# Cassie Can Speak: A .NET Interface to a Robotic Fevahr

Michael Kandefer
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260
201 Bell Hall
Buffalo, NY 14260-2000
mwk3@cse.buffalo.edu

8th May 2009

# Contents

# 1 Introduction

The SNePS Fevahr, usually called "Cassie", is an embodied agent that is capable of natural language (NL) interaction with human participants and acting. Cassie has had numerous embodiments through the years, sometimes as a hardware robot, but mostly in simulated or text-based environments. In "FevahrCassie: A Description and Notes for Building FevahrCassie-Like Agents" [7], Shapiro describes how to build Fevahr agents using the SNePS [6] knowledge representation and reasoning (KR) system using an ASCII environment. To further this work this paper shows a plausible way of implementing Cassie in a WoWee™Rovio™[11] that takes advantage of existing technologies, in particular the speech recognition and production facilities of *Microsoft's .NET Framework* [4]. Focus is placed on the process of vocalizing a response from the Rovio that was produced using the Fevahr's Lisp ATN grammar generation, the connection between that generator and the .NET language, and the audio production facilities of .NET.

# 2 GLAIR

In order to develop the Rovio-embodied Fevahr agent, hereafter "Rovio-Fevahr", a Grounded Layered Architecture for Integrated Reasoning (GLAIR) [2, 3] was employed. This architecture is split into three layers:

**The Knowledge Layer (KL)** contains what is required for conscious reasoning. The agent's propositional belief base, high-level plans, and reasoning system are all contained in this level.

**The Perceptuo-Motor Layer (PML)** serves three primary functions: (1. PMLa) Defining the primitive actions available to the agent (2. PMLb) Establishing a communication layer between PMLa and PMLc and (3. PMLc) Interacting with the agent's embodiment—its sensors and actuators.

**The Sensory Actuator Layer (SAL) and Environment** contains the implementation of the agent's embodiment (hardware or software simulation) and, if needed, the implementation of a simulated environment.

The following sections will make reference to these layers, and discuss their implementation and use.

# 3 SAL: The Rovio

The Rovio [11] (Fig. 1)[1] serves as part of the SAL in the GLAIR architecture (i.e., the embodiment) with the environment serving as the rest. The Rovio is a hardware robot that was developed by WoWee as a mobile webcam that is controlled through a WIFI connection. It has the following sensors:

[1]Image source: `http://www.telepresenceoptions.com/images/rovio.JPG`.

Figure 1: The WowWee Rovio

- Head-mounted VGA camera,
- Head-mounted microphone,
- Head-mounted IR Sensor (location detection),
- Chest-mounted IR Sensor (collision detection),
- WIFI signal strength monitor, and
- Battery charge monitor;

And actuators:

- Neck control (for angling the camera),
- Speaker,
- Three omni-directional wheels, and
- Chest-mounted light source.

Though developed for the purposes of communicating over long distances with family, the Rovio's API [9] is open source and can be utilized to program cognitive robots. Since the Rovio operates over a wireless network and the packaged client for operating the Rovio is web-based, the API is a series of HTTP calls to a web server on the Rovio itself. Thus, to program a Rovio to sense and act "autonomously", one only requires a programming language that can established a TCP/IP connection to a web server, and that can send

HTTP requests and receive the data from these requests.[2] For example, to acquire a report about the Rovio's status (with the exception of camera and microphone data) the following HTTP URI is sent through an established TCP/IP connection:

```
http://192.168.1.2/rev.cgi?Cmd=nav&action=1
```

Here *192.168.1.2* is the IP address of the Rovio, and when sent across a TCP/IP connection will result in a HTTP response with the sensor data. Commands like these are used for developing the PMLc functionality of the Rovio-Fevahr, and will be discussed as needed in the following sections.

# 4 PMLc: Rovio Can Speak

For the Rovio-Fevahr the PMLc was written in C# .NET. The choice of C# is irrelevant as all of the .NET languages have access to the same underlying libraries, but is useful as it uses a syntax similar to popular object-oriented programming languages, like Java and C++. The choice of .NET is important as the .NET libraries include powerful tools for producing text-to-speech and performing HTTP requests. These are crucial to the development of the speech production functionality for the Rovio-Fevahr. Below the implementation of this functionality is discussed. This is not a thorough examination of the code, but details the important aspects.

