TranSqueak - Making the world a smaller place
On-the-fly translation of Etoy projects and instant messaging

Michael Rüger, Yoshiki Ohshima

VPRI Technical Report TR-2004-001

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

TranSqueak - Making the world a smaller place
On-the-fly translation of Etoy projects and instant messaging

Michael Rüger
Yoshiki Ohshima
Viewpoints Research Institute
1209 Grand Central Avenue
Glendale, CA 91201, USA
michael@squeakland.org
Yoshiki.Ohshima@acm.org

Abstract

This paper presents an extension to the existing multilingualization work (m17n) which will allow people to collaborate on Squeak Etoy projects across different natural languages.

Squeak etoys support several languages, both ISO-Latin based ones (erg., English, German, French), and non-ISO languages (e.g., Japanese). Switching between languages for the Etoy tiles is fairly easy to support as the tiles provide a predefined set of words and phrases, which only need to be translated once.

There are two areas where we need to go beyond the predefined and pre-translated set of phrases: user supplied names and communication between collaborators. This paper will present an approach based on online translation services. We will demonstrate a working prototype and a first analysis of the feasibility of this approach.

1. Introduction

"The real technology
—beyond all our other technologies—
is language" [1]

Squeak, and with it Squeak Etoys, is open source software - which means it is a communal effort of people from various backgrounds, both technical and cultural. Throughout the programmer community ([2] [3] English is the lingua franka of choice, but even there the different cultural backgrounds occasionally lead to misunderstandings.

Since the Squeak Etoy system was made available on Squeakland.org [4] to a larger audience in early 2001 it has successfully been adopted in schools around the world. The multi-lingual Squeak developer community provided the necessary support for multiple languages and character sets. During the film shoot for the Squeakers documentary [5] an interactive session between students at a Squeak workshop at CAMP [6] and children from the Open Charter School in Los Angeles took place. It was then that we realized how the language barrier made this almost impossible to do. Being accustomed to more or less fluent communication in English with adults from all over the world we tend to forget that children do not have a fluency in foreign languages.

So when we started work on providing subtitles in several languages for the Squeakers documentary the idea came up to build support into the Squeak Etoy system to overcome the language barrier.

Squeak already supports several languages, both ISO-Latin based ones (e.g., English, German, French), and non-ISO languages (e.g., Japanese) [7]. The Squeak Etoy system with its tile based scripting system makes it fairly easy to support translated versions of the tiles as the tiles provide a predefined set of words and phrases, which only need to be translated once.

There are two areas where we need to go beyond the fixed set of phrases: user supplied names (objects, variables, scripts) and direct communication (instant messaging) between collaborators.

Squeak's network support tools can be used to connect to on-line translation services to translate strings on-the-fly. Thanks to Squeak's reflection capabilities we are actually able to identify strings that need to be translated and replace them with their translated counterparts. There are a number of special cases like composite words that need to be taken into account.

Communication between collaborators making use of the Squeak implementation of the Jabber [8] instant messaging protocol can be intercepted and translated using the same mechanism.

Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing (C5'04)
0-7695-2166-5/04 $20.00 © 2004 **IEEE**

VPRI Technical Report TR-2004-001

2. Machine Translation

Machine translation (MT), as natural language translation has been called historically, has become more widely available in recent years:

"One of the earliest pursuits in computer science,
MT has proved to be an elusive goal,
but today a number of systems are available
which produce output which, if not perfect,
is of sufficient quality to be useful
in a number of specific domains." [9]

In addition to a number of commercial products and services there are also a few free systems available, both standalone and online systems (Ergane, Traduki, Linguaphile, GPLTrans, Google language tools, Babelfish, Excite Japan [10-16]).

Most of the above systems are still in a rather rudimentary state, but Babelfish and the Excite Japan sites seemed promising. Those translators are primarily designed to translate short, but complete sentences. We'll discuss the applicability of them in the section "Evaluation".

Our original design goal was to use the Babelfish web service, but, unfortunately, it turned out it had been disabled. As the Excite Japan site does not support web services we decided to use the standard http protocol to send the translation requests and retrieving the translation results by parsing the returned html page.

