Model based development of speech recognition grammar for VoiceXML

Jaspreet Singh

University of Tampere
School of Information Sciences
Computer Science
M.Sc Thesis
Supervisor: Zheying Zhang
December 2011

University of Tampere

School of Information Sciences

Computer Science

Jaspreet Singh: Model based development of speech recognition grammar for VoiceXML.

M.Sc. Thesis 53 pages, 4 index and 3 appendix pages

December 2011

Abstract:

Speech interaction is a natural form of interaction between human and devices. This interaction technology is currently in high demand but often has limitations. A limited form of interaction is thus in use to achieve best possible efficiency. The limited form of speech interaction uses direct commands instead of complete natural language. The VoiceXML is a W3C (World wide consortium) recommended web based speech interaction application development language that performs the dialogue management. It has been used as a base language in this thesis for the case study.

The VoiceXML uses a grammatical base to recognise the user utterance for the commands. This thesis applies the model based development approach to create a hierarchical data model for the speech grammar. Further, the grammar has been separated from the interaction code. MetaEdit+ tool has been used for developing the grammar model and for generating the grammar file.

The approach is further compared with other grammar models. In conclusion, the applied approach is found suitable for the grammar modelling in VoiceXML application development.

Key words: Data Model, Grammar Model, MetaEdit+ , VoiceXML, Grammar generation, Hierarchical Model

Contents:

List of abbreviations

HDI	Human Device Interface
VoiceXML	Voice Extensible Mark-up Language
GSL	Nuance Grammar Specification Language
NLP	Natural Language Processing
FSA	Finite State Automation
FSM	Finite State Machine
HMM	Hidden Markov Modelling
SDS	Spoken dialogue system
WWW	World Wide Web
W3C	World Wide Web Consortium
DTMF	Dual-Tone Multi-Frequency
ASR	Automatic Speech Recogniser
TTS	Text to Speech
SSML	Speech Synthesis Mark-up Language
PML	Phone Mark-up Language
FSM	Finite State Machine
URI	Uniform Resource Identifier
JSGF	Java Speech Grammar Format
NGO	Nuance Grammar Object
BNF	Backus-Naur Form
ABNF	Augmented Backus-Naur Form
DBMS	Data Base Management System
IMS	Information Management System
ERM	Entity Relationship Model
UML	Unified Modelling Language
MOF	Meta-Object Facility
CWM	Common Warehouse Meta-model
IR	Intermediate Representation
GOPRR	Graph Object Property Role Relationship
IVR	Interactive Voice Response
GUI	Graphical User Interface
API	Application Programming Interface

MVC	Model View Control
IVE	Interactive Virtual Environments
OWL	W3C Web Ontology Language

1. Introduction

Most of the interactions with electronic devices are done using conventional methods such as a keyboard or a pointing device. In addition, there are various other forms of interaction, with speech interaction being one of them. As speech interaction is a natural form of interaction for human to human communication, it is an effective option for improvising the way of interaction in HDI (Human-device interface). Although there are many ongoing researches to achieve this, supports of natural language interaction are still limited [Shneiderman, 2000] . Speech interaction is often combined with other modes of interaction. The combined form of interaction mechanism is called ”Multi-model Interaction”.

Multi-model interactions use more than one mode of communication between the user and the device. These interactions contain speech, vision, touch, cognitive or a combination of different modes of communication. Further, multi-model interaction is utilised in VoiceXML (Voice Extensible Markup Language) for the development of dialogue based speech interaction applications. Interaction modes in VoiceXML contain the capability to recognise speech inputs as well as DTMF (Dual-Tone multi-frequency) inputs. Moreover, VoiceXML responds in speech based output.

The development of VoiceXML uses a web-based architecture in the form of XML (Extensible Mark-up Language). Different types of speech grammar formats can be used in the VoiceXML applications to match the recognised speech from the user. VoiceXML includes grammar definition in the interaction code that match the user inputs. It may have different speech grammars for individual blocks of code called “dialogs”.

In conventional VoiceXML development, each dialog either has its own grammar or shared grammar to match the user input. A VoiceXML grammar contains the data in the form of words/sentences to be matched with the user utterance. The VoiceXML development structure is based on inter-dialog control flow [VXML2,2001].

In large scale projects, VoiceXML code requires multiple inter-related grammars. Consequently, the usage of multiple unconnected inter-related grammars leads to a complex application development structure. Despite having inter-dialog control flow, VoiceXML may also

have inter-grammar control flow [VXML2,2001]. This inter-grammar control flow can be represented in a separate view based on the grammar. In the thesis, this separation of grammar and interaction code is used. It further simplifies the application development with an option of direct manipulation of grammar without interfering with the interaction code i.e. based on inter-dialog flow.

The aim of this thesis is to propose an alternative approach for VoiceXML development by considering grammar control flow in-parallel to the dialog control flow. A grammar model was created as a medium to describe the grammar control flow and grammatical data in a well structured form. The grammar control flow is represented in a separate GSL (Nuance Grammar Specification Language)[Nuance,2002] file, generated by traversing the grammar model using the MetaEdit+ tool.

After this introduction in chapter 2 I have provided background information on the speech interaction domain. In chapter 3 information on the model based code generation has been provided. Further, in chapter 4 the inter-grammar relationships by using the newly developed grammar model in parallel to the interaction code/model is investigated. In chapter 5 this approach is compared with the related researches. Finally, the conclusion has been provided in chapter 6.

2. Speech Interaction

2.1 Introduction

Speech interaction is based on NLP (natural language processing). Recognising speech is the primary work of an NLP system. The speech recognition history begins from 1920's. In 1920 the first speech recognition machine was commercially developed as a toy named 'Radio Rex' [Windmann and Haeb-Umbach, 2009]. Further, the conceptual research on speech technology began in 1936 at Bell labs.

Early research products were based on the vowel recognition, and then on the word recognition algorithms [Furui, 2005]. The Carnegie mellon university later developed the Harpy system by using a graph search based algorithm, which uses a FSA (finite state automation) network by reducing the computations [Lowrre, 1990]. Further, the need of sentence based recognition over the word recognition arrived in around 1980's for the advancement of technology. Later, template based recognition and statistical methods were introduced. In statistical methods, HMM (Hidden Markov Modelling) was the most popular one in laboratories [Su and Lee, 1994].

A SDS (Spoken dialogue system) architecture may contain various NLP systems. Each of these can work as a separate processing module. Further, Figure 1 describes a simple pipeline structure of SDS and are elaborated in upcoming sub-sections.

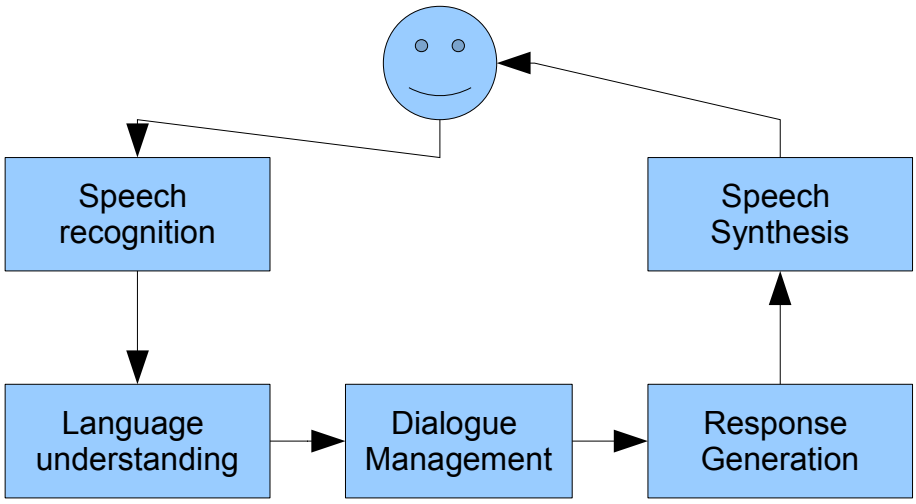

Figure 1. A simple pipeline schema based SDS system

2.1.1 Speech recognition

A speech recognition system transforms a human form of speech into the machine form. The human form of speech is usually referred to natural language. The effective NLP or signal processing is the primary task of a speech recognition system. However, it may perform other tasks such as speaker recognition, noise reduction, morphing etc.

Anusuya and Katti [2009] describe the speech recognition process in a model using mathematical symbols to represent different processes involved, as shown in Figure 2. A probability is to be assumed in the basic speech model by assuming a specific word sequence, W, producing an acoustic observation sequence, A, by probability P(W,A). The goal of the model is then to decode the word string, based on acoustic observation sequence. Acoustic front-end decodes the signal in a digital form for the acoustic model. The acoustic models analyse the digital signal based on the prior word expectations using P(A/W). The system presented in the Figure 1 is based on noisy channel model [Kernighan et al., 1990]. In which, language model is used for the accuracy improvement. However, the search module decodes the combined output probabilities or N-best lists for the used of language understanding module.

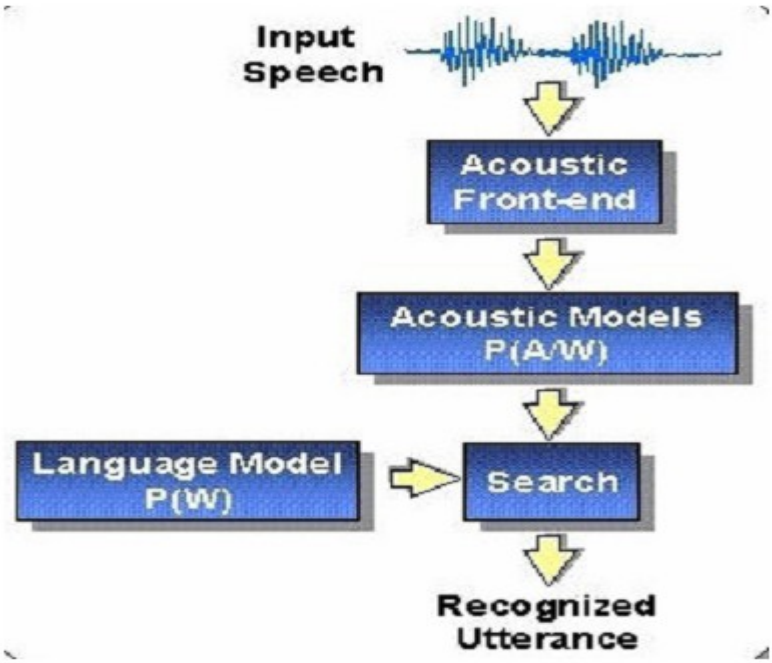

Figure 2. A basic speech recognition model [Anusuya and Katti,2009]

2.1.2 Language understanding

The language understanding module receives speech recognition hypothesis and extracts the meaning. It may contain semantic parser to do the job. The parsing goal is to generate a representation of meanings based on the given grammatical structure. Some usual approaches used for parsing are shallow parsing, grammar-based parsing and stochastic parsing. However, the VoiceXML uses shallow parsing.

In the shallow parsing approach, the parser look for the desired keywords or phrases in the utterances and ignores the rest. These keywords need to be defined beforehand to be matched with the output of speech recognition outputs. This approach is simple and efficient, but it is domain-specific. Figure 3 shows an example of it using flight booking system domain.

….....
….....
on <month> <day>
at <hour>
tomorrow
day after tomorrow
I need to go at <hour> tomorrow
….....
….....

Figure 3. A flight booking system scenario.

In Figure 3, the tagged words represents the place holders for the attributes of temporal types. However, the ignored utterances are shown in Italics. Moreover, a language understanding module also needs to deal with the speech recognition errors, dis-fluencies, reference resolution etc.

2.1.3 Dialogue management

The dialogue management module is responsible for the process control in the SDS. It performs decision making based on language understanding outputs using prior knowledge. It also keeps track of the current dialogue state. It can use different approaches for the task of decision making such as FSA, frame-based approach and optimisation based approach. However, VoiceXML utilizes the frame-based approach.

The frame-based approach consists of multiple slots for holding the related information in them. It is suitable for the shallow parsing that picks relevant information out of sentences. Further, this information fills the slots. Table 1 shows a simple example based on flight booking system. The slots are the required keywords out of sentences and prompts are the questions asked by the system. It is also called “dialogue policy”.

Slot	Prompt
<From>	“where from do you want to fly?”
<To>	“where you want to go?”
<Departure>	“what should be the departure time?”
<Arrival>	“when do you want to arrive?”

Table 1. An example of flight booking system based on frame-based approach.

2.1.4 Response generation

The response generation is the process to generate best possible response representations. The response selection may contain a complex decision making process but VoiceXML utilizes shallow generation. This approach is the reciprocal of dialogue understanding. It maps commands to the prompts. Table 2 displays the command to prompt mappings. The prompts are already stored in the system and the process need to choose the desired.

Command	Response
Confirm(last)	“Did you say Paris?”
Ask(departure)	“From which city you want to fly?”
Greeting(end)	Good bye!

Table 2. An example flight booking system using shallow generation.

2.1.5 Speech synthesis

The speech synthesis is also called TTS system. The process maps the response representations from the response generation module to the actual speech output. It contains two process: text analysis and waveform generation.

The text analysis normalises the text input to phonemic representation. It may add tags to the text for the phonetics and prosodic analysis. Further, the phonetic analysis component processes the text into a phonetic sequence for actual sound. The prosodic component embeds the information such as pitch, duration to the phonetic sequence. However, the waveform generation may perform concatenation or articulatory synthesis. In the concatenation process, it uses pre-recorded speech chunks and joins them to generate the response. On the other hand, the articulatory synthesis uses acoustic models of the vocal tract. Figure 4 displays the speech synthesis process in a pipeline fashion.

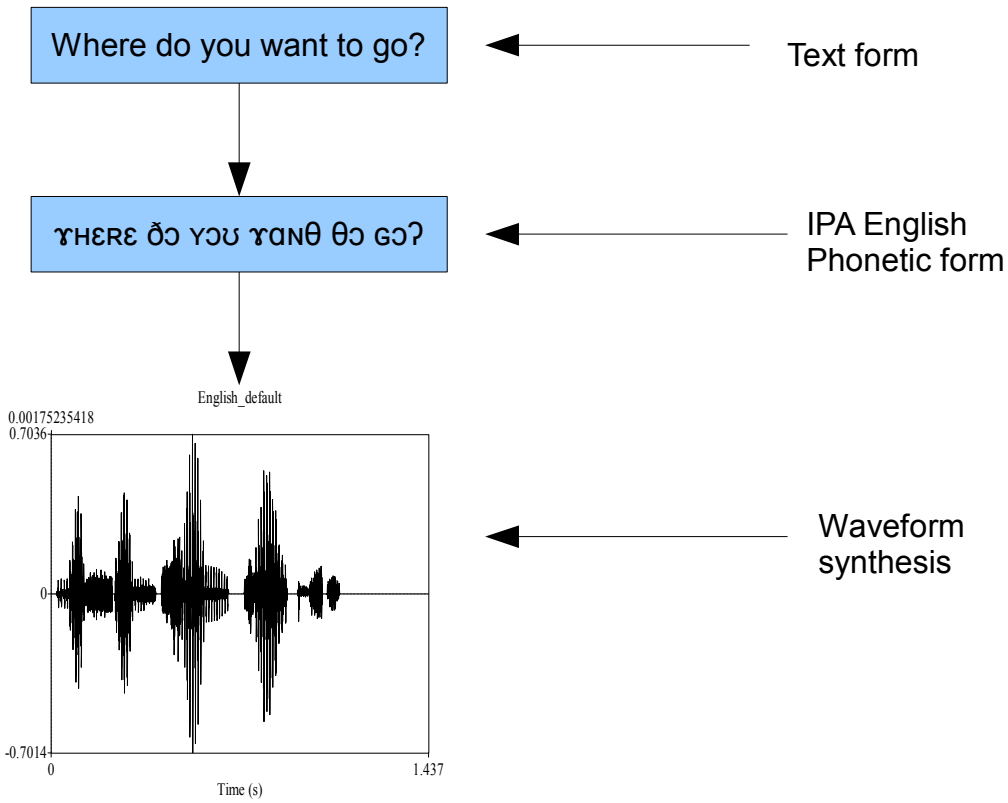

Figure 4. speech synthesis process for the prompt “where do you want to go?”.