## 4.1 Text-to-Speech and Audio Data

Since Fevahr agents utilize a generation procedure that results in NL text, a method for taking that text, converting it into audio data, and playing that data on the Fevahr-Rovio is required. In .NET the text-to-audio data is performed as follows (comments are in line and preceded by '//'):

```
// Create a memory stream to write the audio data in
Stream memStream = new MemoryStream();

// Create the speech synthesizer with the default voice
SpeechSynthesizer speaker = new SpeechSynthesizer();
```

[2]While programming can be done independent of an operational Rovio, in order to actually execute the program one needs a functioning Rovio set up on a network. Instructions for setting up a Rovio can be found here [10].

```
// Set the synthesizer to output audio data to the memory stream.
// This audio data is an audio format the Rovio expects.
speaker.SetOutputToAudioStream(memStream,
   new SpeechAudioFormatInfo(8000,
      System.Speech.AudioFormat.AudioBitsPerSample.Sixteen,
         System.Speech.AudioFormat.AudioChannel.Mono));

// Set the volume (int) and rate (int) of the audio data
speaker.Volume = volume;
speaker.Rate = rate;

// Have the text-to-speech engine convert the (string) NL text into the
// audio data
speaker.Speak(text);
```

After the above code is executed the English NL *text*, a C# string, is converted into audio data and stored into a memory stream, *memStream*. Before actually sending this data to the Rovio, it needs to be converted into a byte stream the Rovio can interpret and play, this is done as follows:

```
// Transfer the contents of the memory stream into a byte buffer for
// HTTP transmission
memStream.Seek(0, SeekOrigin.Begin);
byte[] rovioSpeechOutput;
rovioSpeechOutput = new byte[memStream.Length];
memStream.Read(rovioSpeechOutput, 0, (int)memStream.Length);
```

After the above completes, *rovioSpeechOutput* can be used to make the Rovio say the NL text provided. This is discussed in the following section.

## 4.2 Audio HTTP Requests

As mentioned previously, the Rovio is issued commands, sent information, and delivers information to a client using HTTP. This process varies based on the sensations one wants to access, or actions one wants the Rovio to perform. In order to implement the speech production aspects of the Rovio-Fevahr the HTTP connection not only needs to provide the appropriate URI, but also needs to provide audio data through the use of a HTTP POST method. To start this process in C# .NET one first creates a socket connection to the Rovio as follows:

```
// Rovio information
string rovio_address = "192.168.1.2";
int rovio_port = 80;
```

```
...

// Create the audio socket as a streaming, internetwork (IP) socket using
// the TCP protocol.
Socket rovioAudioSocket =
   new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// Establish a remote endpoint at the Rovio
IPEndPoint remoteEndPoint =
   new IPEndPoint(IPAddress.Parse(rovio_address), rovio_port);

// Connect the audio socket to the remote endpoint
rovioAudioSocket.Connect(remoteEndPoint);
```

The code creates a socket to send HTTP requests and audio data through. This is done by first establishing the properties of the TCP/IP socket; the properties of the end point, in this case the Rovio web server, which is a located at *192.168.1.2*, port *80* in the above example; and connecting the socket to that end point. With the socket created the next step it to create the actual HTTP request, this is done as follows:

```
// Create the HTTP POST command header in order to initiate the transfer of
// audio data
string httpPostString =
   "POST /GetAudio.cgi HTTP/1.1" + Environment.NewLine
   + "User-Agent: AudioAgent" + Environment.NewLine
   + "Host: " + rovio_address + Environment.NewLine
   + "Content-Length: 2147483647" + Environment.NewLine
   + "Cache-Control: no-cache" + Environment.NewLine + Environment.NewLine;

// Convert the data into byte data
byte[] httpPostData = StrToByteArray(httpRequestString);

// Send the byte data through the socket to the Rovio. A return value of
// -1 indicates failure.
if (rovioAudioSocket.Send(szSend) == -1)
   Console.WriteLine("Error sending initial packet");
```

The above code creates an HTTP request header. Understanding the HTTP protocol is beyond the scope of the report, but details of the various fields can be found in [1]. The important field is the *Host*, which provides the necessary connection information, in this case *192.168.1.2*. Also of importance is the actual command accessed on the Rovio's web server,

in this case it's the */GetAudio.cgi*. This command instructs the Rovio to expect audio data, and prepare its speaker to play the audio. In order to provide it with this audio data the following is used:

```
// Creates a byte buffer to fill with audio data
byte [] rovioSpeechOutput;

// Create audio data (previous section)
...

// Send the audio data through the socket, finishing the HTTP transaction
rovioAudioSocket.Send(rovioSpeechOutput);
rovioAudioSocket.Close();
```

The above code sends the contents of *rovioSpeechOutput* across the socket, and closes the socket. This completes the entire HTTP transaction, and will cause the Rovio to speak the audio.

