done. In my report I would like to describe as detail as it is possible the project overview to give possibility for someone else to elaborate my ideas and to use them in another projects.

categories of the dictionary. After the user selected one of them, the proper xml (with current language’s properties and sub-graph for representing structure) is loaded. Now the program object is created and can be processed by our application via user’s actions.

3. He/she can choose icons to create sentences or to go back to some of the already selected icons in order to return back to an old position in graph. Every time the icon is selected (no matter if it is selected from a list with previously selected icons or from next possible group of icons) the current node of graph corresponding to the selected picture is processed. The program code verifies if the node has properties and children, calculates positions on the screen and displays the next possible group of icons. At the same time the sentence is being appended with new parts and the grammar checking process is operating and the text field of GUI-form is updated with new context.
4. The user can also activate speech reproducing by pressing the appropriate button. As a result of this he/she can hear the generated sentence as a speech with current voice’s properties. The user is able to change the voice’s type and the possible options are different men’s and women’s voices.
5. The text area can be cleared by the user via pressing the clear button.
6. The user is also able to modify GUI-frame view. He/she can change the settings from the menu like background color or background image.

In the Use Case diagram the relations <<extend>> have the meaning that via some conditions the given Use Case can be performed or not. The <<include>> relation shows that one Use Case needs another one to perform its action. The relations between actor and Use Cases are realized with the association relationship. We have the generalization relationship in our Use Case diagram as well. It shows that two Use Cases have the same structure and meaning as another one, and they extended it with their specific action.

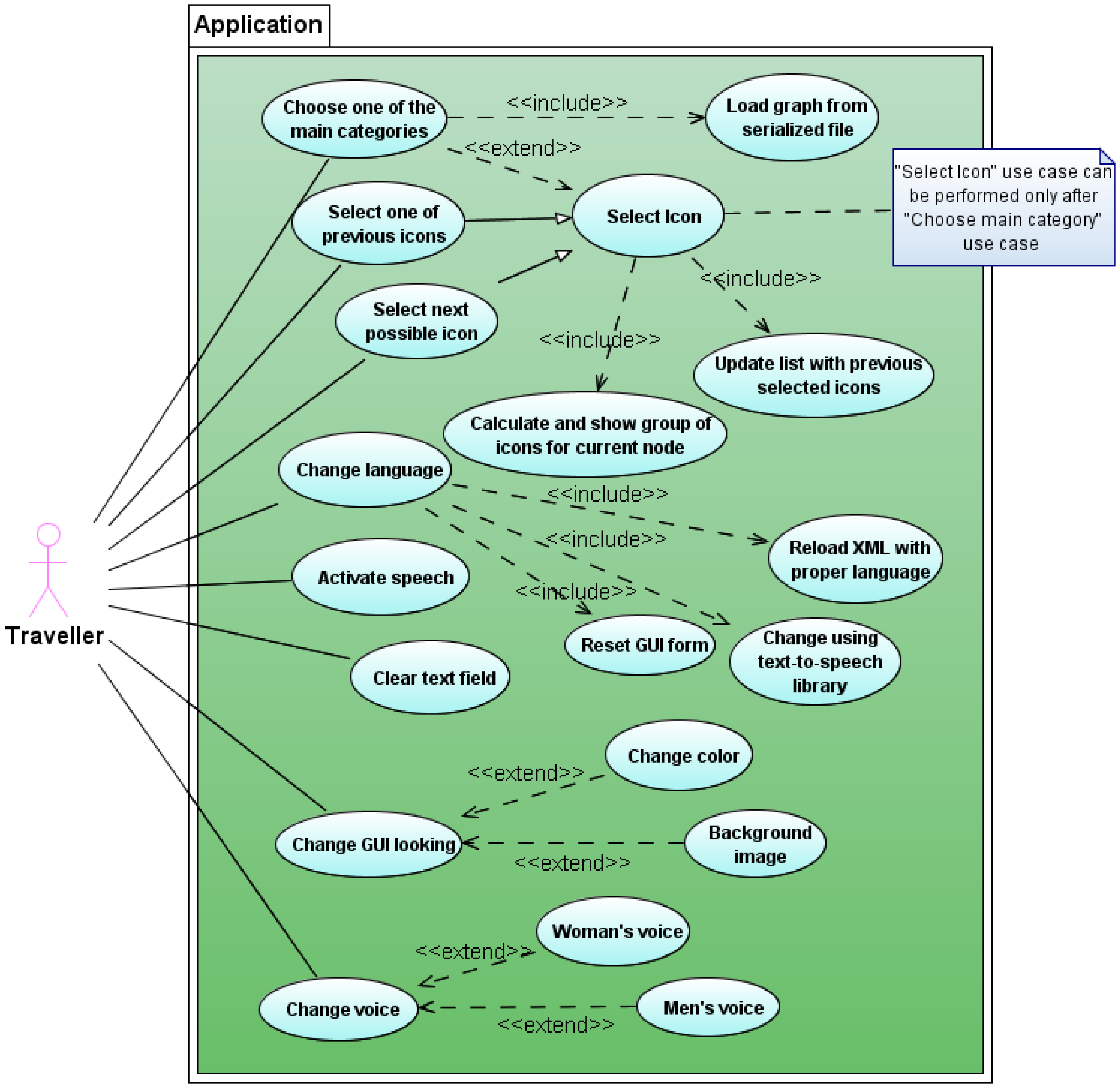
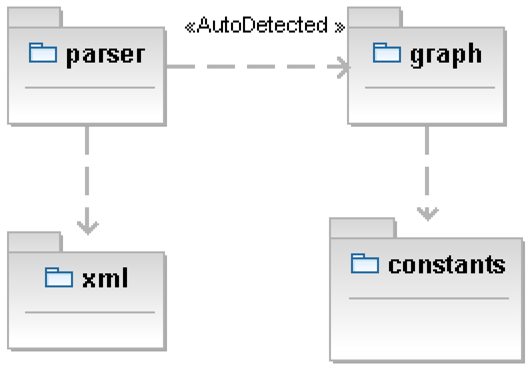

Application
Traveller
Choose one of the main categories
<<include>>
Load graph from serialized file
<<extend>>
Select one of previous icons
Select Icon
"Select Icon" use case can be performed only after "Choose main category" use case
Select next possible icon
<<include>>
<<include>>
Update list with previous selected icons
Calculate and show group of icons for current node
Change language
<<include>>
<<include>>
Reload XML with proper language
Activate speech
<<include>>
Reset GUI form
Change using text-to-speech library
Clear text field
Change color
<<extend>>
Change GUI looking
<<extend>>
Background image
Woman's voice
<<extend>>
Change voice
<<extend>>
Men's voice

Figure 15. Use-Case diagram for system

5.2.2. Class diagram

A class diagram is a diagram showing a collection of classes and interfaces, along with the collaborations and relationships among them. The structure of a system is represented using class diagrams. Using class diagrams we can see classes' attributes, methods and types of relation between them.

Our application consists of two projects: DictionaryLogic (*Figure 16*), that contains main sources and tools for graph representation and treatment, and DictionaryUserInterface (*Figure 19*) that contains all necessary sources for creating and processing user interface part of our project.



All classes from the "parser" package with their attributes and relations between them are shown in the Figure 19. Here we use the SAX Parser in order to parse xml that is an event-based approach. The parser scans through the XML data, calling handler functions whenever certain parts of the document are found.

Figure 16. Content of DictionaryLogic project

Class ***SaxGraphParser*** extends ***DefaultHandler*** in order to override necessary methods like *startDocument(), endDocument(), startElement(), endElement()* etc. to define the actions that the parser has to perform when it will find elements through the xml. For example when the parser reads the tag <children> it knows that new node of the graph has to be created and added to graph as a child-element. You can also see that the parser uses classes from another package, called "graph". This is because every node in a graph structure is represented via classes *GraphNode*, *Property* and etc. and the parser uses them during the graph building.

Class ***ParserManager*** creates an instance of an already defined parser (*SaxGraphParser*) and reads the xml file from a specified class-path. The xml file is read and parsed sequentially. The output of parsing is constructing a graph, which is a programming representation of the data we stored in an xml file. Now we can use this structure and process it in our application.

There are some cases in our model when the graph should be created according to specified rules. Due to this reason the rules are described in xml and are presented as the ***Rule*** class. All possible branches of the graph that depend on rules are described in the beginning of the xml file in the <extras></extras> part. Every separate sub-graph is described in the <extra></extra> tag and has its own ID. The parser reads the xml file from top to bottom, reading the first branch's part (if it exists), and after this - root contain. The extra elements are stored in temporary hash map, as key-value pairs: with the ID for the key and the sub-graph for the value. When the parser reads the xml's content and sees the <rule></rule> tag it will automatically get the branch with the proper ID from the hash table and add it to the current node.

The package "graph" contains all important classes for the graph's structure. Every element in the graph is represented as the **GraphNode** class that obtains a grammar value for this node. All classes representing the structure of the graph implement a *Serializable* interface that allows them to be serialized and deserialized.[6] The class *GraphNode* contains the attribute name that can be interpreted as an element's ID; the attribute value is the lexical meaning of this node and will be shown in the GUI part during the sentence's building; the attribute *imgPath* keeps the directory of the icon represented in this node and should be displayed on the screen; the attribute *visible* (true or false) shows if this node has to be displayed or just has a thematic meaning; the attribute *rules* represents a list with rules that has to be applied; the attribute *children* is a list with all children nodes for the current element of graph; the attribute *parentNode* keeps the parent node for current element and can be used in case we need to process the graph in both directions; the attribute *properties* is a list with all possible values for the current node. If we have multiple choices of values that can represent this thematic meaning of the current node, it will be represented by a *Property* object. After one property is chosen, its value is set as the node's value; the attribute *selected* indicates which property is chosen. The class **GraphNode** also contains a lot of methods for creating and processing the graph: *createChild(), appendChild(), appendProperty()* etc. They are needed during the xml parsing, when the graph is created. The methods *hasChildNodes(), hasProperties(), getImgPath(), getValue(), setValue()* and etc. are needed to process the graph and visualize icons. The class **Sentence** has a container's role for all parts of the sentence. When the user selects an icon, the corresponding word is filled into the proper "slot" (attribute) of the Sentence object. The grammar rules are applied to this object and a grammatically correct sentence is built. (You also can see *Figure 14. Actor-System interaction*).