The mechanics of corresponding with the particular translation service are encapsulated in the implementations of the WebTranslator clients for Babelfish and Excite Japan..

3. Translating Instant Messages

Collaboration on (Etoy) project requires a form of instant communication. Possible forms are communication by (video) phone, audio chat or instant messaging. As we are a looking at a way of building a bridge across the language barrier, none of the audio based forms of communication would work. At least not until real time translation of spoken language becomes available to us.

For our prototype we used the Squeak implementation of the Jabber protocol [8] which is based on XML. The current implementation of the XML tools in Squeak does not yet support multi-byte character encodings and needed to be patched accordingly.

The next step was to connect the chat client with the translation mechanism. There are two options to translate the message text: on the sender or the receiver side. We decided to have the receiver of a message perform the translation avoiding the need to first transmit the destination language setting to the sender first. We still need to transmit the source language setting. The XML standard fortunately already supports this through the "xml:lang" tag:

```
<message
        from="squeak@buccaneer.impara.de/home"
        to="michael@buccaneer.impara.de"
        type="chat"
        xml:lang="en">
        <body>Hello, how are you?</body>
</message>
```

After receiving a message the modified Jabber client extracts the body of the message and hands it over to the WebTranslator client. After receiving the translated string it replaces the original body of the message with the translated string and forwards it to the message list.

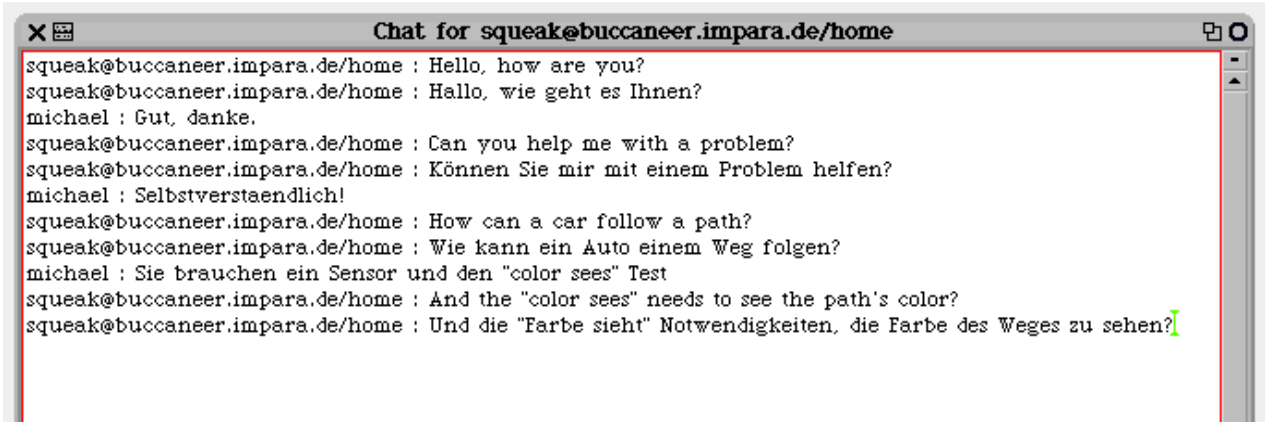

Figure 1:
An example dialog between German and English.
The original English phrases are shown with the translated counterparts for reference.

4. Translating Projects

In this section, we discuss the design and implementation of the Squeak project translation.

A Squeak project can conceptually be considered to represent a desktop, or the live state of a user's work consisting of a set of objects. The objects include visual and non-visual objects, and, of course, strings.

The Squeak system has a mechanism to extract all necessary objects within a project from the entire system, serialize them, and store them into a file. This file is called a project file. The file can be later loaded into Squeak and will continue the execution from the state when it was saved.

Our goal is to translate the strings that represent the names and symbols in a project, here specifically Etoy projects as those are the ones mainly used by children. One could imagine to translate all code written by programmers though.

In a Squeak Etoys project there are only a limited number of places where the user can input text or names.

They can be categorized into five types:

a) object name
b) script name
c) variable name
d) button label
e) text in a TextMorph

When a project is stored into a project file, the strings are saved together with the objects that refer to them. We utilize the knowledge about this structure to identify which strings should be translated. We need three steps to translate the strings in a project:

The first is to identify the strings to be translated (**Identifying**). The second is to translate the strings (**Translating**). The third is to replace the original strings with the translated ones gracefully (**Replacing**).

