

Agilingua, LLC.

Development Process Using ATOM SDK

ATOM Spoken Dialogue SDK

Table of Contents

Outline	3
Step 1 :Data Collection	5
Step 2:Transcription	6
Step 3:Grammar Development	7
Step 4:Dialogue Development	9
Step 5 :Usability Test	10
About Grammar Development	11
Integration Program	14

Outline

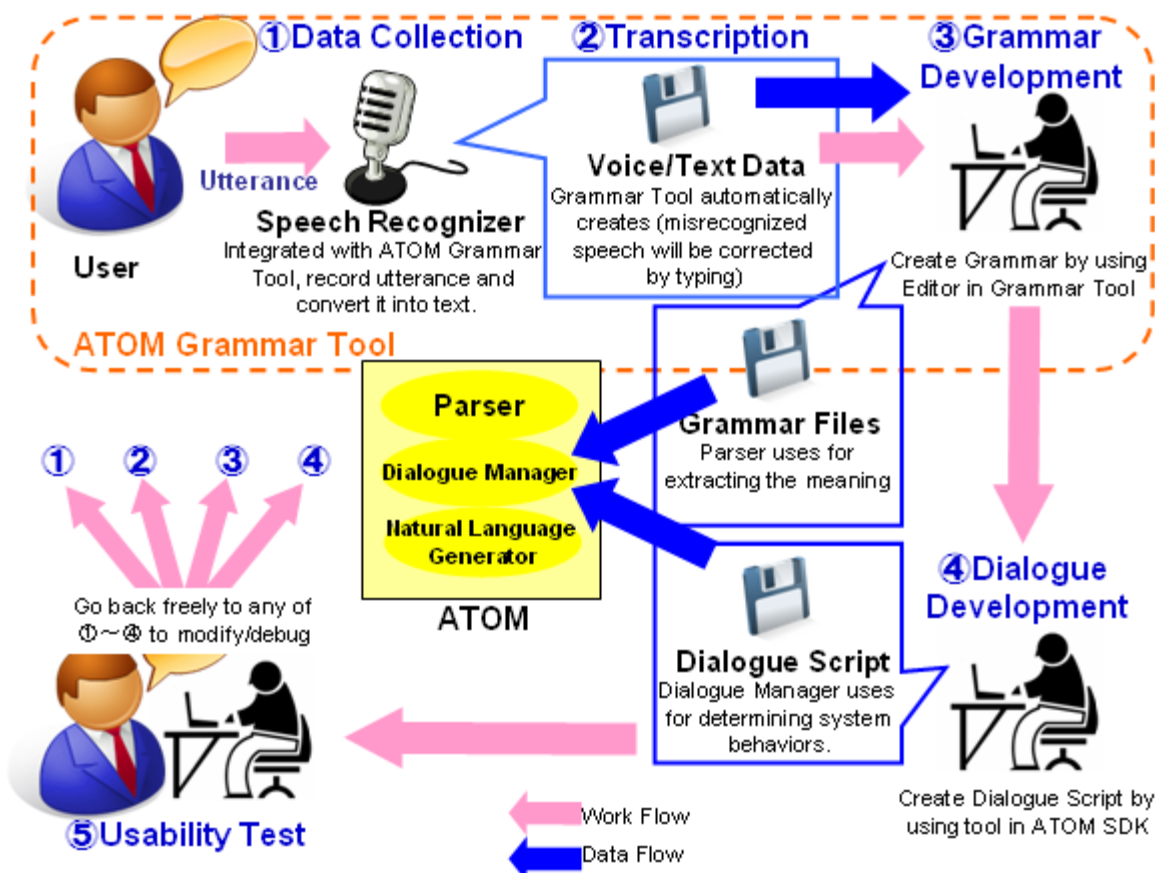
The central feature of the ATOM Spoken Dialogue SDK is CDD (Comprehensive Dialogue Development Process) which for the first time provides integrated support for the entire development cycle for spoken language applications, from data collection to usability testing.

CDD allows highly efficient, integrated development. Making changes to implementations after testing can be done quickly and efficiently to reduce the overall development time.

■ ATOM Spoken Dialogue SDK Development Process

A dialogue system development will usually need the below 5 processes; ①Data Collection, ②Transcription ③Grammar Development, ④Dialogue Development, and ⑤Usability Test.

CDD supports an entire development procedure including the 5 processes. The below illustration shows how..



* Orange dotted line in the illustration indicates the coverage of ATOM Grammar Tool, which includes process 1: Data Collection, 2: Transcription, and 3: Grammar Development. ATOM Grammar Tool is included in all SDKs but also sold singularly.

Step 1 Data Collection

Data collection is the process of collecting spoken natural language data for the purpose of system development, testing and tuning.