[6] About serialization and desirialization see in the section 6.3 that described XML language.

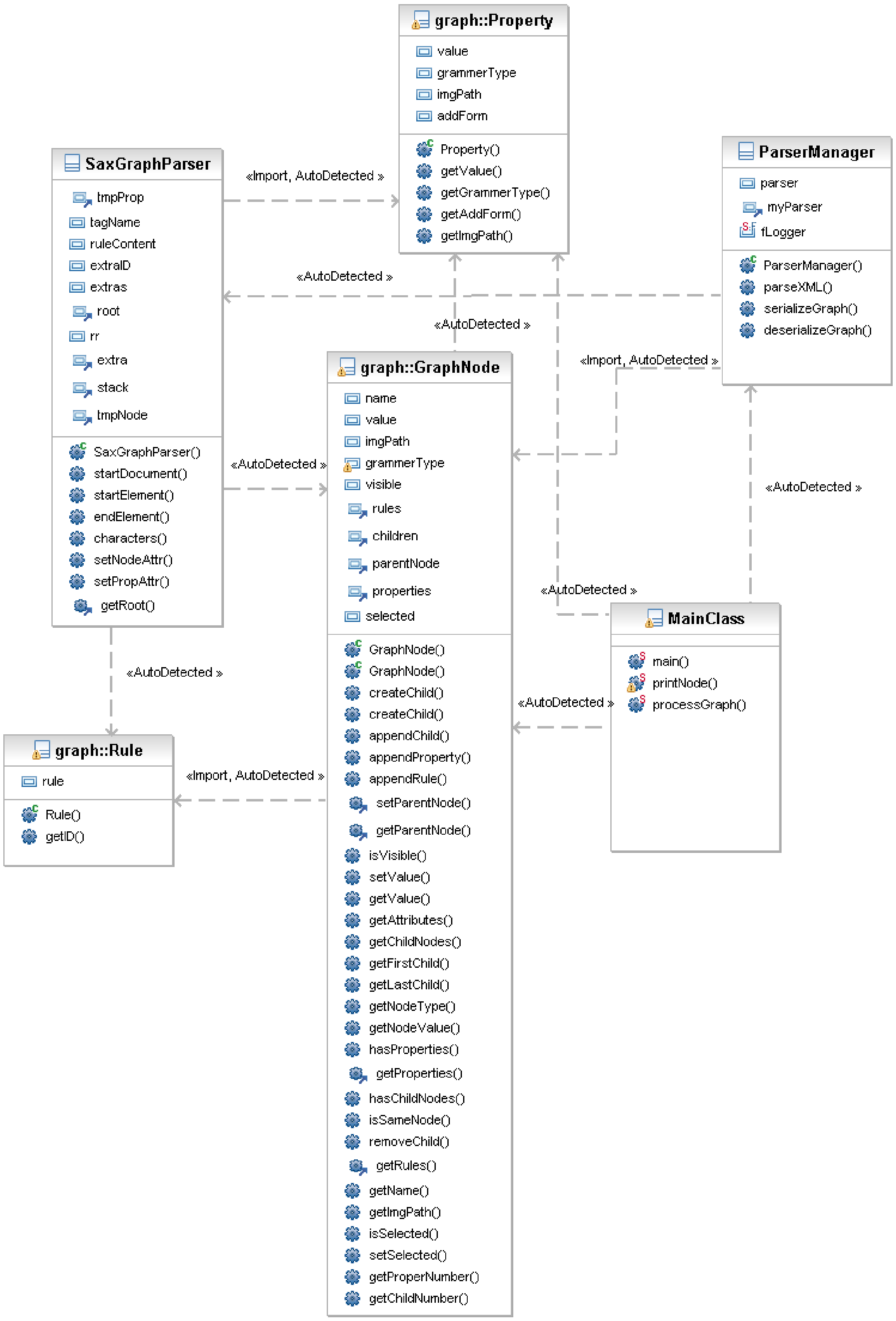


Figure 17. Class diagram for package "parser"

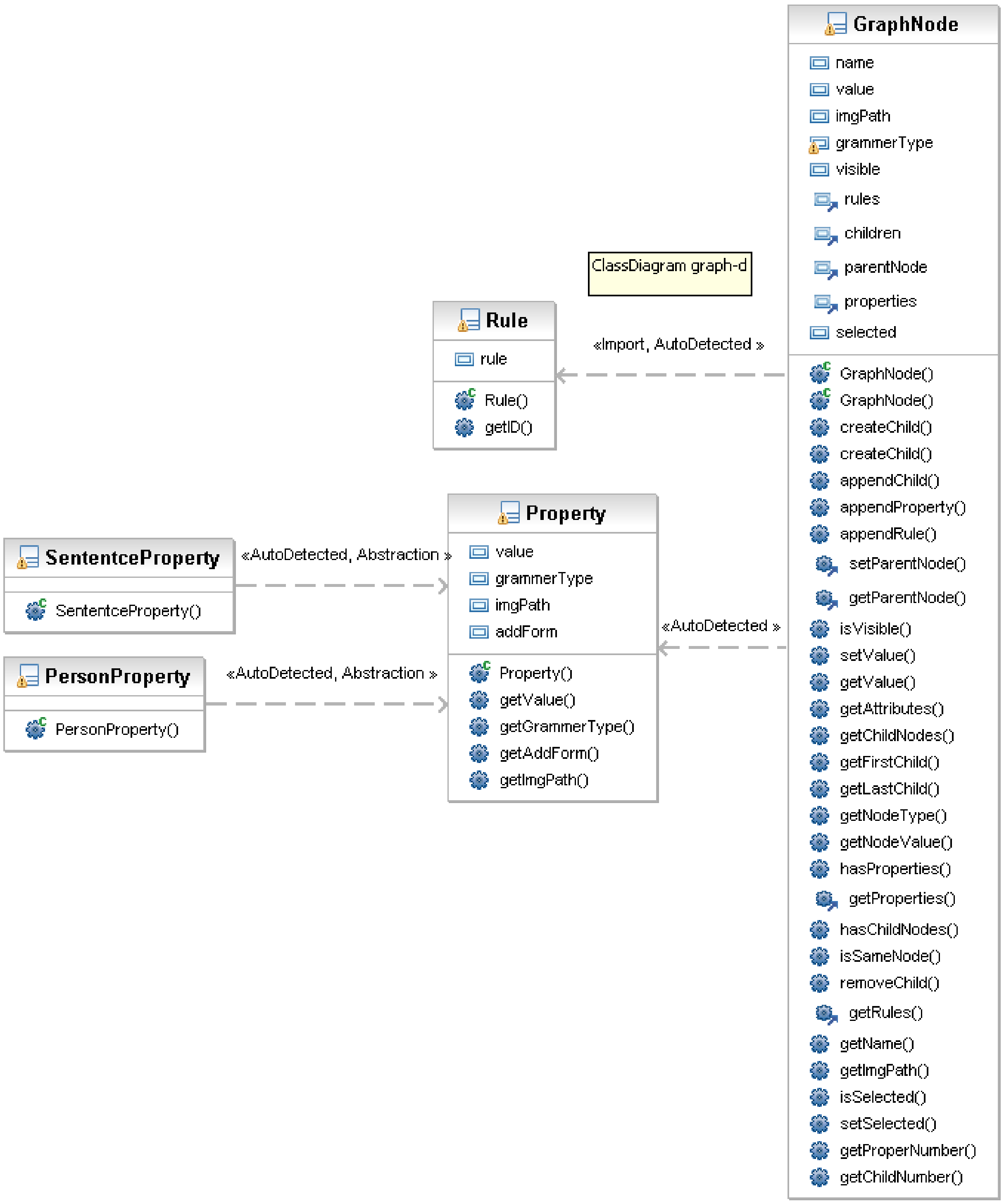


Figure 18. Class diagram for package "graph"

The content of DictionaryUserInterface project is shown in the *Figure 19*. Probably you noticed that there is the same package "graph" from the previous project. The DictionaryUserInterface is the center point of our application; its input data is a serialized graph, that's why this project has to include all needed classes for a graph structure in order to recognize the graph during the deserialization. It can be done if the DictionaryLogic project will be added as an external library or if our project will include specific classes (as it is in our case). The packages "graph" and "constants" contain the same classes that are described above.

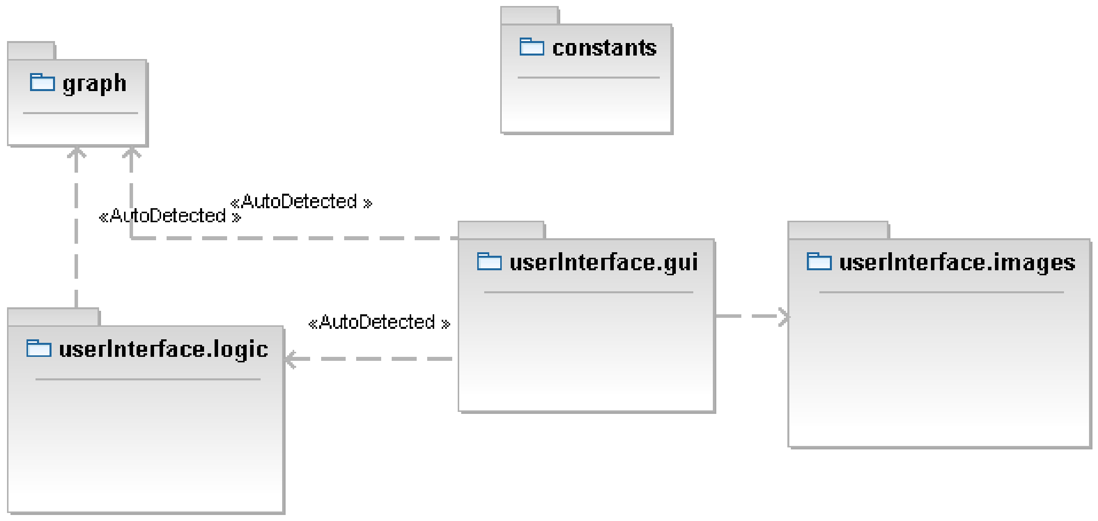


Figure 19. Content of project DictionaryUserInterface

The package **"userInterface.logic"** contains classes necessary for User Interface processing. The class **FrameSizeConstants** is abstract and contains constant values for defining sizes of the user's frame. The class **ReadXML** contains the method *read()* that reads the graph as a binary stream and reverses it to the program object (deserializes it). The class **VoiceClass** uses the ***com.sun.speech.freetts***[7] library for speech reproducing. The method *doSpeak()* is called every time the button "Speak" is pressed.