2.2 W3C speech interface framework

Speech interaction has also been used on the WWW(world wide web). The world wide web consortium (W3C) has a voice browser working group to facilitate browsing the web using voice. It has several mark-up languages to work across different hardware and software platforms. These mark-up languages are for dialogue management, speech recognition & synthesis, understanding

language semantics, dialogue library management and other speech interaction related tasks. It contains a voice browser as a platform to interact through speech. The main work of voice browser is to accept DTMF(dual-tone multi-frequency) inputs and/or speech inputs, and to generate speech output based on concatenation or articulation.

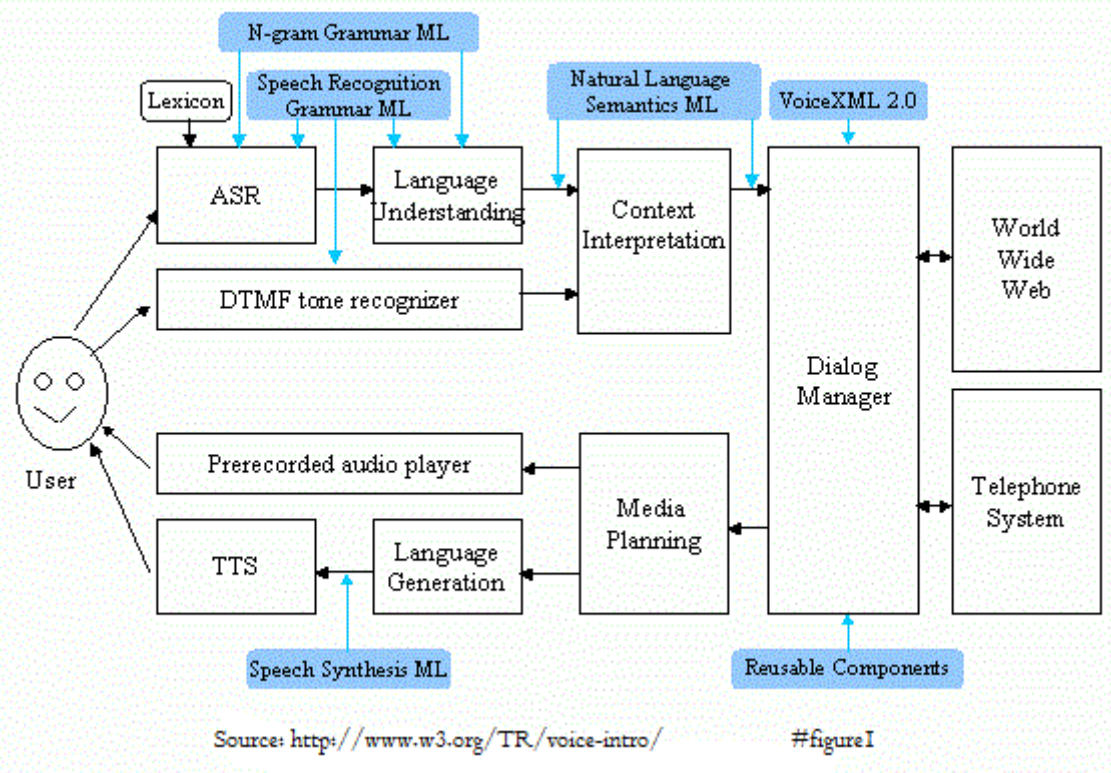

Figure 5. W3C speech interface framework [W3C-Speech,2000]

In Figure 5, components of the speech interface framework are represented by rectangles and arrows represent the data flow. The components are information processing units that take speech inputs and give outputs. The ASR (automatic speech recogniser) recognises user utterances and pass them to the language understanding component. The very basic architecture of ASR has been shown in Figure 1. Further, the recognised utterance will be matched by the stored grammar and/or with the developer specified grammar. A developer can specify grammar using various mark-up languages.

Speech recognition grammar mark-up language and N-Gram grammar support the ASR and language understanding component. However, the user can use DTMF as another mode of input. It utilizes touch tones from the key-presses to recognise the pressed key on the device. DTMF input is also supervised by the grammar. Further, the language understanding component uses semantic

information of the text provided by the ASR to map the information forward to the context interpreter that deals with the contextual information of the dialogue. The dialogue manager defines the dialogue policy using VoiceXML (voice extensible mark-up language). The VoiceXML is the core language for the speech interface framework.

In the speech interface framework, dialogue manager guides the speech interaction process. A dialogue may contain forms, menus, links etc. for the speech interaction. Moreover, it also integrates with the outer sources like WWW and telephone system to process the data. Further, the media planning component chooses the form of output. The output can be either concatenated audio or articulated one. A language generator is a component needed for synthesising the transcript into a meaningful way, and the TTS (text to speech) component then generates the sound out of text. The whole process described here requires a voice browser to execute. However, the mark-up languages are explained in further subsections.

2.2.1 SSML (Speech synthesis mark-up language)

The main task of this language is to synthesise the text to make an effective speech output. This language specifies various properties of the output i.e. speed, volume and quality. Further, the TTS system converts the generated strings to the acoustic forms. The TTS process consists of two steps: document creation, and document processing. The SSML do document creation that defines the form of the input to the TTS system. In the document processing stage, required processing steps are performed by TTS system to generate the speech output.

2.2.2 Natural language semantics mark-up language

This language is used for semantic interpretation of various type of inputs such as speech, text and DTMF. It deals with the user input as utterance and interprets it in a meaningful form of data. There are five components that generate natural language semantics:ASR, natural language understanding, other input mediums, reusable dialog component, multimedia integration component. The ASR is the main speech recogniser component as described in Figure 1.The natural language understanding component deals with interpretation of utterances. However, Input mediums may include keyboard, DTMF, mouse etc. The reusable dialog component uses previous dialogues for similar queries by reducing extra workload. Furthermore, the multimedia integration component is required to include various from of data types.

2.2.3 N-gram grammar mark-up language

The N-gram grammars are collection of symbols based on the probability of occurrence of a symbol based on prior occurrence of N-1 other symbols. Usually, N-gram grammars are constructed on the statistics collected from a large set of text using co-occurrence of words. Therefore, N-gram grammars are useful for the non-strict grammar based applications. It is based on XML syntax and the file format of it is based on tree data structure called the grammar tree.

2.2.4 VoiceXML

The VoiceXML is the epicentre of this thesis. It started in 1995 as an XML-based dialogue design language intended to simplify the speech recognition application development process within an AT&T project called phone mark-up language (PML) [VXML2,2001]. AT&T, IBM, Lucent, and Motorola founded the VoiceXML Forum, which developed VoiceXML version 1.0 [VoiceXML,2011]. The specification was submitted to the W3C [W3C,2011], whose voice browser working group [W3C-Voice] defined version 2.0 [Larson,2003].

As shown in Figure 5, it uses the input in a well specified manner for processing from various components. It contains a series of dialogues for the different interaction sessions. It is a form of mark-up language that incorporates various XML (extensible mark-up language)'s for creating audio dialogues that uses synthesised speech, digital audio, speech & DTMF recognition, speech recording, telephony interface and various other related features. Moreover, the use of high-level menus and forms instead of procedural programming code reduces the programming time and effort to a great extent. Thus, it spares the time for performing additional tasks like testing and improvement.

Further, Figure 6. presents the basic architecture model of VoiceXML. In the architectural view, the document server is a web server that processes the requests from the VoiceXML interpretor. The output of the document server is a set of documents related to the context for processing in the interpretor. Further, the VoiceXML interpretor is wrapped into the VoiceXML interpretor context which runs in parallel to sense any special escape phrase e.g. exit, stop etc.

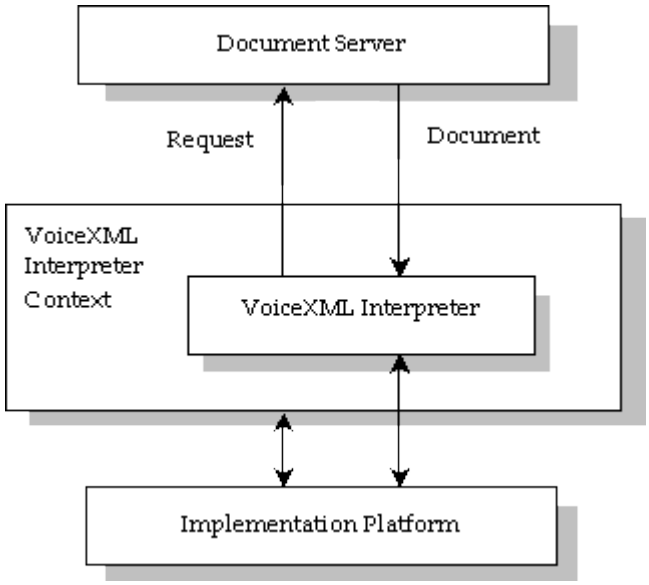

Figure 6. VoiceXML architecture[VXML2,2001]

Moreover, the VoiceXML interpretor and interpretor context interacts with the implementation platform which performs event handling tasks. The VoiceXML forms a finite state machine (FSM) sort of structure, where at any point in time the user remains in a state, called 'dialog'. A dialog is facilitated by speech interface. Furthermore, the main concepts of the VoiceXML are dialogs, sessions, applications and grammars.

Dialogs The dialogs are the basic blocks of speech interaction. These are similar to actual dialogs in natural conversation. Each dialog creates a block in VoiceXML and the VoiceXML document may contain a series of blocks. A dialog can be classified into two types of interaction methods: forms and menus. Forms collect the data and fill the appropriate fields. They may have a specific grammar for a specific field or for the whole form. On the other hand, menu provides a list to select from some options, on the basis of selection, a menu transfers the control to other dialogs. A dialog may have sub-dialogs present. Sub-dialogs work like procedures, and return the value to their calling dialog.

Session A session is a timer that signifies the active state of a VoiceXML interpreter context. It starts when the VoiceXML interpreter starts. A session may end on the request of user, document or the interpreter context itself.

Application An application is a collection of documents in the project. These documents are interconnected for the transfer of control or state. All the documents under one application are governed by the root document. This root document remains active until the user remains in the application session. It may contain root level grammar as well. When the user moves out of the document, the root of the previous application becomes inactive. Further, Figure 7 depicts the control flows between one document to another in the serial manner. In a VoiceXML application, the root document always remain active and control flows between document and root. However, the control flow is also permissible between documents.

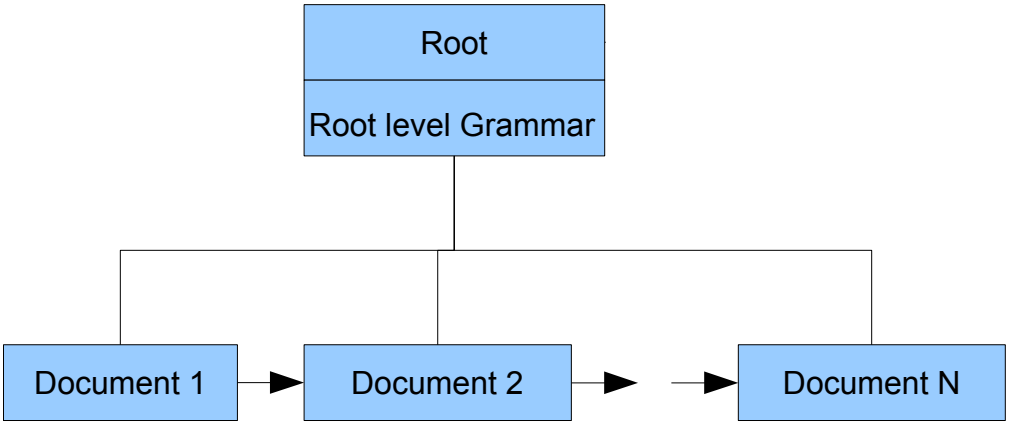

Figure 7. Application structure.

Grammar The speech recogniser uses grammar to bind the processing by defining what to listen. The grammar may define words to match, and specific patterns of these words. It is included in each document of the application. In VoiceXML, there is an option to use speech and/or DTMF grammar. However, the grammar specification is described in section 2.2.5 in detail.