### 4.3 Say Function

The code discussed above is enough to allow the Fevahr-Rovio to say an English NL text string. For convenience it is wrapped inside a function called "say" with the following definition:

```
// Say English phrase text at a given volume and rate
public void say(string text, int volume, int rate)
```

This definition is part of a *Rovio* class in the .NET namespace *RovioInterface*. Thus, to call it the following C# code can be used:

```
// Create an instance of the Rovio with IP 192.168.1.2
RovioInterface.Rovio myRovio = new RovioInterface.Rovio(''192.168.1.2'');

// Have Rovio say a greeting at volume 100 and rate -1
myRovio.say(''Hello there!'', 100, -1);
```

With this function one can get the Fevahr-Rovio to speak NL utterances produced by its grammar. To do this an implementation of the other GLAIR layers is needed.

## 5 KL, PMLa, and PMLb: Answering Questions and Generating a Response

In addition to producing text-from-speech the Rovio-Fevahr needs to generate a text response based on its environmental situation. In Fevahr agents such situations typically involve

questions, assertions, or commands issued by one or more people in the environment in a natural language [7]. For example, after asking the ASCII Fevahr the question, "Who are you", the ASCII Fevahr responds "I am the FEVAHR and my name is 'Cassie'." This is accomplished through the interaction of the upper levels of the GLAIR architecture, the KL, PMLa, and the PMLb. The following sections describe these layers in the Fevahr-Rovio, and how they are used to generate speech.

## 5.1 KL

The KL for the Rovio-Fevahr remains unchanged from the ASCII Fevahr described in [7]. The KL contains various assertions written in SNePSLOG, a higher-order logic syntax used to represent the beliefs of the Rovio-Fevahr, such as the belief that its name is Cassie.

## 5.2 PMLa

The PMLa has gone under slight modification in the Rovio-Fevahr. In order to generate a response from the Fevahr grammar a natural language assertion, question, or command must be provided from the environment. In the ASCII Fevahr this is typically accomplished through a NL interaction prompt. However, for developing cognitive robots, like the Rovio-Fevahr, a way of generating this response outside of a prompt is required. For this purpose a new Common Lisp function was produced and added to SNePS 2.7.1, *parser:nl-tell*. This function is defined as follows:

```
(defun nl-tell (sentence)
  "Given a string sentence. Invokes the parser and returns a string
   result. Useful for agents that need to listen, and produce a response."
...)
```

## 5.3 PMLb

The PMLb for the Rovio-Fevahr requires the most modification from the ASCII Fevahr as it must interact with the .NET code created for the PMLc level. In order to accomplish this *RDNZL*, an interface from Common Lisp to .NET was utilized [8]. To use RDNZL, load the source prior to any code that needs it following the instructions provided here [8]. After RDNZL is loaded any .NET assemblies (e.g., dynamically-linked libraries (DLL)) that one wants to interface with need to be placed in the Common Lisp home directory. For the Rovio-Fevahr there is one DLL, *RovioInterface.dll*, which contains the PMLc functionality discussed above. With this accomplished RDNZL can be initialized as follows and the DLL accessed with the following Common Lisp code:

```
;;; Enter the RDNZL package and enable RDNZL commands
(in-package :rdnzl-user)
```

```
(enable-rdnzl-syntax)

;;; Import the RovioInterface namespace and the Rovio class
(import-types "RovioInterface" "Rovio")

;;; Switch to the RovioInterface namespace
(use-namespace "RovioInterface")
```

The above code prepares the PMLb layer for communication with the *RovioInterface* library. The *enable-rdnzl-syntax* method is a mandatory call needed before using the RDNZL interface that changes the Common Lisp readtable.[3] The *import-types* method's first argument is a .NET assembly and the subsequent parameters are a list of types to import from that assembly. In this case the *RovioInterface.dll* assembly is loaded, and the *RovioInterface.Rovio* type is imported for use. Finally, the *use-namespace* takes one string, which is a namespace to utilize when resolving symbol names. In this case, the "RovioInterface" namespace is the only one needed. With RDNZL initialized and interface established on can begin creating .NET object instances and make .NET foreign function calls. For the purposes of the Rovio-Fevahr the following functions are sufficient:

- *(new type &rest args)*: Invokes the constructor for object denoted by string *type* with arguments *args*.
- *(invoke object method-name &rest args)*: Invokes the .NET *method-name* method with object instance *object* and arguments *args*.