<u>About *com.sun.speech.freetts* package:</u>

FreeTTS is a speech synthesis system written entirely in the JavaTM programming language. It is based upon Flite: a small run-time speech synthesis engine developed at Carnegie Mellon University. Flite is derived from the Festival Speech Synthesis System from the University of Edinburgh and the FestVox project from Carnegie Mellon University.

There is class diagram for package **"userInterface.gui"** on the *Figure 20.* Class **StartGui** is the main class where the SwingFrame is created and all GUI components are initialized. Class **CreateButtons** creates buttons for activation speech and for clearing text field.

[7] I use this library in my demo. A set of text-to-speech libraries for available languages has to be included. Program we'll use the proper library for current system's language.

Class **SpeechButton** extends CustomNode and creates user's component. We need it to create our own buttons.

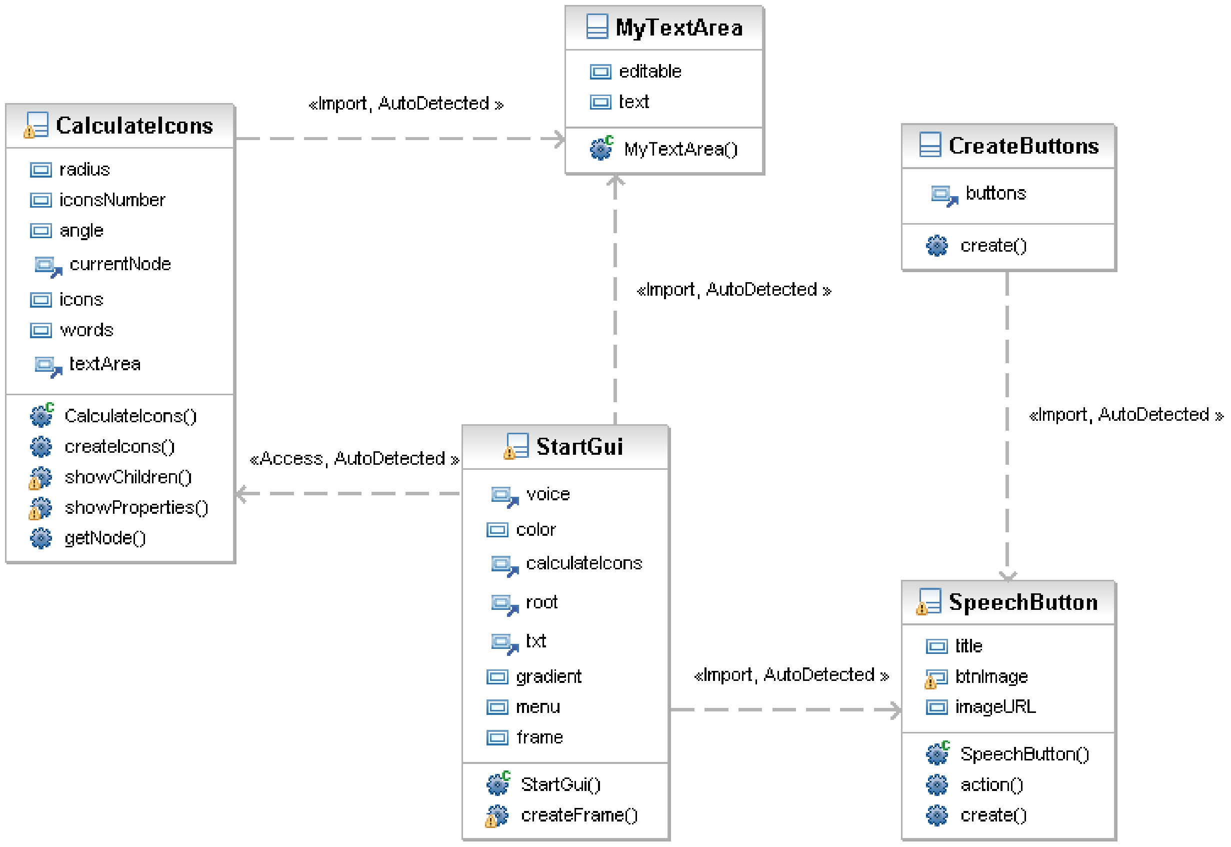
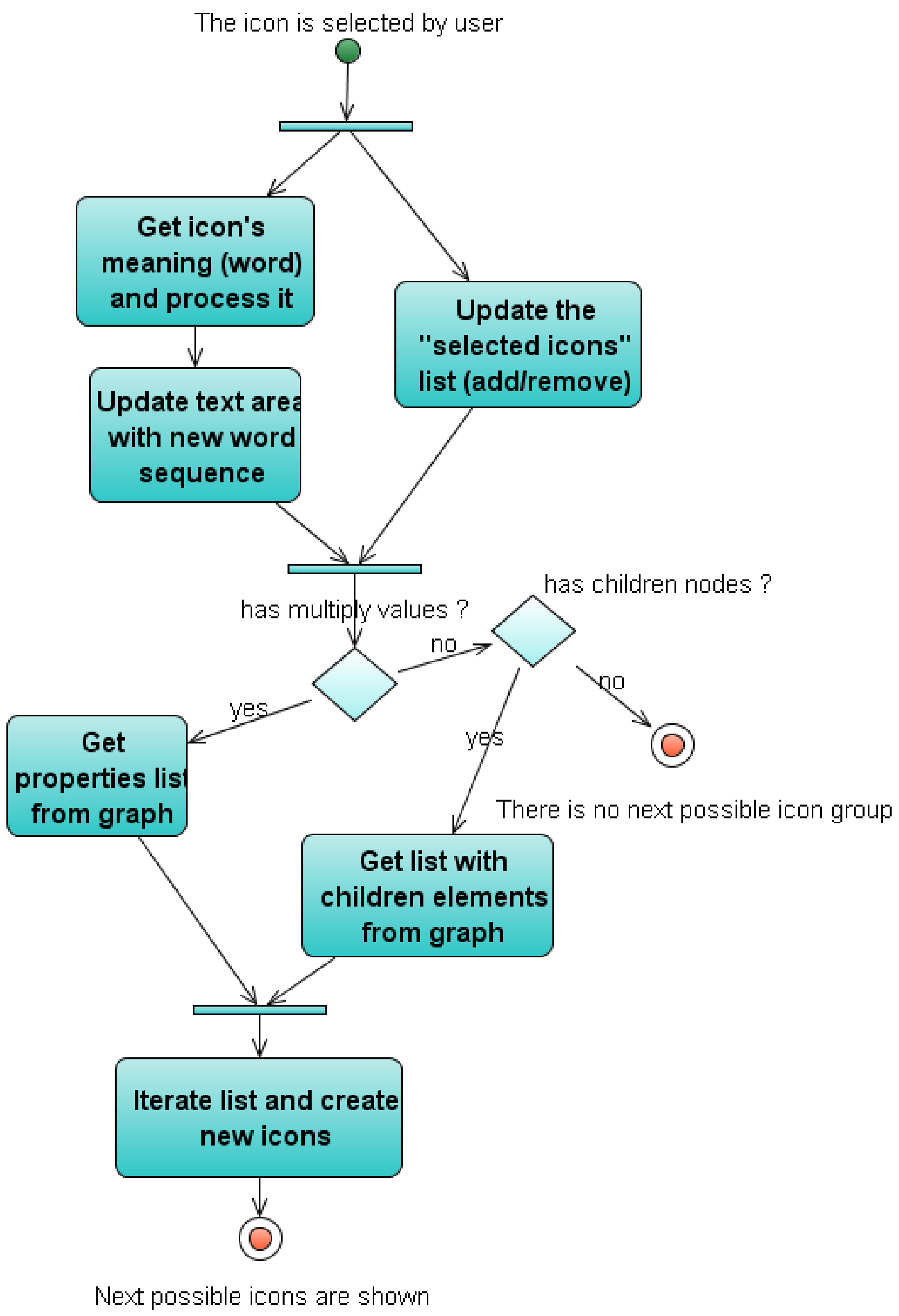


Figure 20. Class diagram for package "userInterface.gui"

5.2.3. Activity diagram

The Activity diagram describes processes including sequential tasks, conditional logic, and concurrency. This diagram is like a flowchart, but it has been enhanced for use with object modeling.



The icon is selected by user
Get icon's meaning (word) and process it
Update the "selected icons" list (add/remove)
Update text area with new word sequence
has multiply values ?
has children nodes ?
no
yes
no
yes
Get properties list from graph
There is no next possible icon group
Get list with children elements from graph
Iterate list and create new icons
Next possible icons are shown

Figure 21. Activity diagram for use case "Select icon"

In the *Figure 21* the activity diagram for use case “Select icon” is shown. With a horizontal line (fork) more than one action can be performed in the same time. There is one initial point (user selected icon) and two final points (the next icons are shown with success or there are no more icons to show and we are on the last level in our group – that means that the question is already created).

5.3. Testing separate modules

In this section some results of the implementation test are provided.[8]

To add more information to our discussion above (*Figure* 5 and *Figure 6*) we provide some test results that show time execution of separate modules. From the

Figure 22 we can see that time needed for parsing XML is much more than time for the deserialization process. It means that we chose the right approach to design our system with the parser separated from the external module. It clearly saves mobile device’s memory. We also checked the system’s reaction time. When the user selects an icon, the system (next icon predictor) has to check if the chosen element has properties (possible values) or children elements. The time for the system’s reaction depends on how many values/children the selected element has. Next, the icons predictor has to calculate coordinates for every icon which should be displayed on the main window, to create all icons and update the user interface. That’s why we can see varying values for the system reaction.

[8] This implement test was performed with the NetBeans tool that is used for creating application (both modules: external with parser and main module of our system) on the workstation with processor Intel (R), Pentium ® 4 CPU 3.20 GHz , 3.21 GHz, 992 MB of RAM. This test can give us approximately results. As the next work that should be finished the prototype of the application has to be tested on the real environment when the prototype of application is ready to be installed.