Furthermore, to demonstrate the use of the above mentioned concepts a very basic example of a menu based application to get directional information is shown in Figure 8. However, there is no special grammar linked to the document. Further, the '<prompt>' tag is used to create a block for the prompt/instruction to the user, requesting the input. The words are included in the '<choice>' tags to match, and for each match, the control transfers to the next document. The link to next document is given with a value to the 'next' property of the '<choice>' tag. However, the '<help>',

'<nomatch>' and '<noinput>' blocks are for special purposes.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1">
<menu>
<prompt>
Please say one of <enumerate/>
</prompt>
<choice next="´North.vxml">
North
</choice>
<choice next="South.vxml">
South
</choice>
<choice next="West.vxml">
West
</choice>
<choice next="East.vxml">
East
</choice>
<help>Please say directions.</help>

<nomatch>Query not matched.</nomatch>

<noinput>Please say one of <enumerate/></noinput>

</menu>
</vxml>
```

Figure 8. A menu based VoiceXML example.

However, the developers need to represent the structured grammar specifications that is to be used by ASR . The W3C uses two forms of grammar format syntaxes: ABNF, XML. In VoiceXML, “<grammar>” element is used to describe the grammar representations to the document. Further, the grammar representations can be provided by using two ways: inline grammars, external grammars. The deference between two is that inline grammar represents the grammar inside the VXML document, while the external grammar can be linked to the document using the source address of the grammar file as uniform resource identifier (URI).

Moreover, the conceptual level representation of the inline grammar usage is depicted in the Figure 9, where individual grammars are embedded under the respected elements. Consequently, the embedding determines the scope of the grammar.

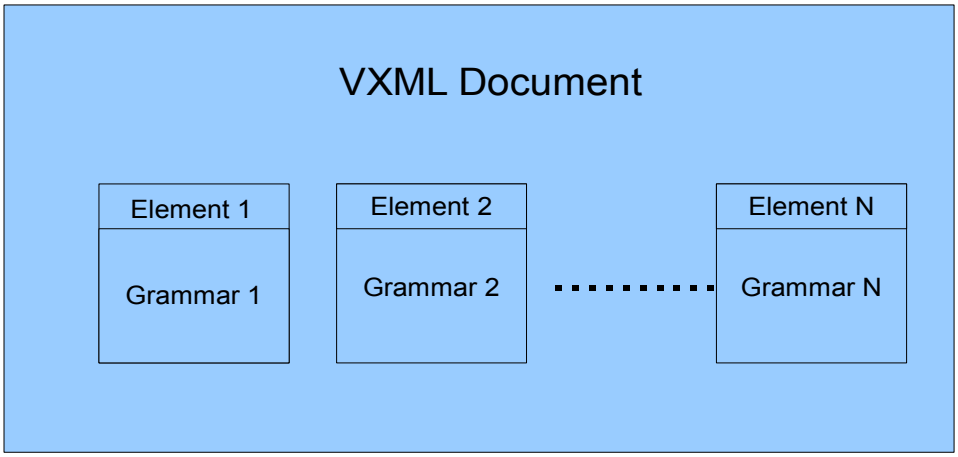

Figure 9. Inline grammar

On the other hand, external grammar method uses grammar files to hold the grammar representation. There are many VoiceXML compatible formats to represent grammars, such as GSL (nuance grammar specification language), JSGF (java speech grammar format), NGO (nuance grammar object) etc. However, the GSL format has been used in this thesis to make external grammar file with an example application in section 4.

In Figure 10, the external grammar structure is represented, that shows the conceptual connection between elements and the external grammar files. Despite the given representation, there can be a common global grammar file connecting distinct elements.

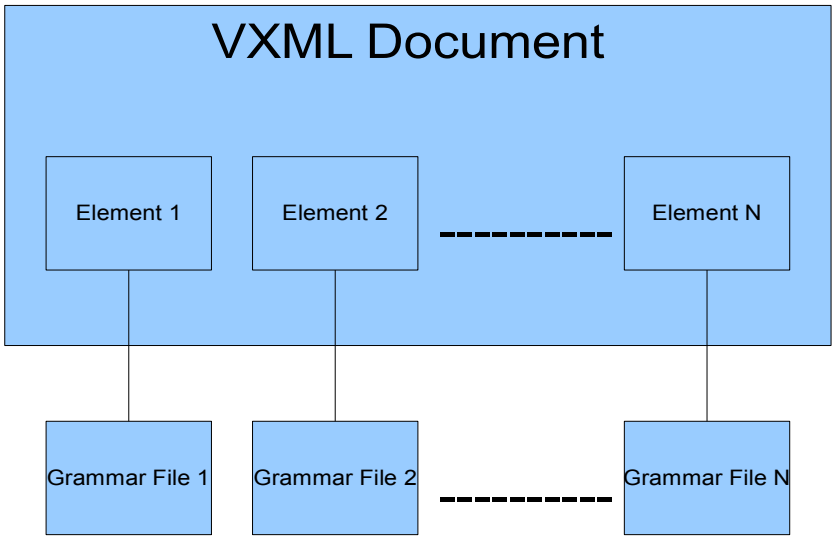

Figure 10. External grammar

2.2.5 SRGS (speech recogniser grammar specification)

SRGS specifies the words and phrases to match from the grammar storage with the speech input. Further, it has a grammar processor that processes the speech inputs. The grammar processor accepts the user input and matches it with the stored grammar to recognise. Speech recognising and DTMF are two primary modes of getting input. Further, the VoiceXML provides in-line grammar facility to use grammar definitions within the documents based on SRGS syntaxes. The SRGS grammar format has two forms: XML, ABNF (Augmented BNF).

2.2.5.1 XML form

It is the XML based syntax for the grammar representation. Figure 11 describes it further. The '<one-of>' tag creates a block to define different options to match with the speech input. Each word to be matched is represented by using '<item>' tag. Moreover, the '<item>' tag may contain weight attribute to bias the matching item over others.

```
<one-of>
<item>Michael</item>
<item>Yuriko</item>
<item>Mary</item>
<item>Duke</item>
<item><ruleref uri="#otherNames"/></item>
</one-of>

<one-of><item>1</item> <item>2</item> <item>3</item></one-of>

<one-of>
<item weight="10">small</item>
<item weight="2">medium</item>
<item>large</item>
</one-of>

<one-of>
<item weight="3.1415">pie</item>
<item weight="1.414">root beer</item>
<item weight=".25">cola</item>
</one-of>
```

Figure 11. An example of XML based grammar [SRGS,2004]

It is based on XML form. Therefore, it requires the declaration of XML version for the grammar file in the grammar element i.e. the root element of XML grammar. The Figure 12 shows the standard grammar element which accommodates the code given in Figure 11.

```
<grammar version="1.0"
xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-
grammar/grammar.xsd">

…

</grammar>
```

Figure 12. a standard grammar element[SRGS,2004]

2.2.5.2 ABNF(Augmented BNF)

On the other hand, BNF (Backus-Naur form) is a formal notation for defining the programming language syntax. It contains alphabets which are sets of terminal & non-terminal symbols, rules which are made for the implication clauses to expressions and axiom which is a starting symbol. The general syntax is,

Syntax *(SYMBOL := EXPRESSION)*

Further, ABNF is a modified version of BNF that is used in many internet specifications. The ABNF has advancements like naming rules, repetition constraint, alternatives, rder flexibility etc. over BNF.

```
#ABNF 1.0 ISO-8859-1;

// Default grammar language is US English
language en-US;

// Single language attachment to tokens
// Note that "fr-CA" (Canadian French) is applied to only
//  the word "oui" because of precedence rules
$yes = yes | oui!fr-CA;

// Single language attachment to an expansion
$people1 = (Michel Tremblay | André Roy)!fr-CA;

// Handling language-specific pronunciations of the same word
// A capable speech recognizer will listen for Mexican Spanish and
//   US English pronunciations.
$people2 = Jose!en-US; | Jose!es-MX;
```

Figure 13. An example of ABNF based grammar [SRGS,2004]

In Figure 13, regular expression structure is used to define the options for words to match using ABNF. However, the basic syntax for defining symbols has been used. Symbols are represented by a '$' sign before the symbol name. Moreover, the language declaration facility is utilised by using language identifier after the regular expressions. The Language identifiers use '!' sign before the declaration, e.g. !en-US for US english. Further, ABNF has been applied for GSL(Nuance Grammar Specification Language) format.

Nuance Grammar Specification Language (GSL) format

GSL is a language used for formal specification of the speech grammar to a nuance system application. It describes a set of word sequences for the speech recognition component. The word sequence is usually specified in a grammar file. Further, a grammar file may contain one or more grammars. A GSL file starts with grammar name and description.

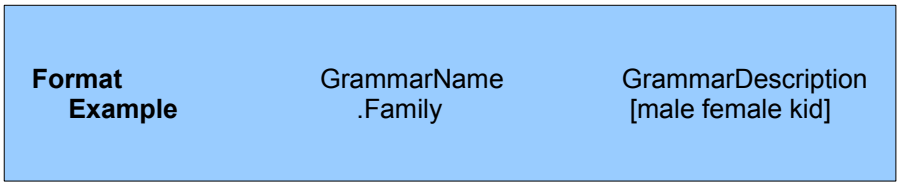

Figure 14. GSL format.

In Figure 14, the grammar file has a name, Family. This grammar name is used for the reference to a grammar file by other grammars and/or applications. Further, GrammarDescription is the phrase or word set with operators.

However, the complexity of a grammar greatly affects the speed and accuracy of the recognizer. Complex grammars must be constructed with as much care as complex software programs [Nuance,2002]. Furthermore, grammar hierarchy can be an approach to reduce the complexity. Moreover, grammar hierarchy is possible using GSL. A grammar hierarchy can be created by breaking grammar into sub-grammars to facilitate grammar development with the re-usability and simplification.

This section started with the brief information on the basic elements of speech interaction technology. Further, the W3C speech interaction framework was explained. Similarly, the next section will discuss the elements of the model based code generation, as well as techniques.

3. Model based code generation

3.1 Model based development

Contemporary software developers need to consider many possible solutions for a given problem. Sometimes multiple technologies are required for a specific situation. For this reason, the technological space can be considered as a set of multiple intra-compatible technologies. It is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [Kurtev et al., 2002]. A model is an abstraction of real world. It helps understanding a complex problem and its potential solution. Therefore, it seems obvious that software systems, which are often among the most complex engineering systems, can benefit greatly [Selic, 2003].

The essence of the model based development is to shift the knowledge about the implementation details from the minds of programmers to the templates of code generators that automatically translate models into implementations [Hemel et. al., 2008]. The objective is to increase productivity and reduce time by enabling development and using concepts closer to the problem domain at hand, rather than those offered by programming languages [Sendall and Kozaczynski, 2003]. It can be characterized by representing different ways in which the models synchronize with the source code [Brown, 2004]. The characterization is called model spectrum which is represented by Figure 15.

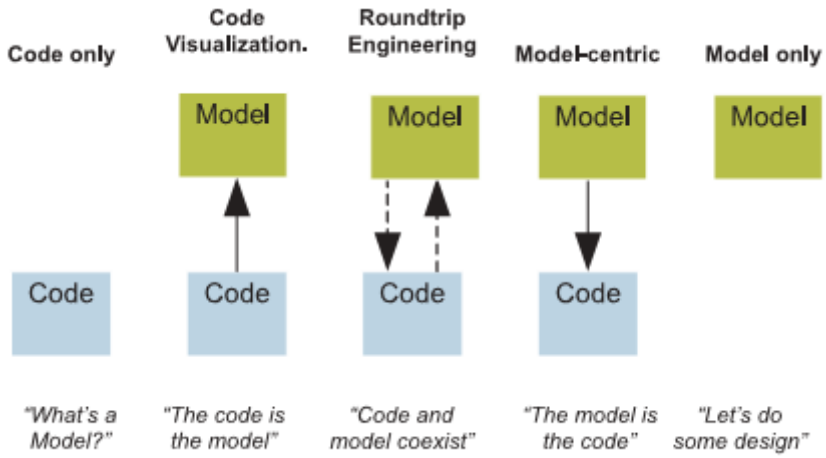

Figure 15. The model spectrum [Brown, 2004].

In figure 15, The model has been characterized according to its role. It can be used for visualizing the code. It can also co-exist with the code in parallel i.e. round trip engineering. It can also contain the code in it which is model centric approach. However, in the case of a design, model can exist independent of the code. Additionally, Selic [2003] describes pragmatics of the model based development such as model level observability, model executablity, generated code efficiency, scalability, and integration.

Model level observability

The code generation is essence of the software modelling. The observability property is required to detect any error at the runtime or compile time. Due to high level of abstraction in the models the task of error diagnosis and correction facility must be provided with the automated code generation facility. Moreover, it is required to achieve the understandability of the system and code through the model as a whole.

Model executability

This property deals with the feature to execute the immature model. Usually software development works as a feature wise development process thus verification of the incomplete version is needed. This property may help in verifying the progress direction, and can be useful in making early changes towards the right directions. Furthermore, it can also be helpful for the developers to run code at the simulation mode by executing the unfinished product.

Generated code efficiency

This property is a main requirement for a typical software development process because it is directly related to the project cost. Further, the efficiency comparison is required between the human developed code and the automated generated code. However, the reliability and organizability of the automation is considered better than the human. There are two factors in which efficiency is decomposed: performance, memory utilization. In both of these, the automation can play a stable role.

Scalability

Similar to software development life cycle, product maintenance is a property required. The scalability is needed in the modern software development processes due to the large industrial applications and the number of developers working on a single project. Two metrics are considered here: compilation time, system size. However, the compilation time is further subdivided into the system generation time and the turnaround time. The compilation time depends on the time that compiler takes to generate the code from the model. Calculated as,

$$Total\ time = Code\ generation + Code\ compilation$$

Integration

World is moving fast with technologies and techniques thus a frequent update becomes a requirement. Therefore, the software should be integrable with legacy systems. Moreover, this kind of integration or adaptability helps a model to remain in the useful state. However, the model must be able to take advantage of the legacy code libraries and the interface software. Further, such an integration can be done by using the tailored code generators or direct calling functionality from within the model.

3.2 Code generation

The automatic generation of programs from the models is the main feature of the model based development. It results in the graphical understanding and skeletal code generation of the concept. However, a fully automated approach is capable of generating complete programs from the model, and execution of the generated program to verify it further. Consequently, the automation results in accelerating the production by a great extent.

A basic example of the code generation concept is the compiler. A complier works by generating an IR (intermediate representation) from the source code. Further, it is parsed into the target program. However, the IR can be a graphical representation of the source code. This graphical from is called syntax tree. Similarly, the syntax tree parsing can be applied to a model to generate the target code. Further, Figure 16 shows the target code to be generated.