In the following, we will first discuss the overall strategy and then the details of each step.

4.1.The Strategy

Translating the strings in an Etoy project turned out to be more challenging than the straightforward approach used in the translation of instant messages.

Simply replacing strings in the project file is not possible as the length (and class) of a string may change in the process. Strings contained in projects also appear in a variety of contexts, each requiring a different approach and interpretation of the translation. Strings in a button can be simply translated and replaced whereas names of scripts or variables have to conform to certain syntactic restrictions (no white space, special characters, beginning uppercase letter etc) and are often composed of several word components. So basically we need to first analyze each string's context before we can start the translation process.

Opening a project internally involves three stages: loading the project (unserializing it), entering the project (setting up internal structures and connecting it to the system environment) and finally starting the interactive session. The second stage modifies some of the project structures, creating objects originally not included in the project. Therefore we need to identify all strings after loading the objects but before entering the project. After the project is fully set up we can then replace the strings with their translations before the interactive session starts (Figure 2).

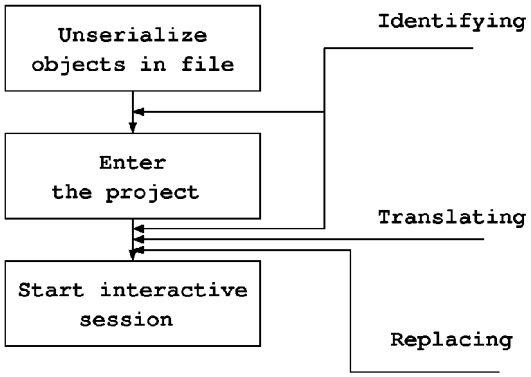

Figure 2:
Stages of project loading and translation

The translation step is relatively simple. After the identifying step, we use the same Web-based translator to get the translated result. However, there are many special cases to be taken into account. As we wrote above, the translated result of a script name, for instance, needs to be an acceptable (syntactically correct) script name. The result from the Web-based translator needs to be checked and modified if necessary to conform to these requirements. We will discuss the details in section 4.3 .

Finally, the original string has to be replaced with the translated result. Although there are existing methods for replacing object names, and to a limited extent for script names, these methods rely on the current Presenter which governs the scripting system status in the current project. This means that the replacing step needs to be done after the project has been entered (see again figure 2)

4.2.Identifying the Strings

Identifying the strings to be translated is done after a project file is loaded, all objects in the file are unserialized, and the references between the objects are proper-

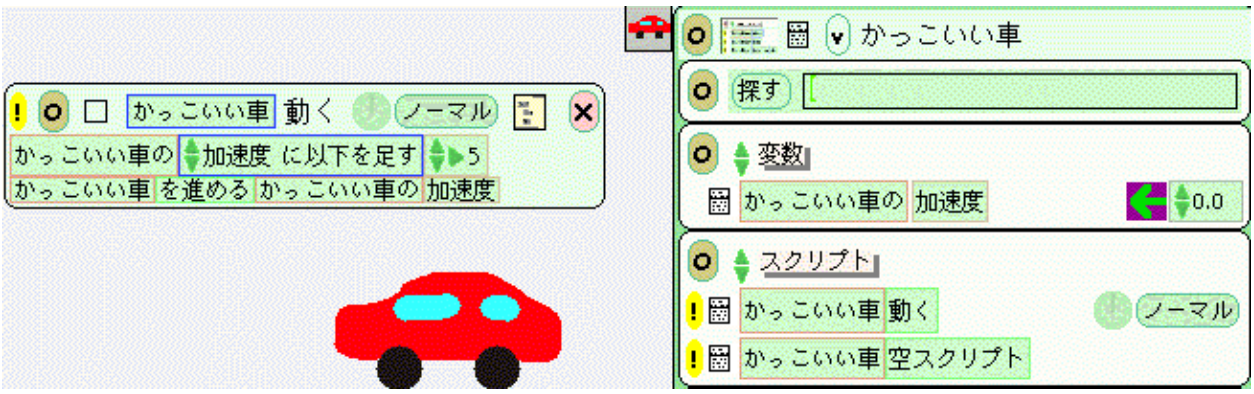

Figure 3:
An example Japanes Etoy with script and variable names

ly restored. At this point, the ProjectLoader class has access to the array of all objects in the project. This array is called arrayOfRoots. The expression in figure 4 retrieves the array of morphs with a name that needs to be translated.

```
morphOrList arrayOfRoots select:  [:ob |
        ob isMorph and: [ob knownName notNil]]
```