	Time
Parse XML	init: deps-jar: compile-single: run-single: Time needed to parse XML is: **7.99** ms BUILD SUCCESSFUL (total time: 1 second)
Serialize graph	init: deps-jar: compile-single: run-single: Time needed to serialize graph is: **3.76** ms BUILD SUCCESSFUL (total time: 0 seconds)
Deserialize graph	init: deps-jar: compile-single: run-single: Time needed to deserialize graph is: **2.35** ms BUILD SUCCESSFUL (total time: 0 seconds)
Initialize user interface window	-----
System's reaction time for user's input (selected icon)	Time of system's reaction (3 prop) is 47.0 ms Time of system's reaction (3 prop) is 31.0 ms Time of system's reaction (6 prop) is 62.0 ms Time of system's reaction (6 prop) is 46.0 ms Time of system's reaction (6 prop) is 47.0 ms Time of system's reaction (9 prop) is 78.0 ms Time of system's reaction (9 prop) is 79.0 ms
	Time of system's reaction is (1 child) 16.0 ms Time of system's reaction is (2 child) 31.0 ms Time of system's reaction is (2 child) 16.0 ms Time of system's reaction is (2 child) 15.0 ms

Figure 22. Implementaion test results

6. User Test

6.1. Icon Usability Test

The most important part of the research has to be done during a project like ours. First of all, before starting the user interface implementation we need to provide the test for the final users in order to understand their ability in recognizing and using the icons to represent their ideas. As we know, different users can understand the same icon in a different way or the same idea can be explained using different icons. This fact comes from the culture background of the user, age, gender, level of education etc. That's why numerous tests have to be performed to find the best solution using icons. [9]

Some example tests can be applied. Icon usability testing can be constructed into two different ways:

- *Icon intuitiveness* test in which an icon is shown to a small number of users without its label. The users are asked to state their best guess as to what the icon is supposed to represent. This test assesses the degree to which the graphic chosen for the icon represented the intended concept.
- Standard *usability test* in which the icons are shown to users as part of the full user interface and where the users are asked to "think aloud" as they use the system to perform set tasks. This test assesses the degree to which the icon would work well in context of the interface as a whole. [14]

We can also ask users to help choosing the set of icons for our system. This can be done as a test: we offer users a few icons (with different graphical style) that represent the same topology and ask them to choose one of them which explain its meaning in the best way.

[9] These kinds of tests are not applied in our application but as a recommendation – it should be performed in projects like this.

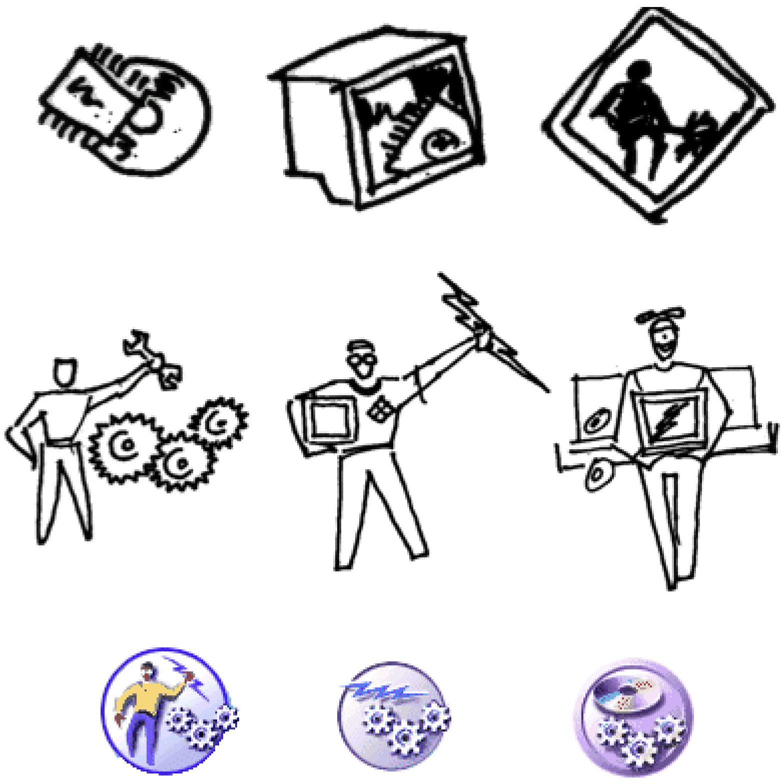


Figure 23. Different icons for Technology and Developer

This kind of tests is very important to be created because it improve the capability of our application, because the main goal of every application is to be useful for users.

7. Conclusions and recommendations

The main goal of this project was to explore the use of an iconic interface on PDAs as a previously created paradigm, to extend it with new ideas, to bring new technologies and contribution to design and implementation approaches. In our project I had shown a few different methods in system's design aspects and one in the implementation as well. We discussed every possible scenario in order to show their place in this project. I hope someone who reads this project report will be inspired by my ideas and that this report will be used for the future researches.

> *"Define key aspects of topic research (1), design a system that is suited for iconic communication for a travelling (2) and implement system prototype with understandable user interface (3). The system should be intelligent enough to assemble and express a correct and up to date world model of traveller."*

1) We investigated the main aspects of the project such as: usage of icons for representing main concepts, usage of icons for software realization. We saw that it provides a good way of the communication not only between humans but between human and program as well. Recommendation: during the design and implementation parts the user test (*Icon Usability Test*) should be performed and icon sets for the representation of dictionaries should be selected very carefully according the test results.

2) The design of our system is complex. It includes: programming design (*Figure 13. Project Architecture*), ontology design (*Figure 8. Ontology model*) and graphical design for a user interface model (*Figure 11. a) User Interface form model b) User Interface form prototype)*.
In this research we described two possible system model from which the more appropriate one was chosen for our system. This task is solved. The ontology design is created for few categories of our system that can cover the prototype's scope.
Recommendation: In fact the ontology model is a very important part of our project, it presents the skeleton for the system's functionality and is the starting point for the implementation process. Due to this reason we have to be very careful and spare a lot of time and attention for the ontology model construction. The user interface design has to be uncomplicated because otherwise it confuses users.
Recommendation: The additional work has to be finished in order to add more functionality to the user interface model to give more opportunities to the user to navigate through the sentences.

3) The implementation part contains enough logical issues to navigate through the data for the creating correct sentences. But the additional testing can raise necessity of corrects and new issues.

Recommendation: Unfortunately a very small part of all described functions our system should have were realized. As future work in the context of this project the prototype can be improved with a new language (in this case the input data has to be prepared according to the implementation plan). Improvement of used icons can be performed as well (one style of all used icons should be kept).

8. References

[1a] Iconic Communication - Iulia Tatomir

[1] Icon Guide - http://wiki.kde.org/tiki-index.php?page=Icon+

[2] Using professional icons for a better communication between user and application interface - http://www.iconshock.com/icons-articles/using-professional-icons-for-a-better-communication-between-user-and-application-interface/

[3] Esperanto - http://en.wikipedia.org/wiki/Esperanto#cite_note-Sikosek_2003-21

[4] Paul Schooneman - "Icon based System for Managing Emergencies"

[5] Siska Fitrianie - An icon-based communication interface on a PDA.

[6] http://users.jyu.fi/~airi/xmlfamily.html#_1._Introduction_1

[7] http://www.sun.com/software/javafx/index.jsp, http://www.sun.com/software/javafx/script/index.jsp

[8] "UML Weekend Crash Course" by Thomas A. Pender

[9] UML: Use Case Diagrams - http://www.alagad.com/go/blog-entry/uml-use-case-diagrams

[10] "An Exploration Of XML in Database Management Systems" by Dare Obasanjo – http://www.25hoursaday.com/StoringAndQueryingXML.html

[11] Parsing XML Efficiently" by Ping Guo, Julie Basu, Mark Scardina, and K. Karun - http://www.oracle.com/technology/oramag/oracle/03-sep/o53devxml.html

[12] SAX vs. DOM - http://www.devx.com/xml/Article/16922/0/page/2

[13] "Java Serialization" by Andrew Downs http://www.mactech.com/articles/mactech/Vol.14/14.04/JavaSerialization/index.html

[14] Icon Usability - http://www.useit.com/papers/sun/icons.html

9. Appendix

9.1. Basic sentence's set

- ✓ **Money**

Bank

- What time does the bank open? (key: money, bank, time, open)
- What time does the bank close? (key: money, bank, time, close)
- I would like to open an account.(key: money, bank, account, open)
- I would like to open a saving account. (key: money, bank, account, save, open)
- I want to withdraw some money. (key: money, bank, money, get)
- I would like to deposit some money. (key: money, bank, money, put)
- I would like to arrange a transfer. (key: money, bank, money, transfer)
- I would like to cash a cheque. (key: money, bank, cash, cheque)
- I would like to get a cash advance. (key: bank, money, advance)
- I would like to cash travelers' cheque.
- What kind of documents I need to do it?

Exchange

- Where can I change money? (key: money,
- I would like to change money. (key: money, change, money)
- Could you give me change for this bill?
- Is there commission?
- What's the exchange rate for <Dutch Guilder>? (key: money, exchange, rate)
- Could you break this <100 dollar> bill?
- Could you include some small change too?
- I would like some change.
- I would like coins of all sizes.