```
start
a = 3;
b = 4;
If ( b > a )
true : print a;
false: exit;
```

Figure 16. The target code.

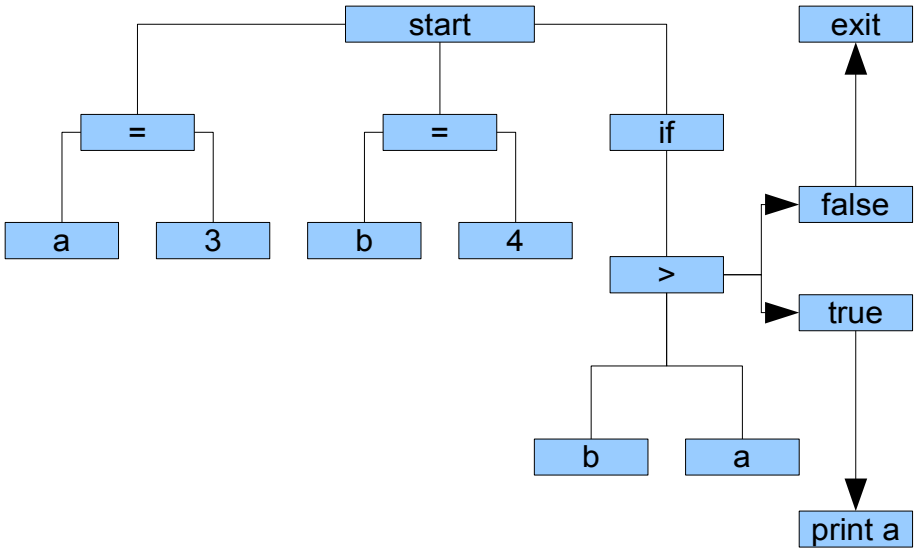

Figure 17. An abstract syntax tree.

In Figure 17, the parsing of an abstract syntax tree is depicted. The parsing of a tree depends on the rules defined for the control flow. However, this example uses depth first parsing. A tree representation could be at an abstract level to facilitate the understandability. For this reason, it contains the information at an abstract level by leaving some non vital information.

However, the application of the code generation concept is referred as model based code generation. The model is based on formal syntax and, in some cases, semantics similar to HLL (high level language). Further, the application source code is automatically produced from graphical models of system behaviour or architecture [Bell, 1998].

Bell [1998] describes the code generation in Figure 18. The left portion of it describes the HLL code generation, and the other shows the model based code generation. In the HLL code generation, the compiler system transforms the HLL code to the executable code. The input to the compiler system can be from various sources such as HLL code, run-time libraries, and code options. Additionally, code options and run-time libraries append the input to the compiler with performance based options and some extra set of code respectively. On the other hand, the model based code generation uses translator to transform the model and its associate information into source code. The translator uses object models, architectural options, and associated libraries to perform the task. However, the architectural options contain templates to generate the source code.

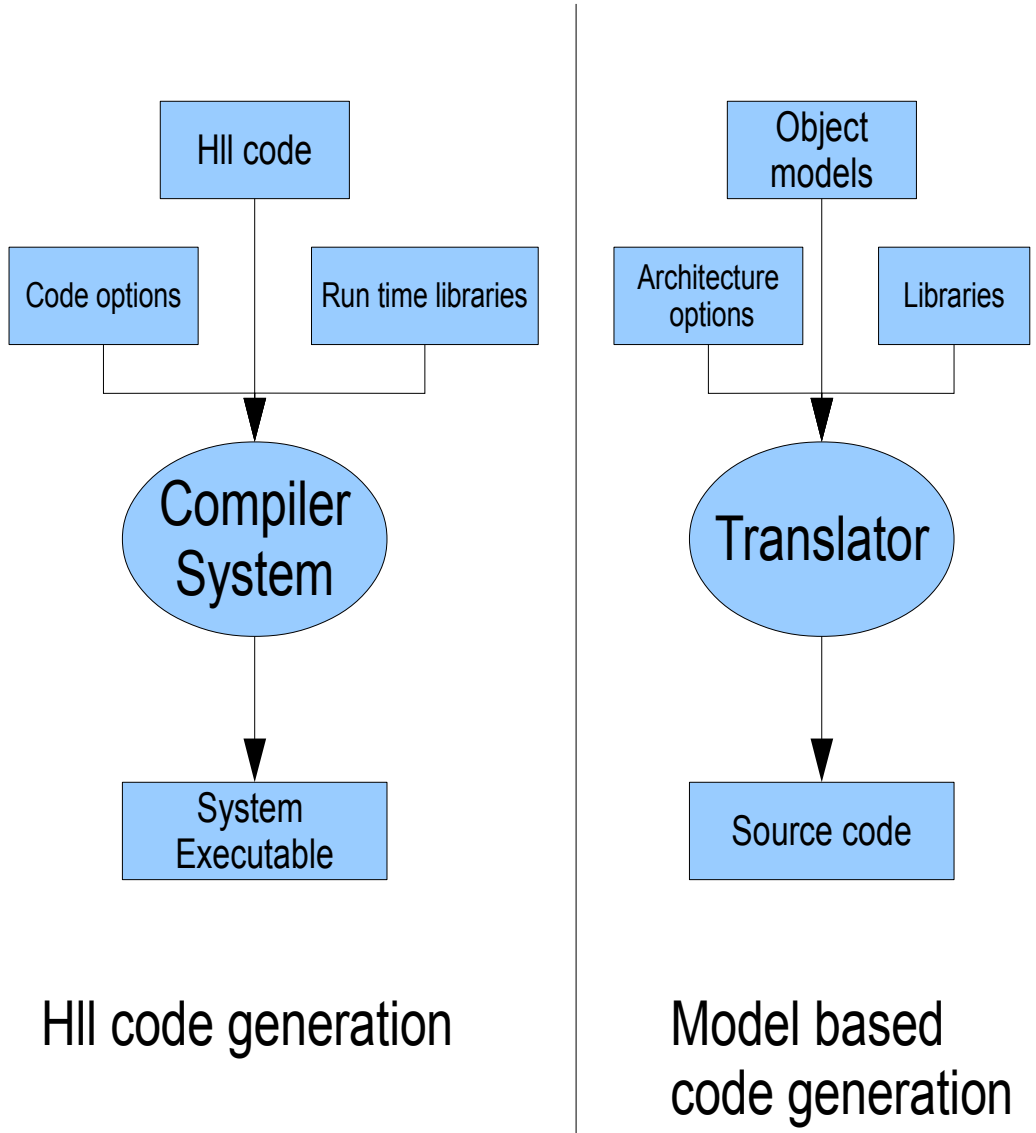

Figure 18. Code generation [Bell, 1998]

Usually, the character based code generation becomes so extensive that it becomes difficult to follow. Therefore, the model based code generation can be accepted. It can be done by defining a generator that maps the graphical model to the target code. Further, the generator definition contains a template for the optimised domain specific code. For this purpose, the GOPRR (graph object property role relationship**)** model has been considered fit because of the features it provides to generate the pace and ease in the development. The features are explained further.

Graph concept

The GOPRR is an extension of the OPRR (object property relationship role) model. Further, the addition of the graph concept increases the extendibility of the OPRR model. A graph represents the whole concept including the objects and their relationships. However, a graph can be decomposed to the parent graph of itself. This decomposition in turn increases the abstraction of the conceptual model. Consequently, it simplifies the larger user concept by showing the minimum required detail at one point of time. Moreover, the re-usability can also be achieved by using the standard sub-graphs in other concept models as well.

In Figure 19, a basic graphical representation for the graph concept has been presented. In which, the main graph contains two elements/objects. Further, these elements are represented in abstracted from in the main graph. Furthermore, each element is logically connected with the respected sub-graph. Consequently, the sub-graph contains the detail information as a set of elements.

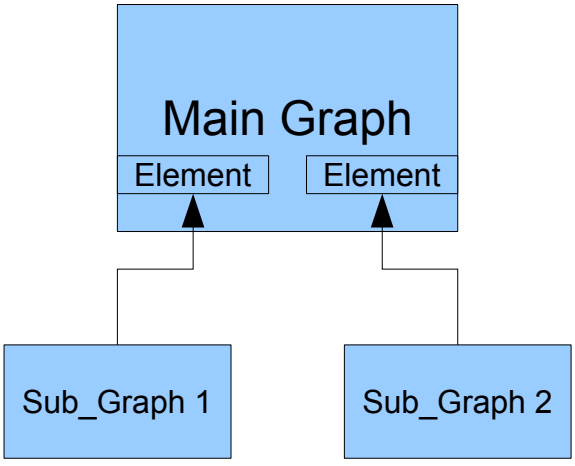

Figure 19. basic graph structure.

Object orientation

The Graph extension makes the use of generalization and specification properties of the object-orientation. Further, the graph separation helps in hiding complexity and helps in generalization by making the sub-graphs to facilitate the specification. However, the polymorphism property of object orientation has also been utilized by reusing the basic concepts in different graphs.

Integration

The concepts have been reused in different graphs but the changes affect the instance level also i.e. change in the object in parent graph makes it visible in the child graph also. This facility is called the integration. As a result, it increases the concept sharing and makes the whole structure well connected . Furthermore, it accelerates the development.

Constraints

The possibility of defining constraints on concepts makes the modelling process error free by using the real time compiler or by including the integrity checker at the runtime. Consequently, it increases the model building speed with effective accuracy.

Furthermore, the upcoming section explains the basic concepts of the GOPRR model.

3.3 Graph object property role relationship (GOPRR) model

The GOPRR model is a modification of the ERM for the meta-modelling. The main concern for developing the GOPRR was the ease of use and tool support i.e. MetaCase. The components of GOPRR are:

Graph The graph represents a whole modelling language. It contains other concepts of GOPRR i.e. object, properties, roles and relationships. However, it may contain other concepts that are not part of the name “GOPRR” itself. The graph can also be seen as a container to hold every concept, and may contain further sub-graphs.

Object The objects are the main concepts in the graph. They are similar to entities in the ERM. These objects are usually described in various shapes, and contain properties. Their definitions are often reused to achieve efficiency and rapidness in the development.

Properties Properties are the attributes as described in the ERM. They give values to different concepts of the model. The values for the properties can be inline or of external type. The inline properties contain various data types such as string, text, list, number, boolean, etc. On the other hand, external types may contain links to other outer sources for the properties such as files, web, links, functions etc.

Relationships Associations/Connections between objects are the relationships. These relationships can depict the flow of the data or control between the concept, and are usually represented by the lines. Further, they may contain constraints and bindings on them in order to interconnect.

Roles Usually roles are intangible representations. They are sketched at the end points of the relationships describing the role that an object is playing in a relationship. Moreover, the roles also show the inheritance level in an ontological meta-model.

However, there are other important concepts related to GOPRR. These concepts normally work in background to define the rules, and monitor the control of the flow.

Bindings The bindings connect relationships between the roles. They need to be defined to verify the connection possibilities. Further, there are one or more objects involved in the binding connections using the relationships with the roles.

Decomposition It can be used to define the sub-graphs decomposed to a graph as described in Figure 19.

Explosions The explosions are used to relate the graph concepts such as objects, roles, relationships etc. with other graphs to encapsulate the details. Consequently, this concept creates an abstract graph for the ease of understanding.

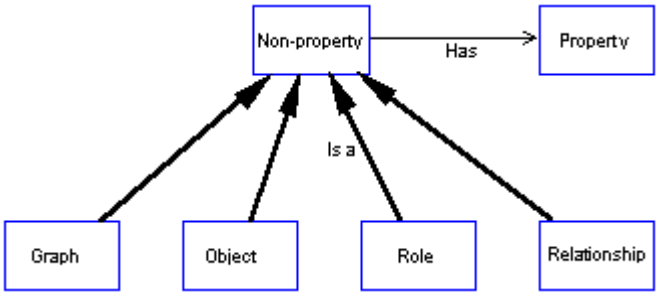

Figure 20. Properties and Non-Properties Relationship [Kelly1997].

In Figure 20, two categories for the GOPRR concepts are described: properties, non-properties. However, concepts like graph, object, role and relationship are non-properties but may contain the property itself. Further, a property can be shared between other concepts. It is defined in a specific data type.

In the upcoming section, the VoiceXML language based development with the GSL grammar has been utilized in the example used for the case study. Additionally, the MetaEdit+ tool has been used for the development of the speech grammar model.

4. Speech grammar modelling

4.1 The problem space

The execution of a typical VoiceXML based application is based on the control flow between dialogs. The dialog can be viewed as a block of interconnected elements. In VoiceXML, a dialog consists of the actual speech interaction sequence. Further, the interactions between these dialogs are governed by the dialog specific, document specific and root based grammar(s). The application structure using root based grammar is depicted in Figure 7. The applicability of a grammar is global when it is linked or described at the root level. Similarly, Figure 9 & 10 show overview with document specific grammar. However, the dialog specific grammars are linked/described in a specific dialog .

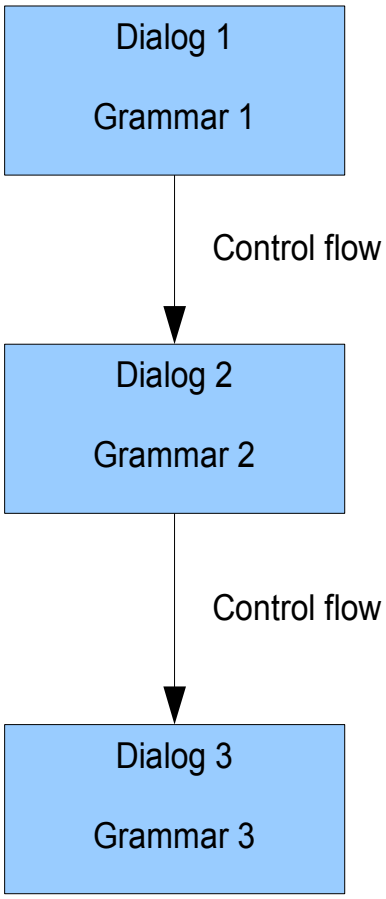

Figure 21. Control flow between dialogs of VoiceXML

In Figure 21, a possible control flow between three dialogs is represented. Moreover, the conceptual level presented in this Figure depicts that each dailog holds its own dialog specific grammar for the ASR. Similarly, the grammars may also hold connections between themselves as depicted by Figure 29. The typical approach concentrates connecting the dialogs while ignoring the grammar based connection. Consequently, the document complexity increases as depicted in figure 22.

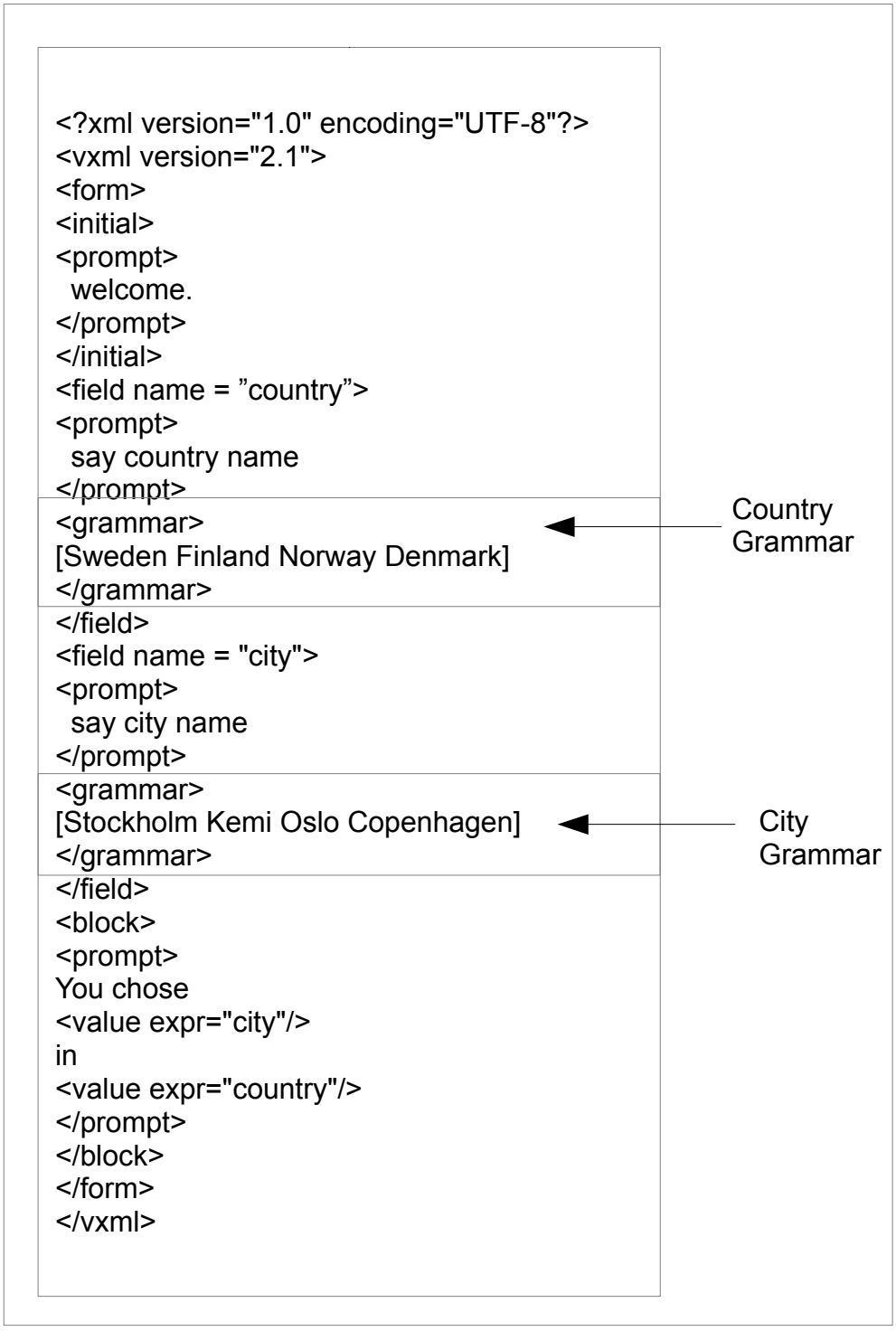

Figure 22. An example using 2 small internal grammars.

In Figure 22, A simple dialogue system example is presented which repeats the input. It uses two grammars, one for country names and other for city names. These are stuffed in between the VoiceXML code. As a result, it increased the code size and complexity.

4.2 The solution space

However, dialogs can share grammars but can not customize them. Instead of only concentrating on the inter-dialog control flow, the inter-grammar control flow can also be included. Moreover, the separation between both contol flows can be done in parallel. Accordingly, this may lead towards complexity reduction. Additionally, in this theses, the grammar model is created for the graphical representation and automated grammar generation of it.

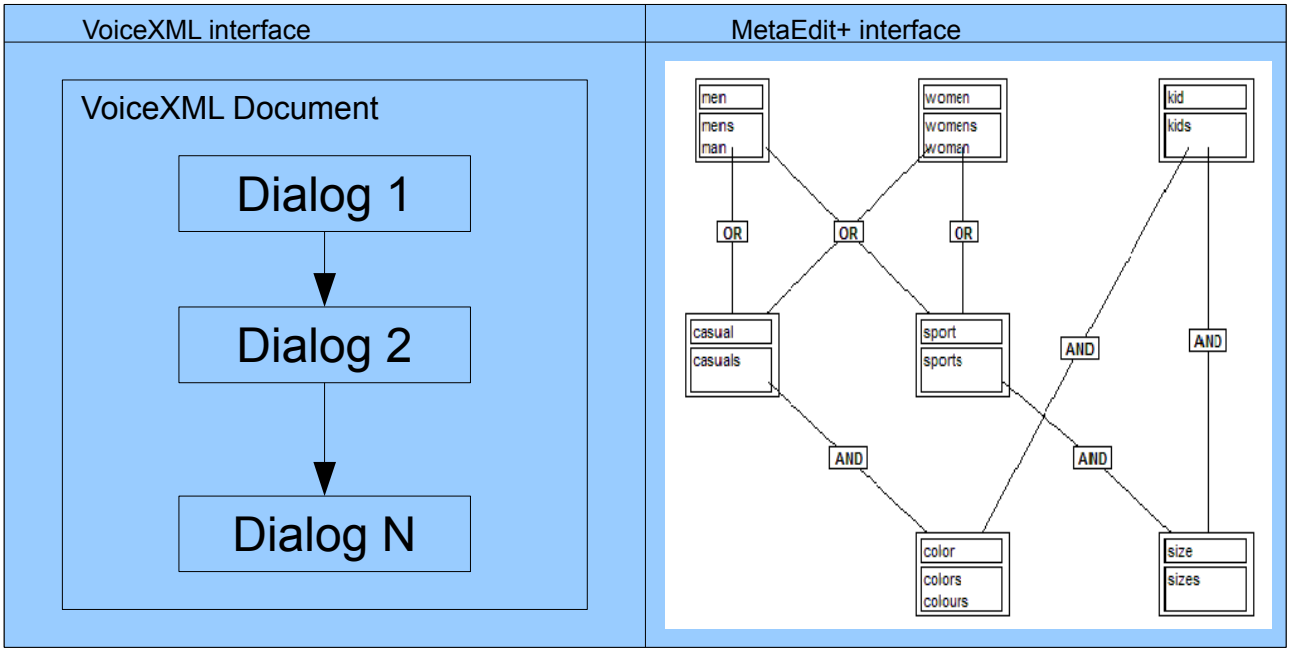

Figure 23. Grammar and VoiceXML development interfaces

In Figure 23, the dual application development view has been represented. It separates the grammar development from the document development. In left side of the panel, the traditional VoiceXML development interface is shown. On the right hand side, the MetaEdit+ interface is presented. It is used to develop the grammar model. Further, it generates the GSL file to connect with the VoiceXML document.

In this work, I have considered the external grammar interface (i.e. linked grammar) as a base for

the investigation of the grammar model structure for the VoiceXML application development. There are two main reasons for using the external grammar over in-line grammar: grammar hierarchy, and complexity reduction.