Step 2 Transcription

Transcription is the process of converting into typewritten form a spoken language source, such as recorded speech.

Step3 Grammar Development

Grammar development is the process of specifying the words and patterns of words to be listened for by a speech recognizer.

Step 4 Dialogue Development

Dialogue development is the process of specifying the actions an interaction manager should take.

Step 5 Usability Test

Usability testing and tuning is an iterative process of incremental refinement of a spoken dialogue application.

Step 1: Data Collection

Data collection is the process of collecting spoken natural language data for the purpose of system development, testing and tuning. In other words, a developer expects what a user may speak to the system when the system is completed and records expected user speech as much as possible. Since data collection needs to be done at the beginning of the development process, no working system is available. Therefore, unbeknownst to the user, a human simulates the behavior of the dialogue system (Wizard of Oz simulation, named after the book *The Wizard of Oz* in which the Wizard was just a simulation controlled by a man behind a curtain).

- Tools for Data Collection

For data collection, two networked computers are needed. The data collection software included in the SDK is installed on the computer used for recording speech. The wizard can interact with the user by listening to the users' recorded speech or by generating utterances through a text-to-speech system. The data collection software acts as a web server, therefore, all interactions from the wizards' side are done through a web browser.

- Results of the Data Collection Process:

The result of Step 1 is a corpus of interactions between users and wizards. The interactions are logged and time-stamped. The log files are stored in XML format and can be viewed using standard web browsers. The users' speech is recorded and referenced from the log files. The recorded speech can be used to train speech recognizers or, after transcription, to develop grammars.

- Supported Standards and APIs for Transcription:

EMMA

EMMA (Extensible MultiModal Annotation markup language) is used to represent the input events from the user.

SAPI 4.0 and 5.1

SAPI 4.0 or SAPI 5.1 compliant speech synthesis engines can be used to synthesize text from the wizard.

Step 2: Transcription

Transcription is the process of converting into typewritten form a spoken language source, such as recorded speech. Transcribed speech, that is, the sound recordings together with its transcriptions, can be used to train the acoustic models of speech recognizers. Furthermore, transcriptions alone can be used to train language models for speech recognizers or to develop grammars for speech recognizers and spoken language understanding.

- Tools for Transcription

The transcription tool lets transcribers annotate recorded sound files. The annotations are inserted in the log files.

- Results of Transcription:

The result of Step 2 is a corpus of interactions between users and wizards in which the sound recordings are transcribed. The transcriptions are inserted in the log files. The transcribed log files contain all the data needed to develop a spoken dialogue system and are used by other tools for grammar development (step 3), dialoguedevelopment (step 4) and usability testing (step 5).

- Supported Standards and APIs for Transcription:

EMMA

EMMA (Extensible MultiModal Annotation markup language) is used to represent the speech events from the user collected in step 1. The transcription tool inserts the transcriptions directly underneath the <emma:interpretation> tags.

Step 3: Grammar Development

Grammar development is the process of specifying the words and patterns of words to be listened for by a speech recognizer. Furthermore, together with semantic annotations, grammars are used by the natural language understanding unit to convert the recognized text into a semantic representation of what has been said.

Grammars can be specified in a variety of interchangeable formats. The commonality among all formats is that they represent a Context Free Grammar. The ATOM Spoken Dialogue SDK supports several grammar formats.

Grammar writing is an iterative process in which a grammar is matched against a set of collected natural language utterances. In cases where there is no match or a partial match, the grammar rules are refined until the grammar matches all collected utterances. Please refer page Grammar Development (page 10) for technical details of grammar development.

- Tools for Grammar Development:

The tools provided in the voice toolkit allow an application developer to extract the utterances from the log files collected in step 1. It is possible to get word count statistics from the log files, test grammars against a set of log files and get coverage statistics. Grammars may be annotated semantically to convert the recognized text into semantic representations.

- Results of Grammar Development:

The result of the grammar development process is one or more grammar files. Grammar files can be referenced from a dialogue script in step 4. It is also possible to convert the developed grammars into N-Gram models for speech recognition.

- Standards and Proprietary Extensions

Supported Standards and APIs for Grammar Development:

- ECMAScript Compact Framework: ECMAScript Compact Framework is used to specify the semantic interpretation parts of the grammars.

- JSGF: JSGF (Java Speech Grammar Format) is one of the formats in which grammars can be represented.

- SISR : SISR (Semantic Interpretation for Speech Recognition) is the format in which grammars can be annotated.

- SRGS : SRGS (Speech Recognition Grammar Specification) is one of the formats in which grammars can be represented. Both the XML and the ABNF formats are supported.