Cash-dispenser

- Where is a cash-dispenser? (key: money, machine, location)

✓ Transport

Train

- Which platform does the train depart from? (key: transport, train, depart, platform, (?))
- What time does this train depart? (key: transport, train, depart, time, (?))
- Where is platform number 1? (key: transport, train, platform, location)
- Is this the train to London? (key: transport, train, direction)
- Is this direct train to <Rotterdam>? (key: transport, train, direct, direction)
- Where is my seat? (key: transport, train, seat, location)
- Is this seat occupied? (key: transport, train, seat, busy)
- Where are we passing now? (key: transport, train, moment, location)
- What is the next stop? (key: transport, bus, stop, next)
- Does it stop at <name of place>? (key: transport, train, stop, place)
- I prefer a seat by the window.(key: transport, train, seat, window)
- How long does the train stop here? (key: transport, train, stop, time)
- I think this is my seat. (key: transport, train, seat, mine)
- Which carriage is for dining? (key: transport, train, carriage, restaurant)
- Which carriage is for 1st class? (key: transport, train, carriage, class, 1)
- Which carriage is for 2st class? (key: transport, train, carriage, class, 2)
- Do I need to change? (key: transport, train, change)
- It is very hot here. May I open the window? (key: transport, train, window, open)
- Where can I see a timetable? (key: transport, train, schedule, location)
- How much is the ticket? (key: transport, train, ticket, price)

- I would like to buy a round-trip (return; two way) ticket to <name place>. (key: transport, train, ticket, buy, return)
- I would like to buy a single ticket to <name of place>. (key: transport, train, ticket, buy, single)
- I would like to get off at <name of place>. (key: transport, bus, stop, leave)
- I would like to buy 2 single tickets to <name of place>. (key: transport, train, ticket, buy, single)
- I want to cancel this ticket. (key: transport, train, ticket, cancel)
- I would like to reserve a seat on this train.
- Could I change seats with you? (key: transport, train, seat, change)
- Which carriage is for <name of place>?
- Does this train go to <name of place>? (key: transport, train, direction, place)
- How long is the delay? (key:

Bus & Tram (the same questions)

- Do I need to change? (key: transport, bus, change)
- I want to cancel this ticket. (key: transport, bus, ticket, cancel)
- Is this a bus stop? (key: transport, bus, stop)
- Where is the nearest bus stop? (key: transport, bus, stop, location)
- Where is a bus station? (key: transport, bus, station, location)
- How often do buses come? (key: transport, bus, schedule)
- Does it stop at <name of place>? (key: transport, bus, stop, place)
- What is the next stop? (key: transport, bus, stop, next)
- Where can I see a timetable? (key: transport, bus, schedule, location)
- I would like to get off at <name of place>. (key: transport, bus, stop, leave)
- I would like to buy a round-trip (return; two way) ticket to <name place>. (key: transport, bus, ticket, buy, return)
- I would like to buy a single ticket to <name of place>. (key: transport, bus, ticket, buy, single)

- Is this seat occupied? (key: transport, bus, seat, busy)
- Could I change seats with you? (key: transport, bus, seat, change)
- Where are we passing now? (key: transport, bus, moment, location)
- I prefer a seat by the window.(key: transport, bus, seat, window)
- Is there any discount? (key: transport, bus, ticket, discount)
- Does this train go to <name of place>? (key: transport, bus, direction, place)

Subway

- Where is the nearest subway station? (key: transport, subway, station, location)
- What line should I take? (key: transport, subway, direction)

Taxi

- I would like a taxi now.
- Where is the taxi rank?
- Is this taxi available?
- Please, take me to <this address>.
- How much is this to <Geleen>?
- Please, stop here.
- Please, wait here.

Plane

- Where is an airport <name of airport>? (key: transport, plane, location)
- Where does flight <number of flight> depart? (key: transport, plane, flight, depart)
- Where does flight <number of flight> arrive? (key: transport, plane, flight, arrive)
- When does boarding begin? (key: transport, plane, boarding, time)
- What gate does this flight leave from? (key: transport, plane, boarding, gate)
- Where is gate 1? (key: travel, plane, gate, location)
- Could I change seats with you? (key: travel, plane, seat, change)
- Where is the baggage check? (key: transport, plane, baggage, check)

- o Where can I get my baggage? / Where is a luggage hall? (key: transport, plane, baggage, location)
- o I can't find my baggage. (key: transport, plane, baggage, lost)
- o Where is the duty-free shop? (key: transport, plane, shop, location)
- o Where is the arrivals hall? (key: transport, plane, arrive, hall, location)
- o Where is a transfer desk? (key: transport, plane, ?, ?

Ferry/Boat

- o Where is the dock? (key: transport, ship, location)
- o Where can I board the ship? (key: transport, ship, board, location)
- o What time do we board? (key: transport, ship, board, time)
- o What time does the ship sail? / When does it sail? (key: transport, ship, departure, time)
- o What time does the ship land? (key: transport, ship, arrival, time)
- o Would you please show me to my cabin? / Please show me to my cabin? (key: transport, ship, room, location)
- o I would like to have breakfast in my cabin. (key: transport, ship, room, breakfast)

✓ Border crossing

Passport

- o I am in transit. (key: travel, border, transit)
- o Here is my passport. (key: travel, border, passport, passport)
- o The children are on this passport. (key: travel, border, passport, children)
- o Would you please stamp my passport? (key: travel, passport, stamp)
- o I am on vacation. (key: travel, passport, reason, vocation)
- o I am on a business trip. (key: travel, passport, reason, business)
- o This is my first visit. (key: travel, passport, visit, 1st)
- o I am going to <name of place>. (key: travel, reason, direction)
- o I plan to stay here <2> weeks. (key: travel, border, stay, time, week)

- I plan to stay here <2> months. (key: travel, border, stay, time, month)
- I plan to stay here <2> months. (key: travel, border, stay, time, day)

Customs

- I only have articles for personal use. (key: travel, custom, goods, private)
- I have nothing to declare. (key: travel, custom, goods, no declare)
- I have something to declare. (key: travel, custom, goods, declare)
- Do I have to declare this? (key: travel, custom)
- Do you have this form in <English>? (key: travel, custom, form, English)
- Give me please another customs form? (key: travel, custom, form, new)
- Please show me how to fill in the form. (key: travel, custom, form, example)
- I am a tourist. (key: travel, border, tourist)

✓ **Hotel**

Reservation

- Where is this hotel? (key: hotel, location)
- I have a reservation (key: hotel, reservation)
- Can I make a reservation (key: hotel, reservation, make)
- When is check-in time? (key: hotel, check-in, time)
- When is check-out time? (key: hotel, check-out, time)
- Do you have any vacant room? (key: hotel, room, free)
- Do you have a single room? (key: hotel, room, single)
- I would like a single room? (key: hotel, room, single, wish)
- Do you have a double room? (key: hotel, room, double)
- I would like a double room? (key: hotel, room, double, wish)
- Do you have a twin room? (key: hotel, room, twin)
- I would like a twin room? (key: hotel, room, twin, wish)
- Do you have room for 4 people? (key: hotel, room, person, 4)

- Is breakfast included? (key: hotel, price, breakfast)
- Can I change room? / I want to change room) (key: hotel, room, change)
- I will take this room for a week (month). (key: hotel, room, use, time)
- Other hotel service: room with extra bed, internet, iron, hair-drier; swimming pool, fitness, bar, spa center; medical service…

Rate

- Do you need a deposit? (key: hotel, deposit)
- What does the price include? (key: room, price, include)
- How much is for room including breakfast? (key: hotel, price, include)
- Is the service charge included? (key: hotel, price, include, service)
- How much is the service charge and tax? (key: hotel, price, service)
- May I have the bill? (key: hotel, bill)

Service

- I would like to put some valuables in the safety-deposit box. (key: hotel,)
- I would like my valuables back. (key: hotel,)
- Please bring me a bath towel. (key: hotel, service, towel)
- Would you give me another blanket? (key: hotel, service, blanket)
- Would you please call a taxi? (key: hotel, service, taxi)

✓ **Shopping**

Information

- Where is a shopping area? (key: shop, area, location)
- What are some specialty items of the town?
- Where can I buy it?
- Where is the nearest bookstore? (key: shop, book, location)
- Is there a department store nearby? (key: shop, food, location)
- Where is a supermarket? (key: shop)
- Where is a grocery shop? (key: shop, food)
- Where is a consumer electronics shop? (key: shop, electronic, location)

- What hours are shops open? (key: shop, time, open)

Price

- How much is this? (key: shop, price)
- Can you write down the price? (key: shop, price, write)
- Do I have to pay? (key: money, pay)
- I want to buy it. (key: shop, money,

Clothes and shoes:

- Do you have (dresses, skirts, women's coats, women's jackets, women's shorts, women's sport suits, women's suits, women's swimsuits, women's scarf, women's belts, socks, women's trousers, tights, women's boots, women's shoes, women's sandals, women's athletic shoes, women's jeans, women's underwear)?
- Do you have (men's coats, men's jackets, men's shorts, men's sport suits, men's suits, men's scarf, ties, men's belt, socks, men's trousers, men's boots, men's shoes, men's athletic shoes, shirt, men's jeans, men's underwear)?
- Do you have this in (red, green, gray, black, pink, white, brown, yellow, violet) color? (key: clothes, color)
- Can I try it on? (key: clothes, try)
- Where is the fitting room? (key: clothes, try, location)
- Do you have smaller size? (key: clothes, size, small)
- Do you have bigger size? (key: clothes, size, big)

Consumer electronics

- I would like to see this (camera, laptop, video camera, mobile phone, printer, scanner, mp3 players, hair-dryer, vacuum-cleaner, iron, washing machine)
- Do you have (cameras, laptops, video cameras, mobile phones, printers, mp3 players, hair-dryers, vacuum-cleaners, irons, washing machine)
- Will you show me how to use it? (key: show, use)

✓ Food and drink

Restaurant

- Are there any good restaurants around here? (key: restaurant, location)

- Where can I find a sandwich shop? (key: fast-food, location)
- Can you recommend a good place to eat? (key: advise, eat, location)
- Where can I eat the best local food? (key: food, local)
- I would like to reserve a table for two (3,4 …). (key: restaurant, table, reserve)
- I would like a table for two.(key: restaurant, table)
- I would like a table in a non-smoking area. (key: restaurant, table, no smoking)
- I would like a table in a smoking area. (key: restaurant, table, smoking)
- I'm ready to order now. (key: restaurant, ordering)
- I'm not ready to order yet. (key: restaurant, no ordering)
- What do you recommend? (key: restaurant, advise)
- May I have the menu and the wine list, please? (key: restaurant, menu)
- Do you serve vegetarian food? (key: restaurant, food, vegetarian)
- What do you have for dessert? (key: restaurant, dessert)
- I would like a (fruit) juice. (key: restaurant, order, drink, juice, (friut))
- I would like a cup of coffee. (key: restaurant, order, drink, coffee)
- I would like a bottle of white wine. (key: restaurant, order, drink, wine, white)
- I would like a bottle of red wine. (key: restaurant, order, drink, wine, red)
- Is it tasty? (key: restaurant, food, tasty)
- This is not my order. (key: restaurant, order, no)
- Could I have the bill, please? (key: restaurant, bill)
- What does this dish include? (key: dish, components)
- I would like this dish without (with) <smth from fruit, vegetables or spaces>

Fruits

- Peach, apricot, pear, cherry, plum, fig, kiwi, orange, lemon, grapefruit, mandarin, mango, pine-apple, coco-nut, strawberry, grapes.

Vegetables

- Cucumber, tomato, pepper, cabbage, onions, potato, carrot, mushroom, broccoli.