Figure 4:
Retrieving the array of "morphs to be translated"

All user created Morphs, either painted, grabbed from the supplies flap, or the object catalog, have a property called knownName. A knownName is indeed a type of the strings to be translated. The translator selects the sub-instances of Morphs which have a knownName defined from the arrayOfRoots In the following, this selected array is referred to simply as "*morphs to be translated*". User-defined scripts and variable names can be tracked from this array. The same applies to buttons and TextMorphs as they also have the knownName property.

A special case here are the watchers for variables. Watchers are instances of UpdatingStringMorph with no knownName defined, but they own a reference to the name of the variable. In order to translate the variable references in watchers correctly, the instances of UpdatingStringMorph are selected from the arrayOfRoots as well.

Note that not all UpdatingStringMorphs are watchers and therefore might not need to be translated. In the subsequent process, we check the accessors in the watchers to determine whether the instance of UpdatingStringMorph should be updated.

Up to this point everything can be done using methods in class ProjectLoading. However, at a later stage we would like to utilize the existing methods of Presenter to distinguish user defined script names and variable names from others. We therefore store the user created objects and the array of UpdatingStringMorph in the project parameters.

After entering the project, Presenter's allKnownScriptSelectors method is used to identify the user defined script names. The user-defined variables are defined in sub instances of Player. So variable names are collected from those players that are associated with morphs in the "morphs to be translated" array.

The contents of TextMorphs, and labels of buttons are also collected.

4.3.Translating the Strings

The translation is performed on the collected strings in a similar manner as with the chat messages. However, care has to be taken for the different context in which strings appear. For example, the script names should not contain any white spaces or special characters. To adjust the result from the Web translators to Squeak's syntactic conventions, the strings have to be processed and modified if necessary. This is simply done by removing the "dangerous" characters.

In English, multi-word script names are often composed out of several concatenated words; e.g. turnRight. The same is applies to variable names. To translate these strings as well, the concatenated words need to be split into their components before being passed to the Web translator. A simple state-machine based converter that looks at the Capital letters in the string suffices in this case. After the translated components come back the string needs to be composed again, meaning the strings have to be processed **before and after** they are passed to the Web translator.

Another detail in this step is the fact that there is already a large set of translated words. The localized version of Squeak Etoys includes a dictionary with app. one thousand entries with translations for button labels, default object names, command and slot names, and dialog messages. Those should be translated

consistently. The translator should take this dictionary into account and consult it before going through the Web translator.