With the above we can create and instance of the Rovio class with IP '*192.168.1.2* and call the speech method on the results of an *nl-tell* with the following code:

```
// Create the connection to the Fevahr-Rovio and store
// a reference to the object in my_rovio
(setf my_rovio (new "Rovio" "192.168.10.2"))

// Invoke the Rovio.say method using the Rovio object created above
// and the results of an nl-tell query
(invoke my_rovio "say" (parser:nl-tell "Who are you?") 100 -1)
```

After the above code executes the Fevahr-Rovio will actually say the appropriate response generated from the grammar (i.e., "I am the Fevahr and my name is 'Cassie'.").

[3]To restore the readtable *disable-rdnzl-syntax* can be called.

# 6 Future Work

This paper has demonstrated how one can start development of a complete Rovio-Fevahr, and contributes part of the NL equation (i.e., producing a response to a NL command, assertion, or query). The other half of the NL equation, speech recognition, has not been implemented yet. Scott Settembre has developed a RovioWrap [5] interface to the Rovio that can do some minimal voice recognition. However, this suffers from an inelegant solution that has very poor performance. This isn't the result of the .NET speech recognition software, so much as that the only solution available to the interface, at this time, is to take two second intervals of voice data, and attempt a recognition from those small audio samples. A solution to this problem is in the works.

Other than speech recognition and generation functionality, work on the various other primitive actions available to the Fevahr needs to be implemented in the Rovio-Fevahr. These include actions like finding certain individuals in the environment and performing special movements on command.

# 7 Conclusions

The .NET Framework is useful for the development of the PMLc layer of cognitive robots, like the Fevahr, when using robot platforms like the Rovio, which have on board microphones, speakers, and web cameras. Of particular usefulness are the .NET speech libraries, which include functions for the production of speech from text and the parsing of text from speech. With interfacing layers between Common Lisp and .NET, like RDNZL, the SNePS reasoner and NL generation facilities can easily be incorporated into robotic platforms using .NET.

# 8 Acknowledgments

Michael Kandefer acknowledges the assistance of Scott Settembre whose *RovioWrap* [5] was indispensable in creating the *C#* code for the text-to-speech and HTTP audio transmission.

# References


[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Maninster, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. `http://www.w3.org/Protocols/rfc2616/rfc2616.html`, 1999.

[2] Henry Hexmoor, Johan Lammens, and Stuart C. Shapiro. Embodiment in GLAIR: a grounded layered architecture with integrated reasoning for autonomous agents. In Douglas D. Dankel II and John Stewman, editors, *Proceedings of The Sixth Florida AI Research Symposium (FLAIRS 93)*, pages 325-329. Florida AI Research Society, April 1993.

[3] Michael Kandefer and Stuart Shapiro. Knowledge acquisition by an intelligent acting agent. In Eyal Amir, Vladimir Lifschitz, and Rob Miller, editors, *Logical Formalizations of Commonsense Reasoning, Papers from the AAAI Spring Symposium Technical Report SS-07-05*, pages 77–82. AAAI Press, Menlo Park, CA, 2007.

[4] Microsoft Corporation. Microsoft .NET Framework. `http://www.microsoft.com/NET/`, April 2009.

[5] Scott Settembre. Roviowrap. `http://roviowrap.codeplex.com/`, February 2009.

[6] S.C. Shapiro. Sneps: A logic for natural language understanding and commonsense reasoning. In M. Iwanska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language*. AAAI Press, Menlo Park, 2000.

[7] Stuart C. Shapiro. FevahrCassie: A Description and Notes for Building FevahrCassie-Like Agents, SNeRG Technical Note 35. Technical report, Department of Computer Science and Engineering, University at Buffalo, 2003.

[8] Edi Weitz. RDNZL - A .NET layer for Common Lisp. `http://www.weitz.de/rdnzl/`, March 2006.

[9] WoWee Group Limited. API Specification for Rovio Version 1.2. `http://www.wowwee.com/static/support/rovio/manuals/Rovio_API_Specifications_v1.2.pdf`, October 2008.

[10] WoWee Group Limited. Rovio: Mobile Webcam with True Track Navigation System: User Manual. `http://meetrovio.com/static/support/rovio/manuals/Rovio_Manual.pdf`, 2008.

[11] WoWee Group Limited. Wowwee rovio. `http://meetrovio.com/en/products/tech/telepresence/rovio/rovio`, April 2009.