Spaces

- Salt, sugar, oil, vinegar, parsley, garlic, chili pepper, mayonnaise, ketchup, chilli, mustard

Drink

- Milk, juice with <smth from fruit>, water, tea, cocoa, coke, coffee.
- Red wine, white wine, beer, whisky, vodka.

Meat

- Chicken, pork, beef, veal, mutton, turkey, lamb, hum, sausage.

Sweet:

- Chocolate, cake, ice-cream, candy, pudding.

✓ **Health**

Doctor

- Where is the nearest doctor? (key: doctor, location)
- I need a doctor. (key: doctor)
- Could the doctor come here? (key: doctor, home)
- I don't feel well. (key: health, feel, bad)
- I have high blood pressure. (key: blood
- I have a cold.
- I have got a sore throat. (key: throat, pain)
- I am sick. (key: person, sick)
- My child is sick. (key: person, child, sick)
- I am allergic to antibiotics. (key: allergic, antibiotics).
- I am allergic to .. (fruits, sweet, flowers, bees). (key: allergic)
- My (parts of the body) hurts.

- I cannot move my (parts of the body). (key: doctor, not move,
- How long will the treatment take? (key: doctor, treatment, time)
- Parts of the body: head, arm, stomach, bum, foot, hand, chest, leg; ear, neck, mouth, nose, eye;

Dentist

- Where is a dentist? (key: doctor, tooth, location)
- I would like to have a tooth filled. (key: tooth, filling, add)
- I have lost a filling. (key: tooth, filling, remove)
- I have a severe toothache. (key: tooth, pain)
- Is it necessary to pull it out? (key: tooth, delete)
- Don't pull it out. (key: tooth, no delete)

Pharmacy

- Where is a pharmacy? (key: pharmacy, location)
- I need something for a headache. (key: pharmacy, headache, pills)
- I need painkillers. (key: pharmacy, pain, pills)
- I need something for period pain. (key: pharmacy, pain, period, pills)
- Can I have something for a cough? (key: pharmacy, cough, pills)
- How often do I take this medicine? (key: pharmacy, medicine, time)

Hospital

- Where is the nearest hospital? (key: hospital, location)
- Call an ambulance please. (key: ambulance)
- How long will it take to recover? (key: hospital, recover, time)
- My blood type is A (B, AB, 0) positive (negative). (key: hospital, blood, type)
- Do I have to stay in the hospital? (key: hospital, stay)
- Would you like to inform my family, please. (key: hospital, family, inform)

✓ Sport

Game

- Soccer/football
- Volley-ball
- Tennis/Table tennis
- Cycling
- Ice-skating
- Caving
- Hiking

Playing sport

- Do you want to play in?
- Can I join it?
- You are a good player.
- Thanks for the game.
- Congratulations on the victory.
- My/Your point.
- Pass it to me.
- Where is the good place to fish?
- Where is the good place to run?
- Where is the good place to surf?
- Where is the nearest golf course?
- Where is the nearest swimming pool?
- Where is the nearest gym?
- Do I have to be a member to attend?
- Where is the changing room?

✓ **Communications**

Internet

Post Office

Mobile/Cell Telephone

- ✓ **Where to go (rest)**

Walk

At night

- o What is there to do in the evening? (key:)
- o Where can I find night clubs? (key:)
- o Where can I find pubs? (key:)
- o Is there a local entertainment guide? (key:)

- ✓ **Society**

Feelings

- o <I> am (not) disappointed. (key: society, feelings, (-), <person>, mood, dissapoint)
- o Are <you> dissapointed? (key: society, feeling, (?), <person>, mood, dissapoint)
- o <I> am (not) embarrassed. (key: society, feelings, (-), <person>, mood, embarrass)
- o Are <you> embarrassed? (key: society, feeling, (?), <person>, mood, embarrassed)
- o <I> am (not) happy. (key: society, feelings, (-), <person>, mood, happy)
- o Are <you> happy? (key: society, feeling, (?), <person>, mood, happy)
- o <I> am (not) surprised. (key: society, feelings, (-), <person>, mood, surprise)
- o Are <you> surprised? (key: society, feelings, (?), <person>, mood, surprise)
- o <I> am (not) sad. (key: society, feelings, (-), <person>, mood, sad)
- o Are <you> sad (key: society, feeling, (?), <person>, mood, sad)
- o <I> am (not) angry. (key: society, feeling, (-), <person>, mood, angry)
- o Are <you> angry. (key: society, feeling, (?), <person>, mood, angry)
- o <I> am (not) tired. (key: society, feelings, (-), <person>, body, tired)
- o Are <you> tired? (key: society, feeling, (?), <person>, body, tired)

- <I> am (not) worried. (key: society, feelings, (-), <person>, mood, worry)
- Are <you> worried? (key: society, feelings, (?), <person>, mood, worry)
- <I> am (not) right. (key: society, feeling, (-), <person>, mood, right)
- Am <I> right? (key: society, feeling, (?), <person>, mood, right)
- <I> am (not) wrong. (key: society, feeling, (-), <person>, mood, wrong)
- Am <I> wrong? (key: society, feeling, (?), <person>, mood, wrong)
- <I> am (not) hot. (key: society, feeling, (-), <person>, body, hot)
- Are <you> hot? (key: society, feeling, (?), <person>, body, hot)
- <I> am (not) cold. (key: society, feeling, (-), <person>, body, cold)
- Are <you> cold? (key: society, feeling, (?), <person>, body, cold)
- <I> am (not) hungry. (key: society, feeling, (-), <person>, food, miss (not))
- Are <you> hungry. (key: society, feeling, (?), <person>, hungry)
- <I> am (not) thirsty. (key: society, feeling, (-), <person>, thirsty)
- Are <you> thirsty. (key: society, feeling, <person>, drink, miss, question)
- <I> am sleepy. (key: society, feeling, <person>, sleep)

Opinions

- Did <you> like it? (key: society, opinion, <person>, like, it, question)
- I like it. (key: society, opinion, <person>, like, it)
- What do <you> think about it? (key: society, opinion, <person>, think, it, question)
- I thought it was / It is awful. (key: society, opinion, impression)
- I thought it was / It is beautiful. (key: society, opinion)
- I thought it was / It is boring. (key: society, opinion)
- I thought it was / It is great. (key: society, opinion)
- I thought it was / It is interesting. (key: society, opinion)
- I thought it was / It is strange. (key: society, opinion)

- Do <you> agree with <me>? (key: society, opinion, (?), agree, <person>)
- <I> (don’t) agree with <you>. (key: society, opinion, (-),<person>, agree, <person>)