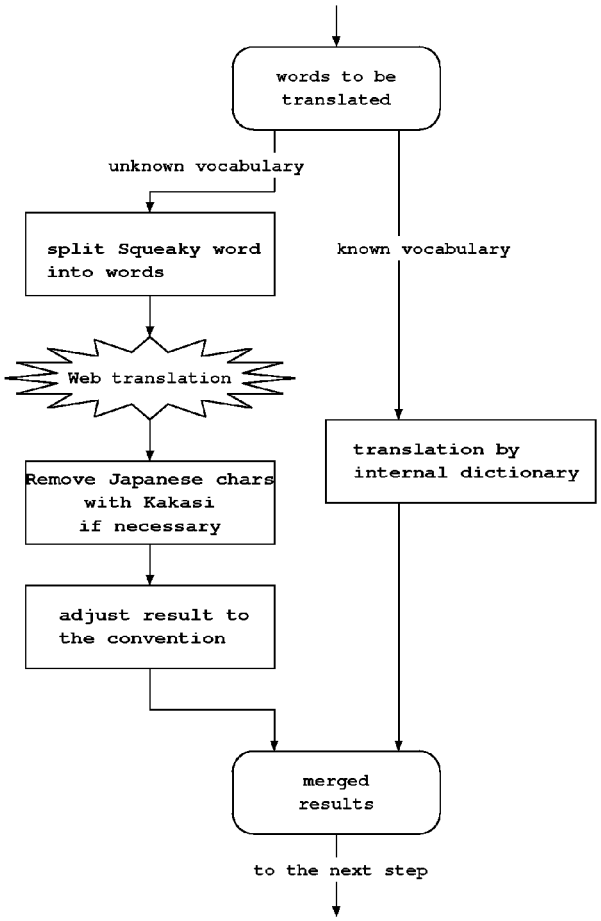

Figure 5:
The steps involved in the translation of script, variable or object names

A final detail is the asymmetrical nature of the translation requirements: a non-English speaking teacher or student is usually forgiving of the occasional appearance of English words, but an English teacher or student will have a problem if a translated project still contains Japanese characters (see figures 6 and 7). Since the Web translator typically returns the original if it cannot translate the given words, this is not just a hypothetical situation.

This issue has to be addressed differently based on the direction of translation. From Japanese to European languages, we use Kakasi[17] to translate "all" the Japanese characters into alphabetical notation. While Kakasi cannot handle uncommon proper names, it allows us to replace all Japanese characters with reasonably good roma-ji notation. The Kakasi program is called via OSProcess [18].

4.4.Replacing the Strings

The final step is to replace the strings. Object names can be simply replaced with the #tryRenameTo: method. As the #tryRenameTo: method notifies the affected objects, we don't have to take any further actions in this case.

If a morph has an associate player the morph is scripted and may have user-defined scripts and variables. Renaming those requires special care because they are also used in the body of scripts.

The existing facility for renaming the script name (Player>>#renameScript: newSelector:) does not change the script names used in the ScriptEditorMorphs. While we implemented a reasonably good mechanism to translate the script names in the tile scripts, this is still an on-going issue. There are cases where the automatic conversion is not guaranteed to work: a script name may be used as the argument for the PlayField's #tellAllContents: or even stored into a variable if the type of the variable is set to script name. The real problem for those cases is that the