In VoiceXML, the inline grammars contain one-to-one structure with the dialog blocks, as shown in Figure 21. The grammar hierarchy is used to represent the connections between different dialog specific grammars. Further, it can be defined in a single external file. This can be done by creating an external grammar on the root level. Consequently, it would reduce the complexity by using a single file to store all the recognition data instead of using a scattered interconnected grammar structure. Since the GOPRR modelling approach has been applied for the grammar representation. For this reason, the MetaEdit+ tool is used for the implementation of the GOPRR based model.

4.3 The development environment: MetaEdit +

The MetaEdit+ is a multi-user, multi-method, multi-platform tool that supports the multi-representation view. Further, the design principles of this tool includes data independence, representation independence and level independence. In Figure 24, the general architecture for the MetaEdit+ is depicted. The main elements of the MetaEdit+ architecture are environment management tools, model editing tools, model retrieval tools, model linking & annotation, and method management.

The environment management tools are used to manage features for the whole working environment. They may contain the basic window features and the main software platform functionalities to launch the program. They can also use credentials if required. Further, the model editing tools can be viewed as a canvas to make the diagrams. The functionalities for creating, modifying and deleting the models are included in these tools. Moreover, the graph designing and viewing can be done using these tools. The model retrieval tools are needed to fetch the designed concepts from the repositories for editing and reviewing. These tools may present the shared view for the model and meta-model level.

Further, the model linking tools are used to link some complex design concepts with the outer source of description. Additionally, the model annotation is required for maintaining the integrity. Similarly, the method management tools are required to perform operations on the methods used in meta-models. Furthermore, making various symbols for various concepts can be

done by using symbol editor. Likewise, the icon editor are used for making icons. However, the method management tools are most frequently used in making a model and defining behaviours for concepts. Lastly, the grammar code generation can be done by using the generator editor tool.

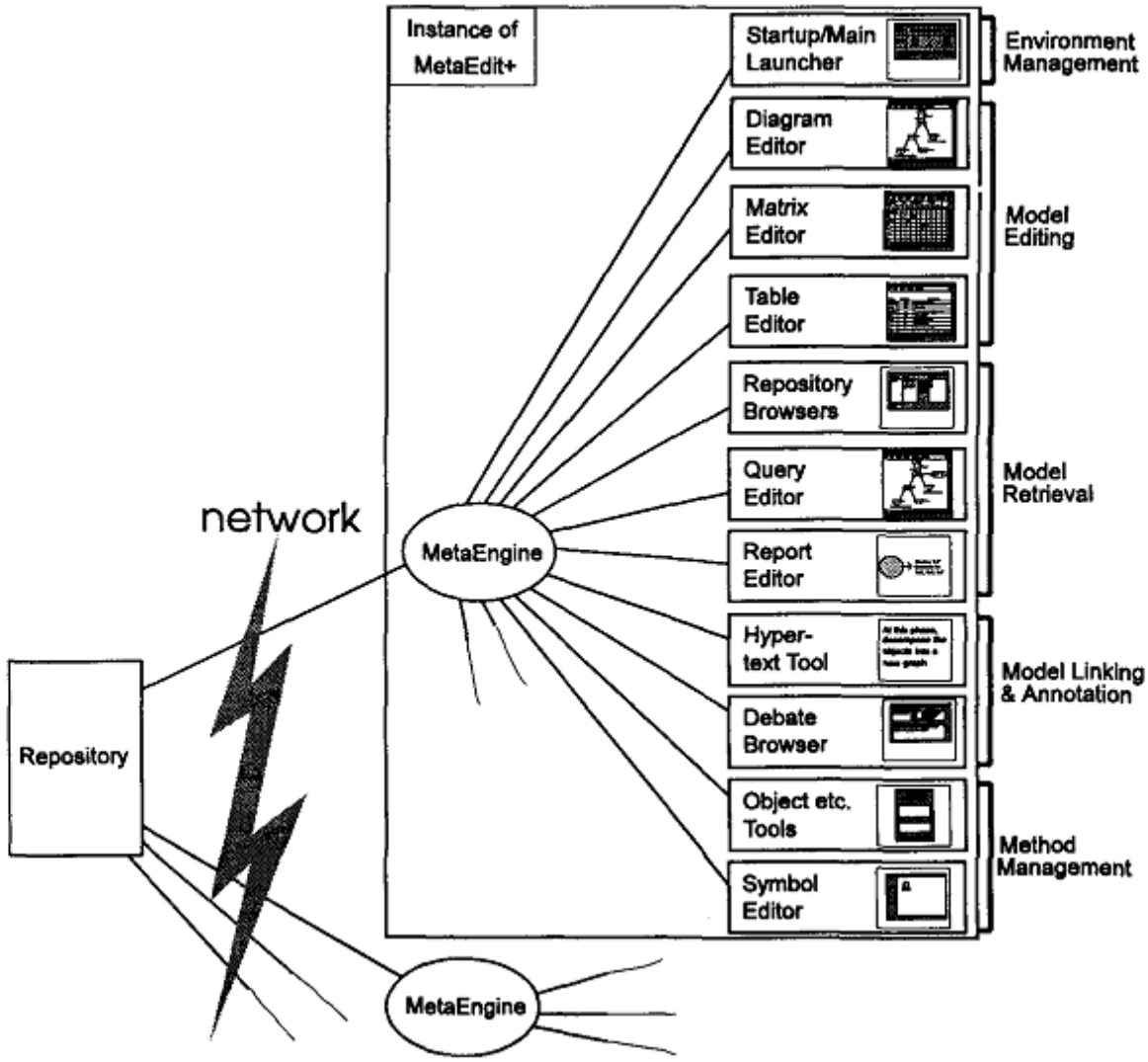

Figure 24. MetaEdit+ Architecture [Kelly et al., 1996]

MetaEdit+ tool was used for the implementation of the grammar model because it supports the rapid development of speech based applications. Tolvanen and Kelly [2009] highlight the rapid development provided by MetaEdit+ for the Voice control application domain in the Figure 25. However, the metric is depicted in man-days instead of months. The development in each domain is represented by combining both language development and generator development. It is evident from the Figure that the model based development of the voice control application is rapid than the other domains.

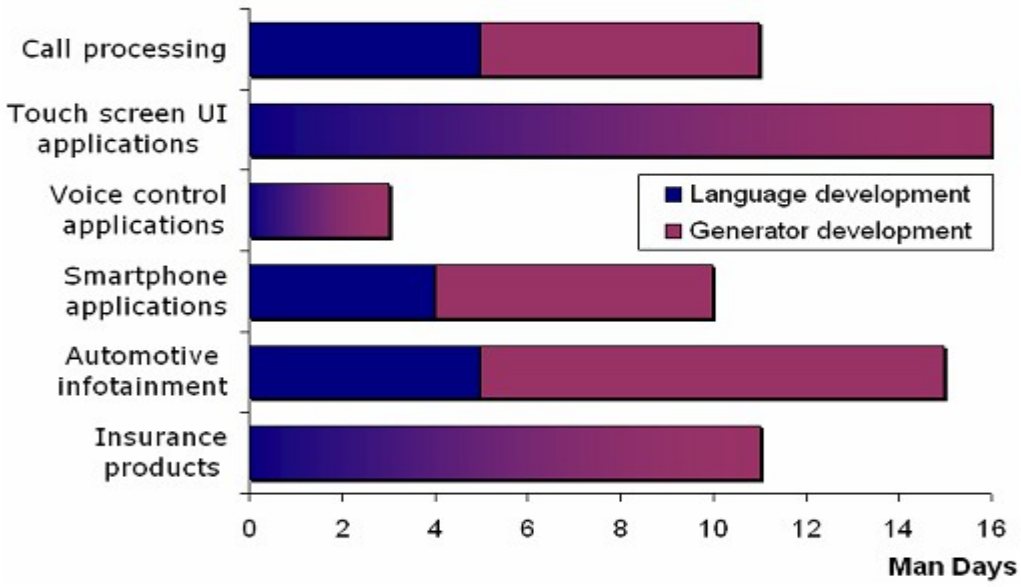

Figure 25. Industry experiences in selected domains [Tolvanen and Kelly, 2009]

For the validation, a basic IVR (interactive voice response) based “clothing shop” example has been taken by applying the GOPRR based approach to build the speech grammar conceptual model. The example contains a menu based system for the selection of appropriate cloth type using different options in an option tree kind of arrangement. The option tree is a general element of a typical GUI (graphical user interface). The application of the elements of GOPRR for the clothing shop example has been described further.

Graph

The clothing shop example uses 8 graphs including 1 main graph and 7 sub-graphs. Further, to maintain the abstract graph representation, seven sub-graphs are connected to the respective elements. This kind of connection is called “explosion” in MetaEdit+. This feature is useful in hiding details to make the grammar model abstract. Figure 26 shows the whole conceptual picture of the clothing shop example.

However, a graph requires bindings to be defined. They connect the modules for the entire meta-model specification. The example uses simple binding structure. It uses two relationship names 'OR' and 'AND', as shown by boxes in Figure 29. It uses two roles named 'from' and 'to' to connect objects using 2 X 2 mapping. The 4 possible bindings are shown in table 3.

Relationship	Role
AND	to
AND	from
OR	to
OR	from

Table 3. Possible bindings in the clothing shop example.

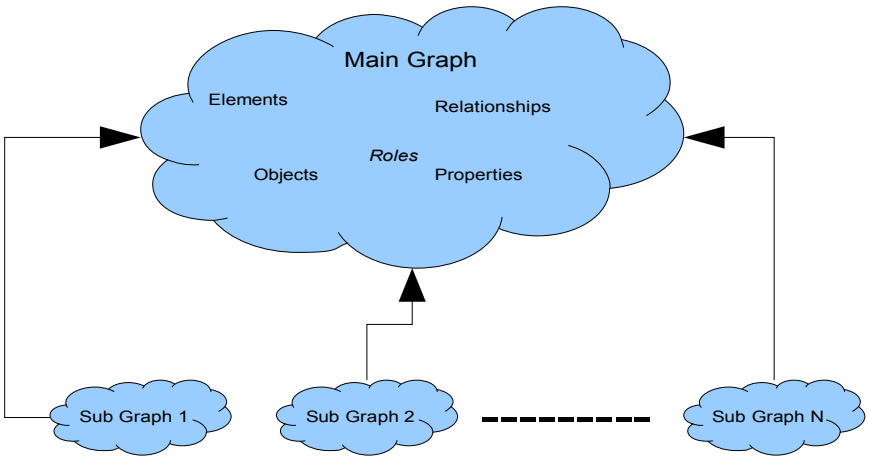

Figure 26. Graph inclusions

In Figure 29, the sub-grammars are connected to the elements as sub-graphs. The sub-graphs named Men_S, Kid_W, Women_S, Women_C, and Men_C are represented in the elaborated form here. They contain a set of options on the same level. Further, to make the graph more abstract, sub-graphs are created under their parent objects. These sub-graphs are connected to objects in main-graph using explosions feature of the MetaEdit+ tool. Furthermore, there are two other sub-graphs named 'size' and 'color' used in the example. These are not represented in Figure 29 to keep the representation simple.

Object

The object represents an element or an option of the menu. Further, there is an object to object movement of control in the GOPRR based structure. The connection between different objects is through the role and relationship concept as shown in Figure 28. In this, each object plays a distinct role in a relationship/connection. In the clothing shop example, the objects contain the set of words to be recognised by the speech inputs.

An object has a two level structure as described in Figure 27. The upper level contains the actual word to match with the utterance, and the lower level contains the alternative possibilities of it. Further, these objects are represented by rectangles in Figure 29.

Figure 27. An Object

Relationships

The relationship is the connection between objects. These relationships can be named. However, the relationships describe the connection type between objects and determines the flow of control within the GOPRR structure. In the clothing shop example transitional relationships named 'AND' and 'OR' have been applied. Further, the 'OR' relationship signifies the optional transition between the objects. On the other hand, the 'AND' relationship signifies the mandatory transition. The relationships have a close connection with roles and are further shown in Figure 28.

Roles

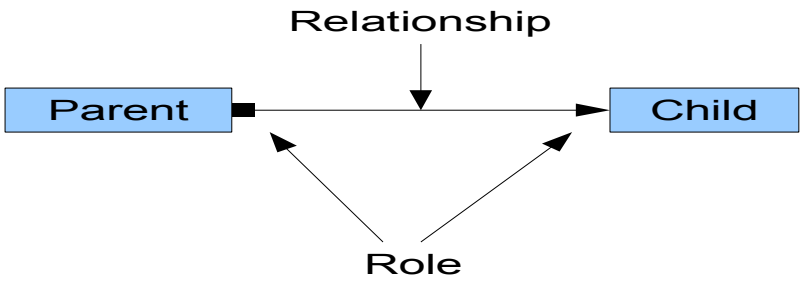

Figure 28. Roles and Relationships

On the other hand, role is a virtual concept that does not require a symbol. A role represents the part that an object is playing in a relationship. Further, it is elaborated in a conceptual diagram based on parent-child relationship in Figure 28. In this, the originative object plays the parent role and the

nodes play the child role. In the clothing shop example, two roles are used i.e. “from” and “to”. The role “from” represents the transition from the parent, and the “to” represents transition to the child.

Properties

Properties are the data values of actual grammar elements. They can be either visible or hidden in the MetaEdit+. A property can be of various permissible data types such as string, text, number, boolean etc. In the clothing shop example the objects have two type of properties: key, alternatives. The key property is used to give a name to the object. On the other hand, alternatives contain similar sounding utterances or words for the object. A property named 'level' is used in the meta-model as a hidden property to guide the code generation process. Further, the property named 'category' has been used to classify the objects. Table 4 further lists the properties used.

Property	Data type
Key	Text
Apx	Text
Category	Text
Level	Number

Table 4. Properties list

4.4 Grammar generation

“Grammars are inherently non-procedural and thus software programming and grammar writing cannot be approached in the same way [Nuance, 2002] ”.

As described earlier, the grammar development is different than the interaction/software development. Usually the VoiceXML document is built by connecting different dialogs with individual or shared grammars. The traditional form of VoiceXML development uses interaction based development. It does not consider the connection between different grammars. Instead, the code and grammar separation is practised in this thesis. It uses the external grammar linking to create a separate grammar file to hold all the grammars used. Further, this grammar file is developed using the MetaEdit+ code generation facility through the grammar model.

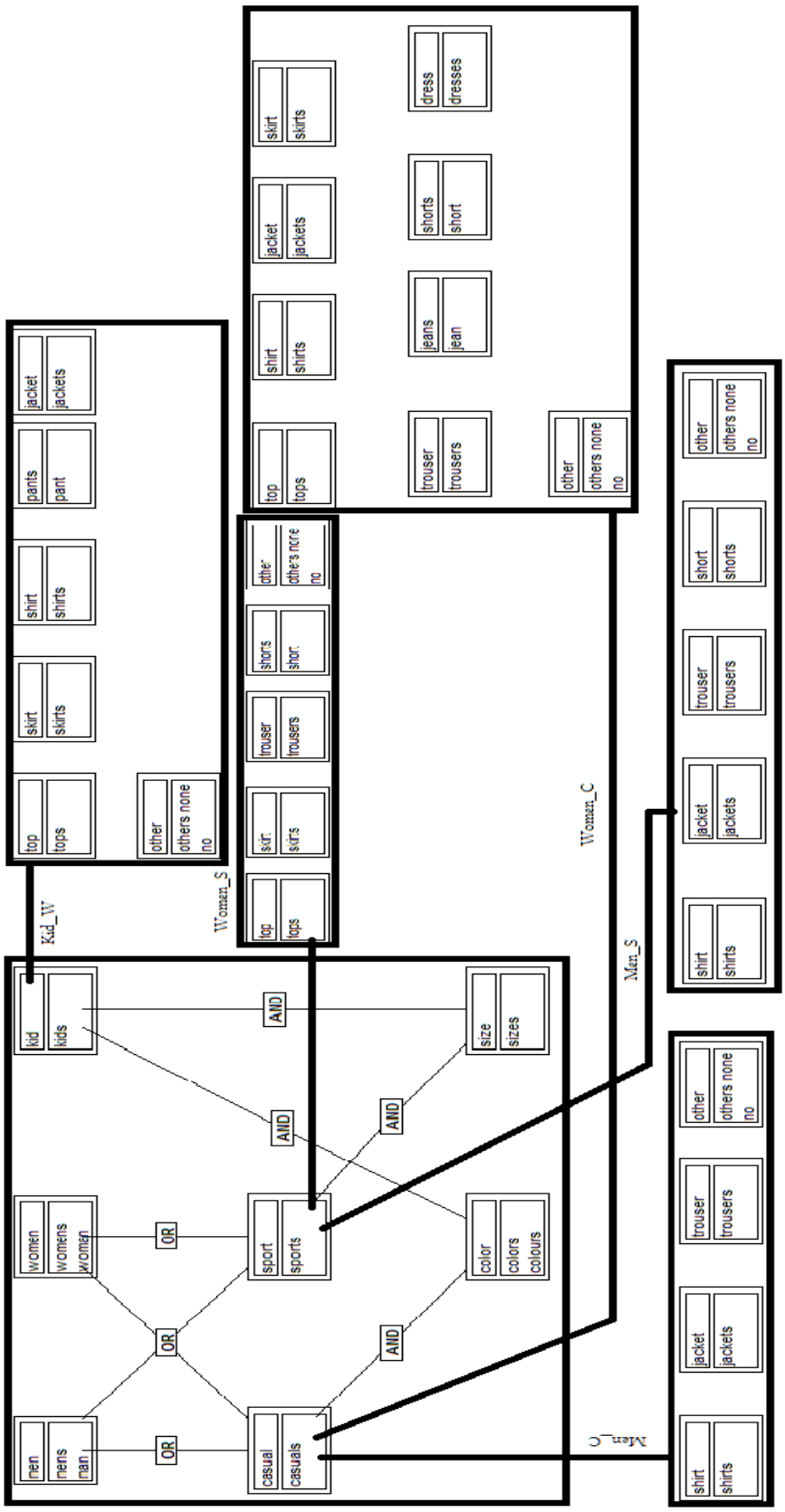

Figure 29. the grammar model with inter-grammar control flow

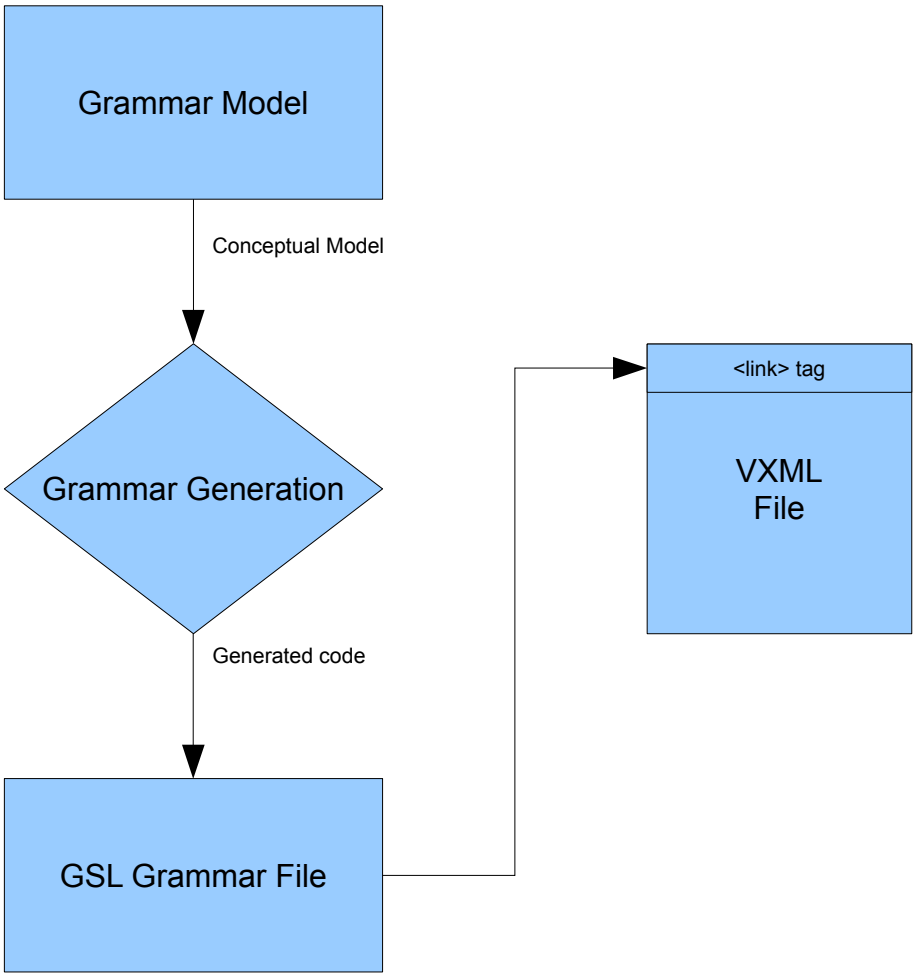

Figure 30. grammar generation process.

In Figure 30, the concept of the grammar generation process is represented. The grammar model gives a conceptual model as an output to the grammar generation facility. Further, the generated grammar file is created by traversing the objects of the grammar model. It combines the same level object information in one block. This block further creates one grammar. These grammars are further combined to make a hierarchy of grammars in a grammar file. Finally, the reference to the generated grammar file which is in this case “Grammar.gsl”, is then given to the VoiceXML file as a link. The VoiceXML file treats it as a base grammar.

Further, an excerpt of the interaction level execution is given in Figure 31. In this, the prompt is the speech from the device and the user responds to it by saying the option. However, the GSL code is generated by the grammar generation process. This is pragmatically done by code generator facility of MetaEdit+ tool. Further, a screen-shot of the grammar generation script used in making the clothing shop GSL grammar is presented in Figure 32.

Prompt: Welcome to Clothing Shop, please say Men,Women or Kids.

User: Men

Prompt: Please select from casual or sports.

User: Usual wear

Prompt: Please select from casual or sports.

User: Any

Prompt: Answer not matched. Please select from casual or sports.

User: Sports

…….

Figure 31. An excerpt of VoiceXML application execution

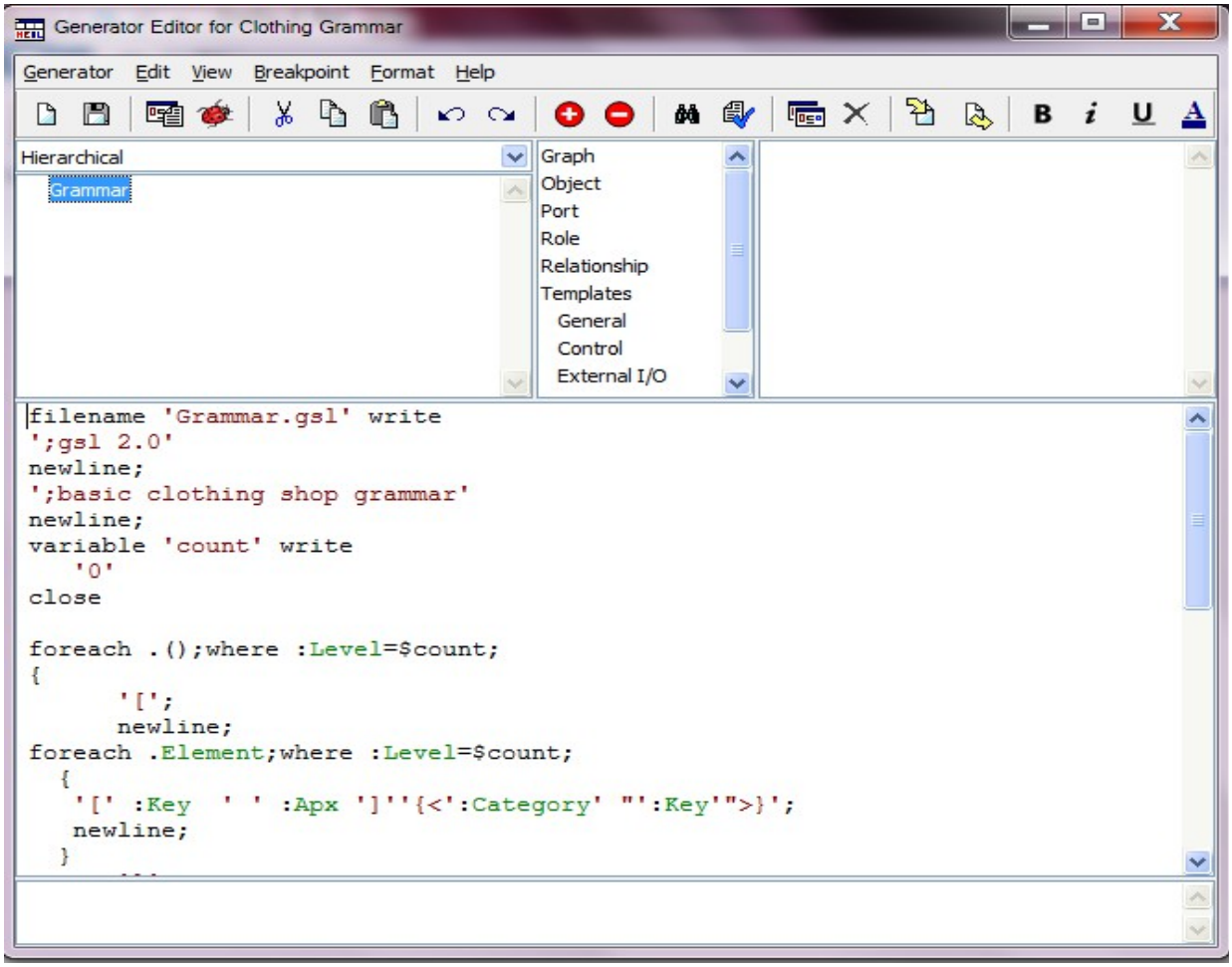

Figure 32. An excerpt from grammar generator editor tool.

As a result, MetaEdit+ seems beneficial for the developers. It creates the manipulation in the grammar model quite straightforward. The addition of a concept can be done by making a new object and by attaching it with the parent object. Furthermore, the interlinking process is governed at real-time by the MetaEdit+ on the given constraints. An existing concept can be updated or expanded by editing the object properties. Moreover, the grammar generation code is universal regardless of graph level updates. Consequently, all these facilities of manipulating the graphs make the grammar generation more flexible and swift.

In the upcoming section, couple of other automated speech grammar development approaches are reviewed, and their applications & features are compared with the VoiceXML speech grammar model.