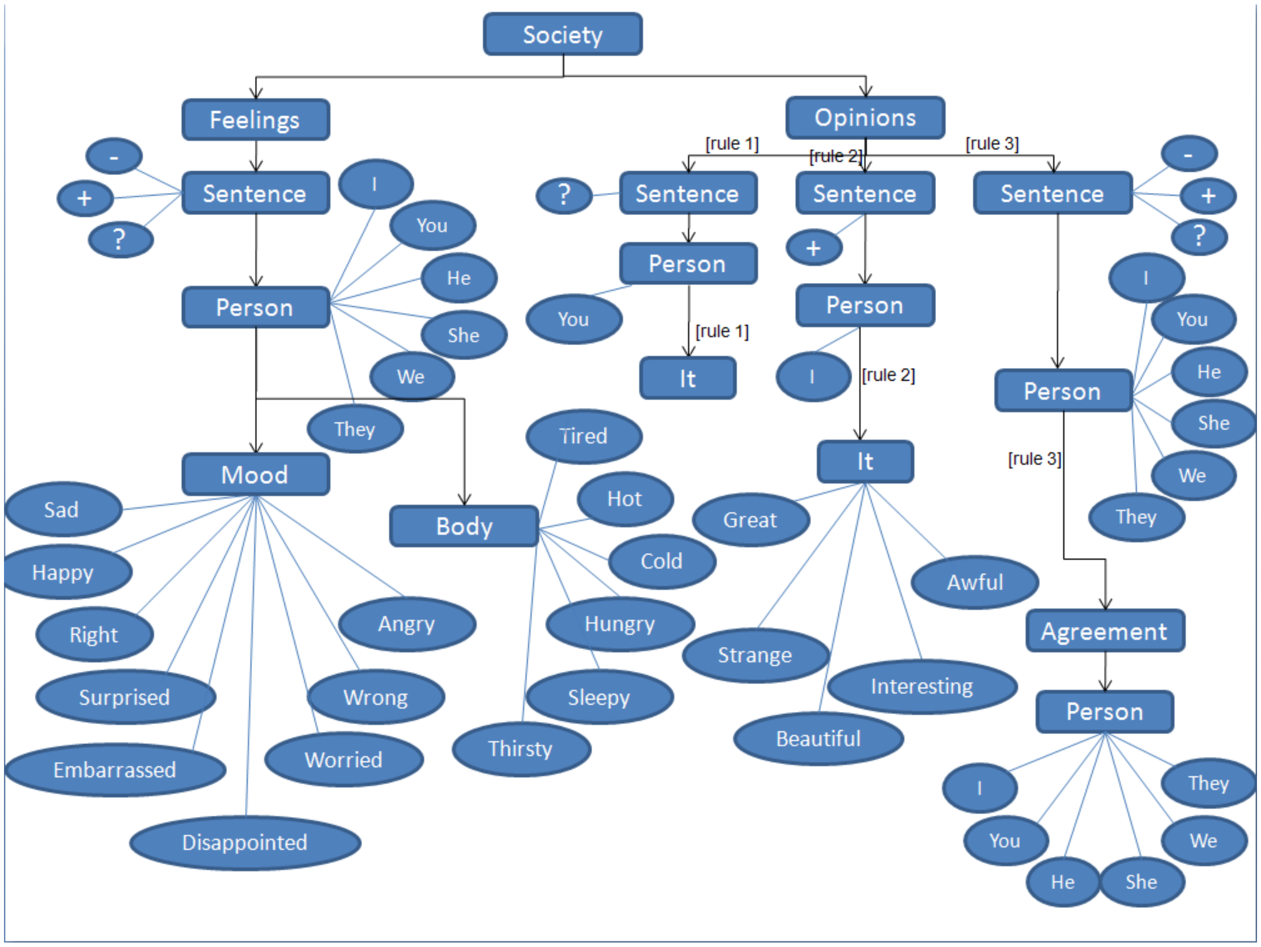
Invitations

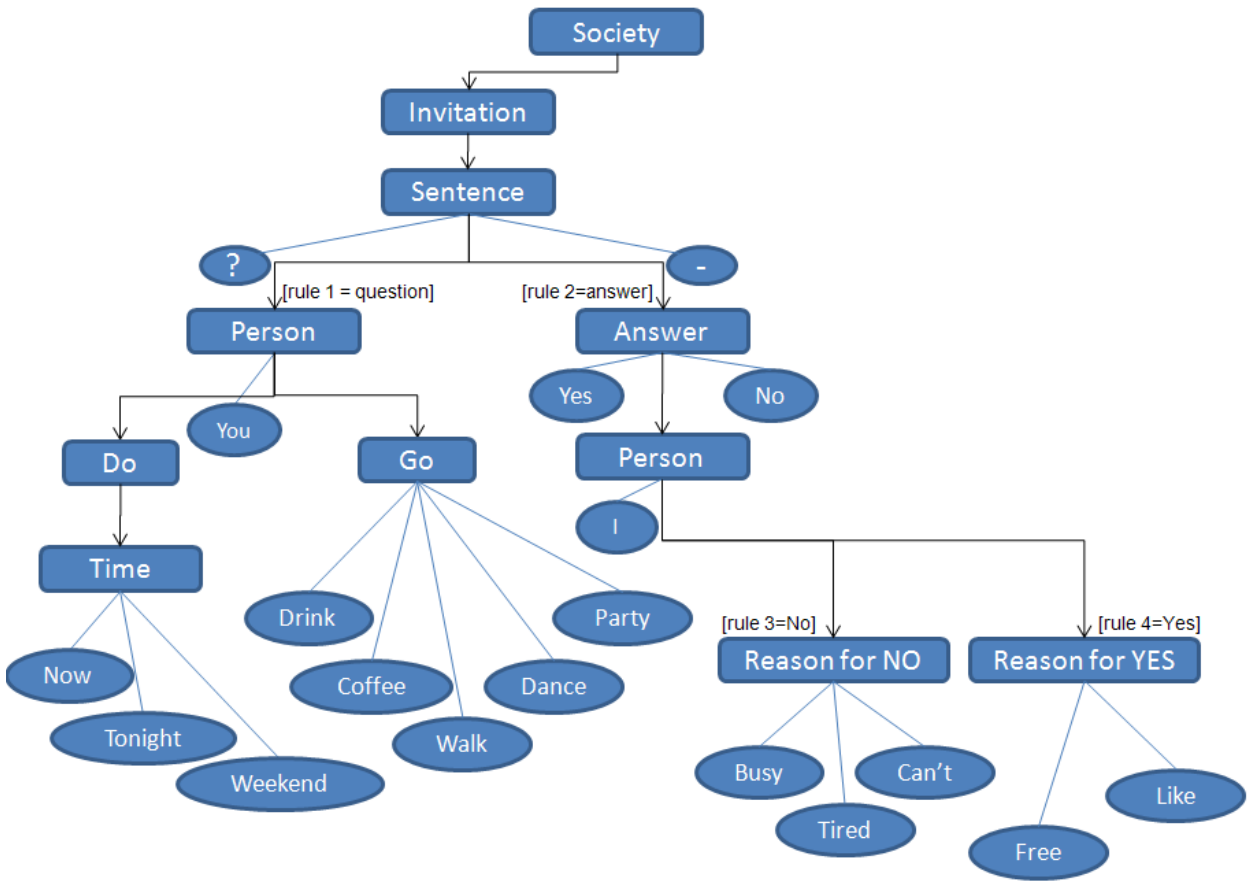
- What are you doing now? (key: society, invitation, person, do, moment)
- What are you doing this evening? (key: society, invitation, person, do, evening)
- What are you doing this weekend? (key: society, invitation, person, do, weekend)
- Would you like to go for a drink? (key: society, invitation, question)
- Would you like to go for a beer? (key: society, invitation, question)
- Would you like to go for a coffee? (key: society, invitation, question)
- Would you like to go for a walk? (key: society, invitation, question)
- Would you like to go dancing? (key: society, invitation, question)
- We are having party. You should come. (key: society, invitation, party, come)
- Sure! I would love to. (key: society, invitation, answer, yes)
- Yes, I am free. (key: society, invitation, answer, yes, free)
- No, I am afraid I cannot. (key: society, invitation, answer, no, can’t)
- No, I am afraid I am busy. (key: society, invitation, answer, no, busy)
- No, I am afraid I am busy. (key: society, invitation, answer, no, tired)

✓ Location

- Where is this street?
- How can I find this place?
- Where is situated this building?

9.2. Graph models





9.3. XML representation

XML file for category “Feelings”

```
<root>
       <node name="Feelings" value="" grammerType="noun"
imgPath="images/feelings/feelings.png"
               visible="true">
           <children>
               <node name="Sentence" value="" grammerType="noun" rule=""
                           imgPath="images/feelings/sentance.png"
visible="true">
                   <properties>
                       <property name="+" value="+"
                                       grammerType="sentence_positive"
imgPath="images/feelings/positive.png" visible="true">
                       </property>
                       <property name="-" value="not"
                                       grammerType="sentence_negative"
imgPath="images/feelings/negative.png" visible="true">
                       </property>
                       <property name="?" value="?"
```

```
                                       grammerType="sentence_question"
imgPath="images/feelings/question.png" visible="true">
                       </property>
                   </properties>
                   <children>
                       <node name="Person" value="" grammerType="noun"
                                       imgPath="images/feelings/person.png"
visible="true">
                           <properties>
                               <property name="I" value="I"
                                                   grammerType="prononun"
imgPath="images/feelings/me.png" visible="true" addForm="am">
                               </property>
                               <property name="You" value="You"
                                                   grammerType="prononun"
imgPath="images/feelings/you.png" visible="true"
                                                   addForm="are">
                               </property>
                               <property name="He" value="He"
                                                   grammerType="prononun"
imgPath="images/feelings/he.png" visible="true" addForm="is">
                               </property>
                               <property name="She" value="She"
                                                   grammerType="prononun"
imgPath="images/feelings/she.png" visible="true" addForm="is">
                               </property>
                               <property name="We" value="We"
                                                   grammerType="prononun"
imgPath="images/feelings/we.png" visible="true"
                                                   addForm="are">
                               </property>
                               <property name="They" value="They"
                                                   grammerType="prononun"
imgPath="images/feelings/they.png" visible="true"
                                                   addForm="are">
                               </property>
                           </properties>
                           <children>
                               <node name="Mood" value=""
                                                   grammerType="noun"
imgPath="images/mood/mood.png" visible="false">
                                   <properties>
                                       <property name="Sad" value="sad"

   grammerType="adjective" imgPath="images/mood/sad.gif" visible="true">
                                       </property>
                                       <property name="Happy" value="happy"

   grammerType="adjective" imgPath="images/mood/happy.gif" visible="true">
                                       </property>
                                       <property name="Disappointed"

   value="disappointed" grammerType="adjective"
imgPath="images/mood/disappointed.gif"

   visible="true">
                                       </property>
                                       <property name="Right" value="right"
```

```
grammerType="adjective" imgPath="images/mood/right.gif" visible="true">
                    </property>
                    <property name="Angry" value="angry"
grammerType="adjective" imgPath="images/mood/angry.gif" visible="true">
                    </property>
                    <property name="Wrong" value="wrong"
grammerType="adjective" imgPath="images/mood/wrong.gif" visible="true">
                    </property>
                    <property name="Embarrassed"
value="embarrassed" grammerType="adjective" imgPath="images/mood/confused.gif"
visible="true">
                    </property>
                    <property name="Surprised"
value="surprised" grammerType="adjective" imgPath="images/mood/surprised.gif"
visible="true">
                    </property>
                    <property name="Worried"
value="worried" grammerType="adjective" imgPath="images/mood/worried.gif"
visible="true">
                    </property>
                </properties>
            </node>
            <node name="Body" value=""
                        grammerType="noun" imgPath="images/mood/body.png" visible="true">
                <properties>
                    <property name="Hungry"
value="hungry" grammerType="adjective" imgPath="images/mood/hangry.gif"
visible="true">
                    </property>
                    <property name="Sleepy"
value="sleepy" grammerType="adjective" imgPath="images/mood/sleepy.gif"
visible="true">
                    </property>
                    <property name="Cold" value="cold"
grammerType="adjective" imgPath="images/mood/cold.gif" visible="true">
                    </property>
                    <property name="Hot" value="hot"
grammerType="adjective" imgPath="images/mood/hot.gif" visible="true">
</property>
```

```
                                    <property name="Thirsty"
    value="thirsty" grammerType="adjective"
   imgPath="images/mood/thirsty.gif"
    visible="true">
                                    </property>
                                    <property name="Tired" value="tired"
    grammerType="adjective" imgPath="images/mood/tired.gif" visible="true">
                                    </property>
                                </properties>
                            </node>
                        </children>
                    </node>
                </children>
            </node>
        </children>
    </node>
</root>
```