- Proprietary Extensions:

The proprietary extensions allow to combine context free grammars with representations from the semantic web standard RDFS.

- Typed Grammars

The non-terminals of typed grammars can be annotated with RDFS concepts. It is then possible to do compile-time checking of the grammar specifications. This process can be compared to verifying XML markup with the help of an XML schema or a DTD.

Step 4: Dialogue Development

Dialogue development is the process of specifying the actions an interaction manager should take. Actions can be specified as response to speech recognition input, timeouts, low confidence values or more.

- Tools for Dialogue Development:

The tools provided in the voice toolkit allow an application developer to write dialogue scripts in an extended version of ECMAScript. The extensions allow the application developer to specify conditions and actions the interaction manager should undertake, as well as to reference grammars in a context-dependent way.

Integrated with the interaction manager is an SQL compatible database. Semantic interpretations from the recognition and natural language understanding process can be converted easily to SQL queries.

- Results of Dialogue Development Development:

The result of the dialogue development process consists of one or more dialogue script files. Dialogue script files and grammar files can be packaged for modular application development.

- Supported Standards and APIs for Dialogue Development:

- ECMAScript Compact Framework : ECMAScript Compact Framework is used to specify the dialogue processing algorithms. Extensions to ECMAScript are used to specify conditions and actions.

- SAPI 4.0 and 5.1 : SAPI 4.0 or SAPI 5.1 compliant speech synthesis engines can be used to synthesize text generated by the interaction manager.

- Support for SQL Databases : SQL databases can be directly accessed from the interaction manager. The retrieved datasets can be converted to text and presented to the user.

- Proprietary Extensions

The proprietary extensions allow the application developer to match conditions against the output of the semantic interpretation process. The conditions and actions are specified using proprietary extensions in ECMAScript.

Step 5: Usability Testing

Usability testing and tuning is an iterative process of incremental refinement of a spoken dialogue application. The goal is to improve the specifications created in step 3 and step 4. During usability testing, users interact with the spoken dialogue system as if it was already deployed. All actions are logged.

- Tools for Usability Testing

Tools supplied with the Voice SDK allow to annotate the log so that problematic interactions can be identified by the application developers.

- Results of the Usability Testing Process

The result of the usability testing consists of a set of log files. The log files are similar to those generated in the data collection phase except that transcriptions from the speech recognizer are included. In addition, annotations identify those interactions that could not be handled correctly.

- Supported Standards and APIs for Transcription:

- EMMA : EMMA (Extensible MultiModal Annotation markup language) is used to represent the input events from the user.

About Grammar Development

- Grammars for Speech Recognition and Natural Language Understanding

A grammar is a way of telling a speech recognizer what the user can be expected to say. As an example, imagine a voice operated video recorder. You would expect the video recorder to understand user commands such as: "I want you to record 'Ugly Betty' tonight", "Can you record tonight's 'Ugly Betty'"

- Grammar Formats

Grammar are organized in rules. You can think of a grammar rule as a list of phrases. The above example can be represented in a grammar rule like this:

```
<RecordShow> = I want you to record 'Ugly Betty' tonight  
              | Can you record tonight's 'Ugly Betty'  
              ;
```

In this example, the vertical bar | stands for alternative, meaning that the user may say one of the two phrases.

Rule References

For a useful video recorder application, you would want to record more than one show. To do so, you may introduce a new rule <ShowName> that contains a list of all shows and reference the new rule from <RecordShow> like this:

```
<RecordShow> = I want you to record <ShowName> tonight  
              | Can you record tonight's <ShowName>  
              ;
```

```
<ShowName> = Ugly Betty  
            | Grey's Anatomy  
            | Two and a half Men  
            | ...  
            ;
```

- Stable Grammars

Different persons have different ways of saying the same thing. Therefore, a major part of developing robust speech applications consists of extending grammars so that more variants of the same commands are correctly recognized. For example, we can make the above grammar more stable by also allowing the command I would like you to record... as follows:

```
<RecordShow> = I <Want> you to record <ShowName> tonight  
              | Can you record tonight's <ShowName>
```

;

```
<ShowName> = Ugly Betty
```

```
            | Grey's Anatomy
```

```
            | Two and a half Men
```

```
            | ...
```

;

```
<Want> = want
```

```
        | would like
```

;

- Adding Commands

It is possible to add more commands to the grammar. For example, we can cause the video recorder to tell us the time with the following grammar rule:

```
<TellTime> = I <Want> to know the time
```

```
            | I <Want> you to tell me the time
```

;

Note that we are referencing the rule <Want> so that the user may say I want to know the time as well as I would like to know the time .