5. Related work and discussions

5.1 Automated derivation of speech interfaces

Lahtinen et al. [2008] published a research on the speech grammar derivation using a model based approach. It uses the application model view instead of the data model view for the model development. The application model view represents a conceptual model from the user interaction viewpoint. On the other hand, data model is based on the data flow representation. Further, the approach used by Lahtinen et al. [2008] requires manual development of the interface depending on the application specific API (application programming interface). However, the grammar generation is automatic. The core concepts or facilities provided by the approach are described below.

Application model

The model is basically a structure of interrelated objects. The operations can be applicable on these objects in the form of creating/editing/removing objects and manipulating attributes. This scheme is called object configuration, and the instance of a model is called “an application model”. Further, the application model represents the current state of the application. However, the application model itself represents a conceptual form of the real application, but it may not support implementation.

Stereotypes

There is a possibility to add constraints on the operations. Further, the constraints ensure mutual integrity between classes. Some of the stereotypes are addable, removable, and alwaysActive. The addable constraint represents the element that can be added to the instance of the model. Further, the removable constraint represents that the element can be removed. Finally, the alwaysActive constraint ensures that the element remains in active state in any condition of the state of application.

Speech commands

To limit the speech input in an understandable form, the rules for input grammar have to be set.

Therefore, the simple imperative structure is used for the speech command containing command verb, target and/or qualifiers. The basic structure is,

Syntax:	*Command + Target* [*+ qualifiers*]	
Example 1:	*Add Tree*	(i.e. Command + Target)
Example 2:	*Set Color to Blue*	(i.e. Command + Target + qualifier)

Grammar generation

The algorithm that is responsible for generating the grammar automatically, traverses through the application model with extracting information from each element. The grammar is then generated on the basis of speech commands.

Speech control architecture

Figure 33 represents the speech control architecture which is based on MVC (model view control) structure. The model part consists of the application model and the state configuration of it. In the control part, the main control gets speech commands and maps it to the control API through speech recognition engine. The control API is the implementation of the target API on the application context. Further, the main control is responsible for grammar generation and controls active commands. Finally, the implementation view consists of generated grammar and the target application user interface.

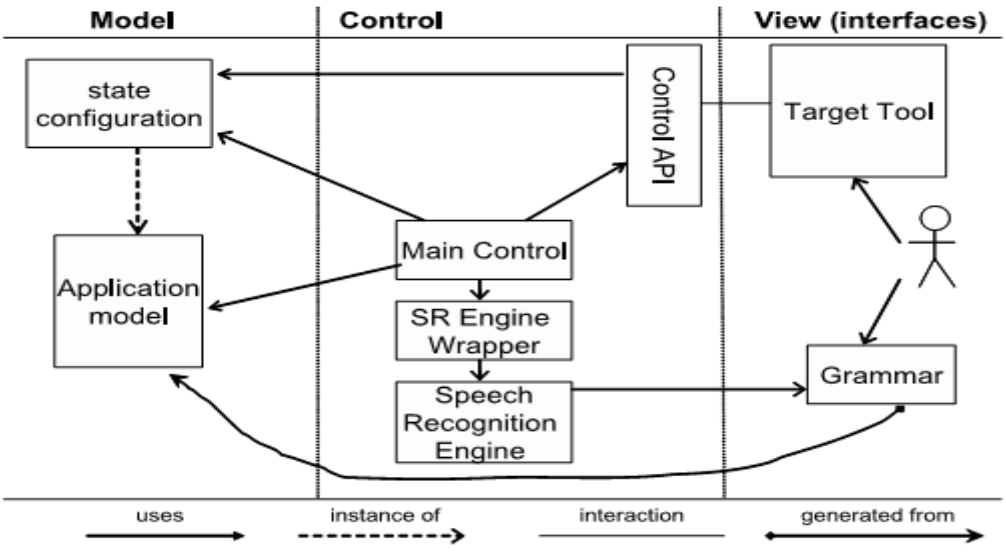

Figure 33. Speech Control Architecture[Lahtinen et al., 2008]

5.2 Automatic speech grammar generation during conceptual modelling of virtual environments

On the other hand, Vanacken et al. [2008] considered speech interaction a natural way to interact within virtual environment applications. Thus, a grammar model is developed to define the interactive virtual world using speech interaction. The grammar model is based on the conceptual modelling approach to ease the development by using higher level of abstraction. However, the development of interactive virtual environments (IVEs) is not a straightforward process because both interaction techniques and input/output devices differ from classic user interfaces and applications [Cuppens and Coninx, 2005].

However, the system is not capable of understanding general natural language but only language which is appropriate to a particular task. Further, the users should use a limited vocabulary and syntax if they need to have successful interactions with the system [Mcglashan ,1995]. Thus, analysing limited language understanding of the system, Mcglashan [1995] divided the required knowledge into ontological, linguistic and contextual knowledge. The ontological and linguistic knowledge is to be developed at the design time, and contextual knowledge is to be collected at run time from the system user. Further, the ontological knowledge contains the knowledge about the domain to model the virtual environment. For this reason, the ontological knowledge is used in this approach to represent the virtual world. Moreover, the linguistic knowledge is used for mapping between domain concepts and the virtual objects. It uses the OWL(W3C web ontology language) [OWL,2004] to formulate ontology.

Ontology

The ontology based model is a class/instance structure utilizing the object orientation base of the OWL. Further, it defines the concepts in the domain level and an instance is created on the bases of the domain rules.

In Figure 34, the domain is named “place” with properties as “structure” and “color”. Further, the instance should follow the domain rule to select the permitted values for the properties. In this case, the “structure” can have either “open” or “close” as a valid value but “color” may have any value.

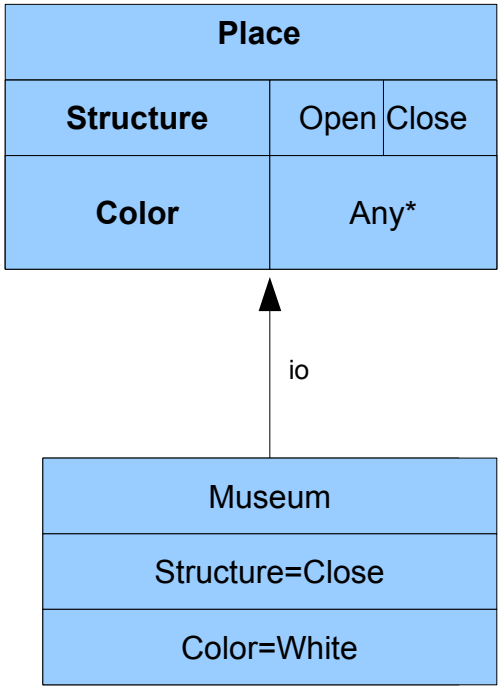

Figure 34. Excerpt of a simple ontology.

Speech grammar generation

The speech grammar generation uses an automatic speech grammar generation facility with following steps:

1. The virtual world is modelled conceptually by which semantic data is generated.
2. The semantic data is used to automatically generate a speech grammar.
3. This speech grammar is further annotated with synonyms using a lexical database of English, WordNet [WordNet, 2011].

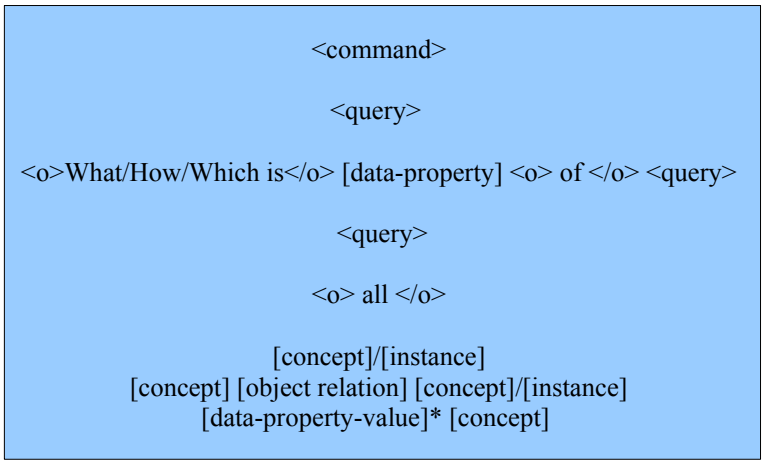

Figure 35. A speech grammar structure

In Figure 35, the speech grammar structure has been represented. It is the outcome after processing ontology data on each step. Further, the “command” tag represents the command to perform such as add, select etc. The “query” is a recognised user utterance, and “o” represents the optional tag. Furthermore, the items between “[ ]” are of semantic information modelled in the ontology (e.g. place), and the “[instance]” is the instance of that item (e.g. museum). Further, the “[concept]” stands for the class. The “[object relation]” represents the operations between objects like (e.g. is-a). Finally, the “[data property]” and the “[data property value]” represents property and its instance respectively like “color = white” as in Figure 34.

Further, Figure 36 represents the speech interaction process from the user point of view. Firstly, the speech based command comes from the user. Further, the speech command is to be recognised using speech grammar. In parallel to that, the grammar generation component uses the data from the OWL and perform various operations on it as discussed earlier. Further, the OWL-based model depends on the conceptually modelled virtual world representation and the WorldNet service for the synonyms. Further, a query is performed on the OWL base validating speech grammar. Finally, the interaction part comes in the scene for the actual interface.

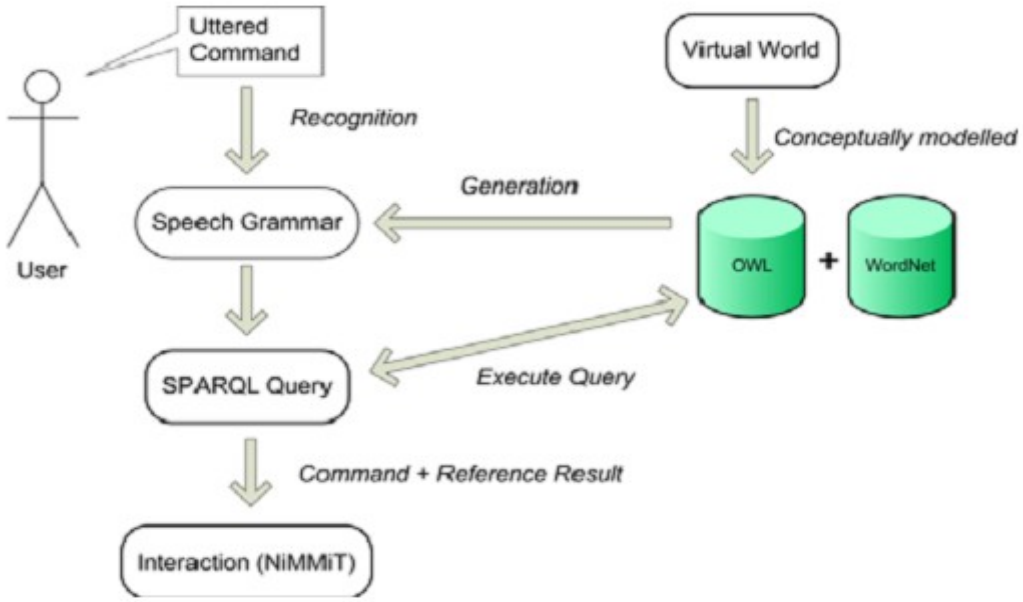

Figure 36. Speech interaction steps [Vanacken et al.,2008].

Further, in the upcoming section previously discussed speech grammar modelling approaches will be compared with the VoiceXML based speech grammar development approach.

5.3 Analysing grammar models

The speech grammar modelling approaches discussed earlier contain distinct application domains and facilities. Therefore, the comparison between this approach and other related approaches discussed in previous subsections is done here. In table 5, various aspects of each kind of approach are listed, and are elaborated later.

	VoiceXML model	**[Lahtinen et al.,2008]**	**[Vanacken et al.,2008]**
Meta-model	Custom	UML	OWL
View	Data model	Application model	Ontology
Data separation	Yes	No	-----
Grammar type	Menu based	Command based	Domain specific
Grammar Generation	Automatic	Automatic	Automatic
Links	Not used	Not used	WordNet

Table 5. comparison between different approaches.

Meta-model

A meta-model defines the rules for a model. The VoiceXML grammar model uses a free form of connections between objects. Thus, the meta-model applied on this does not contain strict constraints. However, the model of Lahtinen et al. [2008] use the UML as the meta modelling language. The models in this approach are based on mixed representation of functions/procedures and data. On the other hand, the approach used by Vanacken et al. [2008] is based on the OWL. The OWL is based on the class & instance structure. However, class & instance structure is not suitable for the required data model.

View

The view is the interaction platform for the user/developer. However, the VoiceXML model uses the data model view for the manipulations on the speech grammar model. On the other hand, Lahtinen et al. [2008] use application model view. The application model view is the implementation view for the manipulations on the model. Furthermore, Vanacken et al. [2008] use the ontology based

manipulations. Consequently, both of the later listed views do not support the requirement of the data separation.

Data separation

Further, the VoiceXML model based approach separates data from implementation code. The data in this approach is stored in the GSL grammar file attached to the implementation code with link tag. On the other hand, approaches developed by Lahtinen et al. [2008] and Vanacken et al. [2008] do not follow this concept.

Grammar type

The grammar type property is rather less valuable and it depends on the domain of the required application. However, the VoiceXML model uses the menu based approach. In menu based approach, the limited vocabulary could be created by providing options for the inputs. On the other hand, Lahtinen et al. [2008] use command based approach to limit the options. Further, Vanacken et al. [2008] use domain specific vocabulary defined by OWL.

However, the grammar generation is automatic in all three approaches compared. In addition, the Vanacken et al. [2008] approach uses external links to facilitate with synonyms for the speech input. On the other hand, the synonym finding feature is not considered as a requirement in this grammar model. As a result, the VoiceXML grammar model is found fit for the purpose. Further, the conclusion section will summarise the thesis work with features and prospects of the approach.

6. Conclusion

Consequently, the use of data model made the grammar more structured, and the MetaEdit+ tool made the manipulation work easy. Further, the development work became more effective in terms of complexity reduction. Furthermore, the comparison with [Lahtinen et al.,2008] and [Vanacken et al.,2008] signified that the model is beneficial and more flexible in terms of openness and model structure. The model based grammar development achieved the goals by improving the factors effecting development and maintenance of an application. Moreover, the separation of the interaction code/model and grammar model resulted in better understandability, rapidness, and flexibility of the speech based application development.