121.1. Program code

Class for graph representation

```java
package graph;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

import org.w3c.dom.DOMException;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;

/**
 * @author jenka
 *
 */
public class GraphNode implements Serializable {

    private String name;
    private String value;
    private String imgPath;
    private String grammerType;
    private boolean visible;
    private List<Rule> rules;
    private List<GraphNode> children;
    private GraphNode parentNode;
    private LinkedList<Property> properties;
    private boolean selected;

    /**
     *
     * @param name
     * @param value
     * @param grammerType
     * @param imgPath
     */
    public GraphNode(String name, String value, String grammerType,
            String imgPath) {
        this.name = name;
        this.value = value;
        this.grammerType = grammerType;
        this.imgPath = imgPath;
        //System.out.println("~Create node '" + name + "'~");
    }

    /**
     *
     * @param name
     * @param value
     * @param grammerType
     * @param visible
     */
    public GraphNode(String name, String value, String grammerType,
            boolean visible) {
        this.name = name;
        this.value = value;
        this.grammerType = grammerType;
        this.visible = visible;
    }

    /**
     * Create new node for graph
     *
     * @param name
     * @param value
     * @param grammerType
     * @param imgPath -
     *            icon representation for this node, if it is visible
     * @return
     */
    public GraphNode createChild(String name, String value, String
grammerType,
            String imgPath) {
        //System.out.println("~Create node '" + name + "'~");
        return new GraphNode(name, value, grammerType, imgPath);
    }

    /**
     * Create new node for graph
     *
     * @param name
     * @param value
     * @param grammerType
     * @param visible -
     *            false, if this node is invisible and has only semantic
meaning
     * @return
     */
    public GraphNode createChild(String name, String value, String
grammerType,
            boolean visible) {
        //System.out.println("~Create node '" + name + "'~");
        return new GraphNode(name, value, grammerType, visible);
    }

    /**
     * Add new element to existing graph
     *
     * @param newChild
     * @return
     */
    public GraphNode appendChild(GraphNode newChild) {
        // TODO Auto-generated method stub
        if (this.children == null) {
            this.children = new ArrayList<GraphNode>();
        }
        this.children.add(newChild);
        //System.out.println("~~Append child '" + newChild.getName() + "' to
'" + this.name + "'~~");
        return this;
    }

    /**
     * Add new property to property's list for current node
     *
     * @param newProp
     * @return
     */
    public GraphNode appendProperty(Property newProp) {
        //System.out.println("<-Append property '" + newProp.getValue() + "'
to node '" + this.name + "'.->");
        if (this.properties == null) {
            this.properties = new LinkedList<Property>();
        }
        this.properties.add(newProp);
        return this;
    }

    /**
     * Add new rule to the list with rules for current node
     *
     * @param newRule
     * @return
     */
    public GraphNode appendRule(Rule newRule) {
        //System.out.println("#-Append rule to node '" + this.name + "'.-#");
        if (this.rules == null) {
            this.rules = new LinkedList<Rule>();
        }
        this.rules.add(newRule);
        return this;
    }

    /**
     *
     * @param parentNode
```

```
 */
public void setParentNode(GraphNode parentNode) {
    this.parentNode = parentNode;
}

/**
 * Returns parent node for current element (node) of graph
 *
 * @return
 */
public GraphNode getParentNode() {
    return this.parentNode;
}

/**
 *
 * @return
 */
public boolean isVisible() {
    return this.visible;
}

/**
 *
 * @param value
 */
public void setValue(String value) {
    this.value = value;
}

/**
 *
 * @return
 */
public String getValue() {
    return this.value;
}

public NamedNodeMap getAttributes() {
    // TODO Auto-generated method stub
    return null;
}

/**
 * Returns list with all children nodes for current node
 *
 * @return
 */
public List<GraphNode> getChildNodes() {
    if (children != null) {
        return this.children;
    } else {
        return null;
    }
}

/**
 *
```

```
     * @return
     */
    public GraphNode getFirstChild() {
        if (hasChildNodes()) {
            return children.get(0);
        } else {
            return null;
        }
    }

    /**
     *
     * @return
     */
    public GraphNode getLastChild() {
        if (hasChildNodes()) {
            return children.get(children.size() - 1);
        } else {
            return null;
        }
    }

    public short getNodeType() {
        // TODO Auto-generated method stub
        return 0;
    }

    /**
     * Return grammaer value of this node
     * @return
     * @throws org.w3c.dom.DOMException
     */
    public String getNodeValue() throws DOMException {
        return this.value;
    }

    /**
     * This method returns true if this node has list with properties and
false - if not
     * @return
     */
    public boolean hasProperties() {
        if (this.properties == null) {
            return false;
        } else {
            return true;
        }
    }

    public List<Property> getProperties() {
        return this.properties;
    }

    /**
     * Verifies if current node has children or not
     * @return true - if there are children, false - if not
     */
    public boolean hasChildNodes() {
        if (this.children != null) {
            return true;
        } else {
            return false;
        }
    }

    public boolean isSameNode(Node other) {
        // TODO Auto-generated method stub
        return false;
    }

    public Node removeChild(Node oldChild) throws DOMException {
        // TODO Auto-generated method stub
        return null;
    }

    public List<Rule> getRules() {
        return rules;
    }

    public String getName() {
        return name;
    }

    public String getImgPath() {
        return imgPath;
    }

    /**
     *
     * @return
     */
    public boolean isSelected() {
        return selected;
    }

    /**
     *
     * @param sel
     */
    public void setSelected(boolean sel) {
        selected = sel;
    }

    /**
     * Returns number of properties, or 0-if there is no properties
     * @return - size of 'properties' attribute
     */
    public int getProperNumber() {
        if (hasProperties()) {
            return properties.size();
        }
        return 0;
    }

    /**
     * Returns number of children, or 0-if there is no children
     * @return
     */
    public int getChildNumber() {
        if (hasChildNodes()) {
            return children.size();
        } else {
            return 0;
        }
    }
}
```

SAX parser implementation.

```
package parser;

import java.util.HashMap;
import java.util.LinkedList;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import constants.XMLTagNames;
import graph.*;

/**
 * @author jenka
 *
 */
public class SaxGraphParser extends DefaultHandler {

      private LinkedList<GraphNode> stack;

      private GraphNode tmpNode;// temporary keep node of the graph
      private Property tmpProp;// temporary keeps property for current node

      private String tagName;// name of current tag
      private StringBuffer ruleContent;
      private HashMap<String, GraphNode> extraMap;
      private String extraID;

      private boolean extras = false;
      private GraphNode root;
      private int rr = 0;

      public SaxGraphParser() {
            // TODO Auto-generated constructor stub
      }

      @Override
      public void startDocument() throws SAXException {
            // TODO Auto-generated method stub
            this.stack = new LinkedList<GraphNode>();
            super.startDocument();
      }

      @Override
      public void startElement(String uri, String localName, String name,
                  Attributes attributes) throws SAXException {
            this.tagName = name;
            //System.out.println("Start element '" + name + "'.");
            if (name.equalsIgnoreCase(XMLTagNames.TAG_NAME_NODE)) {
                  rr++;
                  this.tmpNode = setNodeAttr(attributes);
                  if (!this.stack.isEmpty()) {

      this.tmpNode.setParentNode(this.stack.getFirst().appendChild(
                                    tmpNode));
                  }
                  if (!extras && rr == 1) {
                        this.root = tmpNode;
                  }
                  this.stack.addFirst(tmpNode);
            } else if (XMLTagNames.TAG_NAME_CHILDREN.equalsIgnoreCase(name)) {
            } else if (XMLTagNames.TAG_NAME_PROPERTY.equalsIgnoreCase(name)) {
                  this.tmpProp = setPropAttr(attributes);
            } else if (XMLTagNames.TAG_NAME_EXTRAS.equalsIgnoreCase(name)) {
                  this.extras = true;
                  this.extraMap = new HashMap<String, GraphNode>();
            } else if (XMLTagNames.TAG_NAME_EXTRA.equalsIgnoreCase(name)) {
                  this.extraID =
attributes.getValue(XMLTagNames.PROPERTY_NAME_ID);
                  //System.out.println("extra id= " + this.extraID);
            } else if (XMLTagNames.TAG_NAME_RULE.equalsIgnoreCase(name)) {
                  this.ruleContent = new StringBuffer();
            }
            // super.startElement(uri, localName, name, attributes);
      }

      @Override
      public void endElement(String uri, String localName, String name)
                  throws SAXException {
            //System.out.println("End element '" + name + "'.");
            if (XMLTagNames.TAG_NAME_NODE.equalsIgnoreCase(name)) {
                  if (extras)
                        this.extraMap.put(this.extraID,
this.stack.getFirst());
                  this.stack.removeFirst();
                  this.tmpNode = null;
            } else if (XMLTagNames.TAG_NAME_PROPERTY.equalsIgnoreCase(name)) {
                  this.stack.getFirst().appendProperty(tmpProp);
                  this.tmpProp = null;
            } else if (XMLTagNames.TAG_NAME_RULE.equalsIgnoreCase(name)) {
                  this.stack.getFirst().appendRule(new
Rule(ruleContent.toString()));
                  this.ruleContent.substring(ruleContent.indexOf("="));
            } else if (XMLTagNames.TAG_NAME_EXTRA.equalsIgnoreCase(name)) {
                  this.extraID = null;
            } else if (XMLTagNames.TAG_NAME_EXTRAS.equalsIgnoreCase(name)) {
                  this.extras = false;
                  this.rr = 0;
            } else if (XMLTagNames.TAG_NAME_CHILDREN.equalsIgnoreCase(name)) {
            } else if (XMLTagNames.TAG_NAME_PROPERTIES.equalsIgnoreCase(name))
{
            } else if (XMLTagNames.TAG_NAME_RULES.equalsIgnoreCase(name)) {
```

```
			GraphNode temp = this.stack.getFirst();
			int lenght = temp.getRules().size();
			//System.out.println("EXTRAS MAP SIZE=" + extraMap.size()
					//+ ". RULES SIZE=" + lenght + ".");
			String id;
			for (int i = 0; i < lenght; i++) {
				id = temp.getRules().get(i).getID();
				this.stack.getFirst().appendChild(extraMap.get(id));
			}
		}
	}

	@Override
	public void characters(char[] ch, int start, int length)
			throws SAXException {
		if (XMLTagNames.TAG_NAME_RULE.equalsIgnoreCase(tagName)) {
			this.ruleContent.append(ch, start, length);
		}
	}

	/**
	 * Creates GraphNode object with set attributes, read from xml
	 *
	 * @param attributes
	 * @return
	 */
	private GraphNode setNodeAttr(Attributes attributes) {
		return new GraphNode(attributes
				.getValue(XMLTagNames.PROPERTY_NAME_NAME), attributes
				.getValue(XMLTagNames.PROPERTY_NAME_VALUE), attributes
				.getValue(XMLTagNames.PROPERTY_NAME_GRAMMER_TYPE),
attributes
				.getValue(XMLTagNames.PROPERTY_NAME_IMAGE_PATH));
	}

	/**
	 * Creates new Property object with all attributes, read from xml
	 *
	 * @param attributes
	 * @return
	 */
	private Property setPropAttr(Attributes attributes) {
		return new Property(
				attributes.getValue(XMLTagNames.PROPERTY_NAME_NAME),
attributes

	.getValue(XMLTagNames.PROPERTY_NAME_VALUE), attributes

	.getValue(XMLTagNames.PROPERTY_NAME_GRAMMER_TYPE),

	attributes.getValue(XMLTagNames.PROPERTY_NAME_IMAGE_PATH));
	}

	/**
	 * Return the root node of built graph
	 *
	 * @return
	 */
	public GraphNode getRoot() {
		return this.root;
	}
}
```