- Tuning Grammars

While testing the video recorder application, we found that one test user actually said I want to ask you to record 'Ugly Betty'.. To accommodate this utterance, we change the rule <Want> as follows:

```
<Want> = want
      | would like
      | want to ask
;
```

However, this introduces a problem. While it is now possible to say I want to ask you to record 'Ugly Betty', it is also possible to say I want to ask to know the time. since the rule <Want> is referenced from the rule <TellTime> as well.

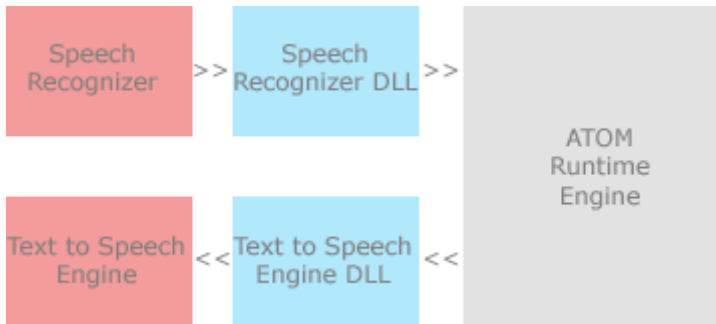
The ATOM Grammar Tool lets you detect and correct problems like this easily and thus speeds up grammar development.

Note: The grammar format in the examples in this page has been simplified for expository reasons.

Integration Program

The ATOM Runtime Engine is already integrated with SAPI 5.1 compliant speech recognition and text-to-speech engines. In addition, it is possible to integrate the ATOM Runtime Engine with third party speech recognition and text-to-speech engines, provided the engines offer a C or C++ API.

In order to integrate the ATOM Runtime Engine with a third party engine, the ATOM API calls need to be translated to the engines' API calls and vice versa. The program code responsible for the translation needs to be provided as a Dynamic Link Library (DLL). Our SDKs contain an example project illustrating how such an integration can be implemented.



The architecture diagram shows how DLLs are used to integrate third party speech recognition and text-to-speech engines with the ATOM Runtime Engine.

- Integration with Speech Recognition Engines

The DLL responsible for integrating ATOM with a third party speech recognizer needs to support functionality to:

- * Create and destroy instances of the speech recognizer
- * Load grammars dynamically at runtime
- * Activate and deactivate the recognizer
- * Send events from the recognizer to the ATOM Runtime Engine
- * Provide recognition results to ATOM, including n-best lists and confidence scores

- Integration with Text-To-Speech Engines

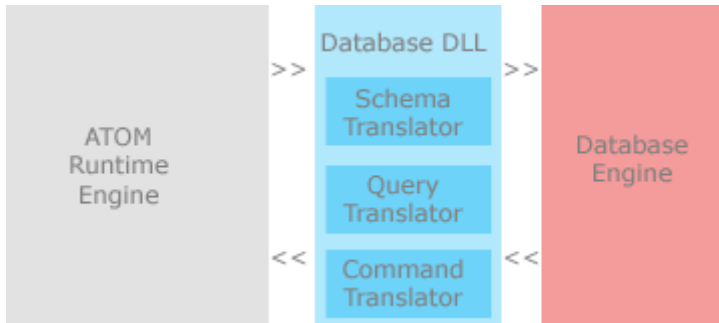
The DLL responsible for integrating ATOM with a third party text-to-speech engine needs to support functionality to:

- * Create and destroy instances of the text-to-speech engine
- * Activate and deactivate the text-to-speech engine
- * Send events from the text-to-speech engine to the ATOM Runtime Engine

- With Database Engines

The ATOM Runtime Engine is already integrated with MS Access and SQLite databases. In addition, it is possible to integrate the ATOM Runtime Engine with third party database engines, provided the engines offer a C or C++ API.

In order to integrate the ATOM Runtime Engine with a third party engine, the ATOM API calls need to be translated to the engines' API calls and vice versa. The program code responsible for the translation needs to be provided as a Dynamic Link Library (DLL). Our SDKs contain an example project illustrating how such an integration can be implemented.



The architecture diagram shows how DLLs are used to integrate third party database engines with the ATOM Runtime Engine. The query, schema and command translators are components used to translate calls from the ATOM API to the database API and vice versa.

The DLL responsible for integrating ATOM with a third party speech recognizer needs to support functionality to:

- * Create and destroy instances of the database engine
- * Execute queries, insertion and deletion commands and generate schemas.

Agilingua, LLC.

24-65 38th Street #5D Astoria, NY 11103 USA

EMAIL: info@agilingua.com

WEBSITE : www.agilingua.com

FAQ: <http://www.agilingua.com/jp/qa/overview.php>

Agilingua, LLC. All rights Reserved. 2004-2010

Revised August 15, 2010

###