Understanding

A model helps in understanding a complex problem by the means of abstraction [Selic, 2003]. Accordingly, it is easier to understand than the code. However, mixed models are used by [Lahtinen et al.,2008] and [Vanacken et al.,2008]. The understanding can be evaluated by using cognitive dimensions framework [Green and Petre, 1996]. It contains abstraction gradient, mapping closeness, diffuseness, hidden dependencies, and viscosity. The abstraction gradient deals with the level of abstraction available. Mapping closeness checks the closeness between the model and the real world. Diffuseness is the factor related to mapping closeness. It focuses on the reduction of terseness. Hidden dependencies are the relationships not visible in the model and the viscosity is the resistance to the local change (e.g. change between neighbouring entities). The comparison is shown in table 6.

	VoiceXML model	**[Lahtinen et al.,2008]**	**[Vanacken et al.,2008]**
Abstraction gradient	High	Low	High
Mapping closeness	Yes	Yes	Yes
Diffuseness	Low	High	Low
Hidden dependencies	Yes	No	No
Viscosity	Yes	No	Yes

Table 6. Comparison based on cognitive dimensions.

Vanacken et al. [2008] use ontology based model for the speech grammar. This parent-child relationship based model uses single level abstraction. On the other hand, Lahtinen et al. [2008] use application model. The abstraction level of it is low due to mixture of data and code. However, the VoiceXML based model developed in this thesis uses sub-graphs based abstraction. Thus multiple levels can be implemented. The mapping closeness in the Vanacken et al. [2008] is close to real world due to the use of ontology based model. Further, Lahtinen et al. [2008] used application model which can be implemented using UML. Hence, it seems close to the real world process. Moreover, VoiceXML based model is also close to the real world because of the dialogue flow based structure.

However, diffuseness in the VoiceXML based model is less due to its simple design. Moreover, the ontology based implementation used by Vanacken et al. [2008] is also less diffused. However, the application model approach of Lahtinen et al. [2008] is more diffused. The hidden dependencies are present in the VoiceXML based model due to the use of sub-graphs. On the other hand, approaches used by Vanacken et al. [2008] and Lahtinen et al. [2008] do not contain hidden dependencies. The VoiceXML based model is controlled by meta-model specification which enables viscosity. Moreover, the ontology based model used by Vanacken et al. [2008] also maintains viscosity due to the class-object relationships. However, Lahtinen et al. [2008] model do not require controlled approach.

Rapidness

The model based development of voice control applications is rapid than other systems as [Tolvanen and Kelly, 2009] describe in figure 25. The viscosity dimension of model helps in achieving speed of development by allowing local changes. Further, the MetaEdit+ tool governs the consistency of the change applied. Moreover, the code generation facility generates the GSL file to link with the implementation code.

Flexibility

The use of the flexible data model design and the grammar format made the model easy to fit with any kind of domain by performing minor changes in the model level and/or grammatical level. Figure 37. shows a simple 6 grammar based graph. It uses only two type of relationships, “AND”

and “OR”. The “AND” relationship allows an entity to connect with 2 or more entities at one time. On the other hand “OR” connection works on option based criteria. As a result, one can add new entity anywhere in the model by just selecting one of these two relationships. Consequently, this made the approach fit for an extensive set of applications in comparison with the other approaches.

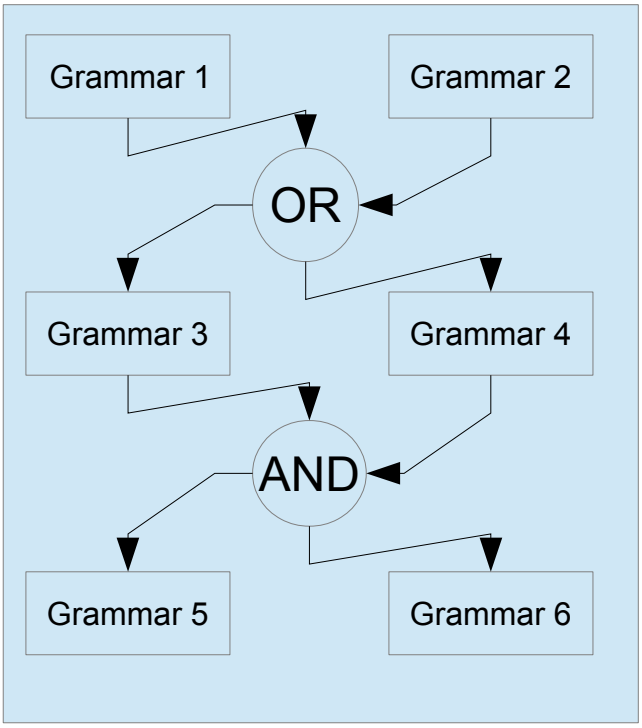

Figure 37. A graph repenting the relationship types allowed.

Future directions

The development of speech based applications using the VoiceXML is common development practice to achieve speed, and by using Voxeo designer [2011] it becomes even more faster. Since the Voxeo designer does not utilize grammar models, projects with large grammars may suffer. A solution can be the use of custom developed IDE(integrated development environment) for VoiceXML development that may use dual view architecture. One view may be used for VoiceXML interaction code model, and other can be used for grammar model instead of mixing both concepts.

7. References

[Anusuya and Katti, 2009] M. A. Anusuya and S. K. Katti, Speech Recognition by Machine: A Review. *International Journal of Computer Science and Information Security,* Vol. **6**, No. 3, 2009.

[Atkinson and Kühne,2003] C. Atkinson and T. Kuhne, Model-driven development: a metamodeling foundation. University of Mannheim, Software, *IEEE*, Issue Date: Sept.-Oct. 2003,Volume:**20**, Issue: 5, On page(s):36 – 41, ISSN:0740-7459.

[Bell, 1998] R. Bell, Code generation from object models. *Embedded system programming*, 1998, **11**(3):23–33

[Chen,1976] P. Chen, The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*1, 1 (March 1976), 9-36.

[Codd, 1990] E. F. Codd, *The Relational Model for Database Management: Version 2*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Cuppens and Coninx, 2005] E. Cuppens and K. Coninx, CoGenIVE: Code Generation for Interactive Virtual Environments. Limburgs Universitair Centrum - Expertise Centre for Digital Media, Universitaire Campus, B3590 Diepenbeek, Belgium.

[Furui, 2005] S. Furui, 50 years of Progress in speech and Speaker Recognition Research. *ECTI Transactions on Computer and Information Technology*, Vol.**1**. No.2 November 2005.

[Green and Petre, 1996] T. Green and M. Petre, Usability analysis of visual programming environments: A 'cognitive dimensions' framework, *J. Vis. Lang. Comput.*, **7**, 2, pp. 131–174.

[Hemel et. al., 2008] Z. Hemel, L. C. L. Kats and E. Visser, Code Generation by Model Transformation: A Case Study in Transformation Modularity. Theory and practice of model transformations, *lecture notes in computer science,* 2008, volume **5063/2008**, 183-198.

[Kelly et al., 1996] S. Kelly, K. Lyytinen and M. Rossi, MetaEdit+ A fully configurable multi-user and multi-tool CASE and CAME environment, Lecture Notes in Computer Science, 1996, Volume 1080/1996.

[Kelly, 1997] S. Kelly, Appendix 1 in PhD dissertation,1997. Available as http://metaphor.it.jyu.fi/a1goprr.html.

[Kernighan et al., 1990] Mark D. Kemighan, Kenneth W. Church and William A. Gale, A spelling correction program based on a Noisy channel model. Proceedings of *COLING1990*, pp.205-210.

[Kurtev et al., 2002] I. Kurtev, J. Bézivin and M. Aksit, Technological Spaces: an Initial Appraisal. Software Engineering Group (TRESE), University of Twente, The Netherlands.

[Kühne, 2006] T. Kühne, Matters of (Meta-) Modeling. Darmstadt University of Technology, Darmstadt, Germany. In: *the Journal on Software and Systems Modeling*, Volume **5**, Number 4, pp. 369-385, December 2006.

[Lahtinen et al., 2008] Samuel Lahtinen, Heikki Suontausta and Kai Koskimies, Automated Derivation of Speech Interfaces: A Model-Based Approach. aswec, pp.289-299. In: *19th Australian Conference on Software Engineering.*

[Larson, 2003] J. A. Larson, VoiceXML and the W3C Speech Interface Framework. Intel Corporation, 2003, *IEEE*.

[Lowrre, 1990] B. Lowrre, *The HARPY speech understanding system.* Trends in Speech Recognition. Ed., Speech Science Pub., pp.576-586, 1990.

[Mcglashan, 1995] S. Mcglashan, Speech Interfaces to Virtual Reality. *Proceedings of 2nd International Workshop on Military Applications of Synthetic Environments and Virtual Reality*.

[MDA, 2011] Also available as http://www.omg.org/mda .

[Navathe,1992] S. Navathe, Evolution Of Data Modeling for Databases. September 1992/Vol.**35**, No.9, *COMMUNICATIONS OF THE ACM.*

[Nuance,2002] Nuance Speech Recognition System, Version 7.0, Nuance Grammar Developer's Guide, Copyright © 1996-2001 Nuance Communications.

[OWL, 2004] OWL Web Ontology Language Overview,W3C Recommendation 10 February 2004. Also available as http://www.w3.org/TR/owl-features .

[Schmid and Swenson, 1975] H. A. Schmid and J. R. Swenson, On the semantics of the relational data model. Department of Computer Science, University of Toronto.

[Selic, 2003] B. Selic, The Pragmatics of Model-Driven Development, IBM Rational Software. *IEEE Computer Society*, 2003.

[Sendall and Kozaczynski, 2003] S. Sendall, W. Kozaczynski, Model transformation: the heart and soul of model-driven software development," *Software, IEEE* **20**, 5, 42- 45.

[Shneiderman, 2000] B. Shneiderman, The limits of speech recognition. *ACM* **43**, 9 (September 2000), 63-65.

[SRGS, 2004] Speech Recognition Grammar Specification Version 1.0. W3C Recommendation, 16 March 2004.

[Su and Lee, 1994] K. Su and C. Lee, Speech Recognition using weighted HMM and subspace Projection Approaches. *IEEE Transactions on Audio,* Speech and Language, 1994.

[Taylor and Frank,1976] R. W. Taylor and R. L. Frank, CODASYL database management systems. *ACM Computing Surveys.*

[Tolvanen and Kelly, 2009] J. Tolvanen and S. Kelly, MetaEdit+: defining and using integrated domain-specific modeling languages. In: *Proceeding of the 24th ACM SIGPLAN conference*

companion on Object oriented programming systems languages and applications(OOPSLA '09). ACM, New York, NY, USA, 819-820.

[Tsichritzis and lochovsky, 1976] D. C. Tsichritzis and F. H. Lochovsky, Hierarchical Data-Base Management: A Survey. *ACM Comput. Surv.***8**, 1 (March 1976), 105-123.

[Vanacken et al.,2008] Lode Vanacken,Chris Raymaekers and Karin Coninx, Automatic Speech Grammar generation during conceptual modelling of Virtual Environments, Hasselt University, Expertise Centre for Digital Media and transnationale Universiteit Limburg Wetenschapspark 2, B-3590 Diepenbeek, BELGIUM.

[VoiceXML, 2011] Also available as http://www.voicexmlforum.org .

[Voxeo Designer, 2011] Also available as http://designer.voxeo.com/Designer/WebHelp/102-introduction.htm.

[VXML2,2001] Voice Extensible Markup Language (VoiceXML) Version 2.0,W3C Working Draft, 23 October 2001.

[Windmann and Haeb-Umbach, 2009] S. Windmann and R. Haeb-Umbach, Approaches to Iterative Speech Feature Enhancement and Recognition. *IEEE Transactions on audio, speech, and Language processing*, Vol. **17**, No. 5, July 2009.

[WordNet, 2011] Also available as http://wordnet.princeton.edu .

[W3C-Speech, 2000] Introduction and Overview of W3C Speech Interface Framework, W3C Working Draft, 4 December 2000.

[W3C,2011] Also available as http://www.w3.org .

[W3C-Voice] Also available as http://www.w3.org/voice .

Appendix A

Grammar generator script:

```
filename 'Grammar.gsl' write
';gsl 2.0'
newline;
';basic clothing shop grammar'
newline;
variable 'count' write
  '0'
close

foreach .();where :Level=$count;
{
          '[';
                newline;
foreach .Element;where :Level=$count;
 {
        '[' :Key  ' ' :Apx ']"{<':Category' '":Key'">}';
        newline;
 }
                ']';
 to '%null' newline '* $' endto
 $count++%null;
 newline;
}
foreach .Element;
 {
        do explosions
  {
                newline;
          '[';
                foreach .Element;
                        {
                        newline;
                        '[' :Key  ' ' :Apx ']"{<':Category' '":Key'">}';
                        }
                newline;
                ']';
  }
 }

close
```

Appendix B

Generated grammar file:

```
;gsl 2.0
;basic clothing shop grammar
[
[kid kids]'{<ch_main "kid">}
[men mens man]'{<ch_main "men">}
[women womens woman]'{<ch_main "women">}
]
[
[casual casuals]'{<cs "casual">}
[sport sports]'{<cs "sport">}
]
[
[color colors colours colour]'{<prompt "color">}
[size sizes]'{<prompt "size">}
]

[
[jacket jackets]'{<mc "jacket">}
[other others none no]'{<mc "other">}
[shirt shirts]'{<mc "shirt">}
[trouser trousers]'{<mc "trouser">}
]
[
[dress dresses]'{<wc "dress">}
[jacket jackets]'{<wc "jacket">}
[jeans jean]'{<wc "jeans">}
[other others none no]'{<wc "other">}
[shirt shirts]'{<wc "shirt">}
[shorts short]'{<wc "shorts">}
[skirt skirts]'{<wc "skirt">}
[top tops]'{<wc "top">}
[trouser trousers]'{<wc "trouser">}
]
```

```
[
[blue ]'{<color "blue">}
[green ]'{<color "green">}
[red ]'{<color "red">}
]
[
[jacket jackets]'{<kw "jacket">}
[other others none no]'{<kw "other">}
[pants pant]'{<kw "pants">}
[shirt shirts]'{<kw "shirt">}
[skirt skirts]'{<kw "skirt">}
[top tops]'{<kw "top">}
]
[
[l ]'{<size "l">}
[m ]'{<size "m">}
[s ]'{<size "s">}
[xl ]'{<size "xl">}
[xxl ]'{<size "xxl">}
]
[
[jacket jackets]'{<ms "jacket">}
[other others none no]'{<ms "other">}
[shirt shirts]'{<ms "shirt">}
[short shorts]'{<ms "short">}
[trouser trousers]'{<ms "trouser">}
]
[
[other others none no]'{<ws "other">}
[shorts short]'{<ws "shorts">}
[skirt skirts]'{<ws "skirt">}
[top tops]'{<ws "top">}
[trouser trousers]'{<ws "trouser">}
]
```