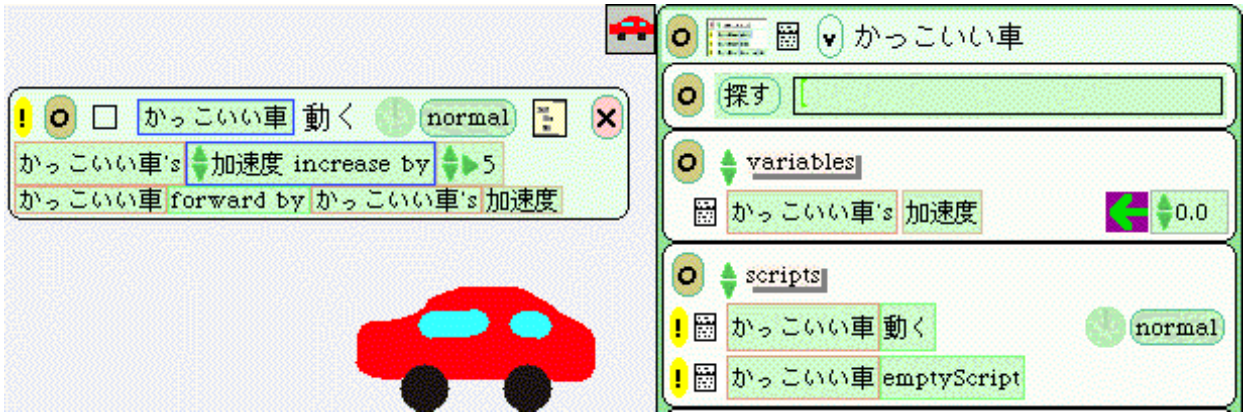

Figure 6:
An example Japanese Etoy with only partial translations

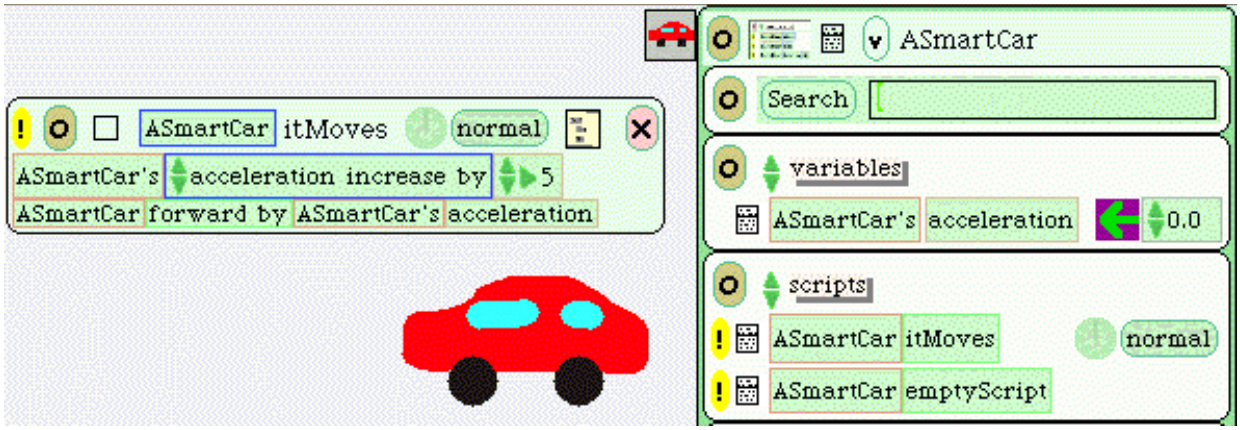

Figure 7:
The fully translated Japanese Etoy

script name can be detached from the Player that owns the script. The interactive script rename function of the Squeak Etoy system cannot tell which of the symbols it should rename, because there may be different scripts in different objects that share the same name.
Fortunately, we can expect that the same script name in different objects will be translated to the same one. If we translate "all" of the occurrences of a script name (represented as an instance of Symbol), no matter where they are, into the same translated string at once, most projects can be expected to continue to work after the translation process.

One may think that renaming variable is as tricky as it is the case with script names. However, a variable name is strongly tied to the player in a script, and it only appears in the scriptors. It is possible to implement an interactive function to rename a variable that can be guaranteed to do the right thing. We have implemented the interactive function first, and then the translator utilizes this function in this step. The interactive function recursively traverse the tiles in the ScriptEditorMorphs, and renames the appropriate variable occurrences.

The interactive variable renaming function has to take the two special cases into account. One is watchers and the other is siblings.

A watcher, or an UpdatingStringMorph which behaves as a watcher, accesses the value of the variable it is associated with by calling the standard getter and setter. If the target is the Player that owns the variable being renamed and the setter and the getter selectors match the standard getter and setter, the function updates the accessors as well.

The other case is siblings. If a variable of one of the siblings is changed, all of the siblings and the watchers watching the siblings have to be updated as well.
Translation of button labels is mostly straightforward. However, some of the system defined buttons, such as the "trigger" setting button at the top of the ScriptEditorMorph, has the knownName. While translating it does not hurt program execution, it isn't the "precise" way.

TextMorph content translation is straightforward.

5.Evaluation

For testing purposes we gathered the 115 projects from "Project Okiba"[19]. Not all of them are pure Squeak Etoy projects, but we did get a good sense on how well the translator performs.

Attempting to translate all of the projects provided great input to the debugging process, as the site has a wide variety of projects by different authors. The flexibility of the Squeak Etoys system allows the user to construct her projects in very different ways. Limiting the tests to our own projects only would never have shown all the different cases.

We found that the translation quality of strings in projects was not satisfying. One of the reason is that online translators are not designed to handle short names or incomplete sentences as they are typically used in programs. In case of Squeak Etoys, programmers tend to use simpler words, often words in Hiragana. An incomplete sentence with excessive Hiragana seems to confuse the Web based translators easily.

One the reason they use simpler words is there are many inexperienced programmer (or people who consider themselves non-programmers) trying to write. Also the existing translation is aimed towards children and tends to use hiragana.

Even with the small samples size of instant messaging dialogs we made a couple of interesting observations concerning the behavior of the web based translators we used. For one the Japanese translation tends to err on the side of politeness. This is even

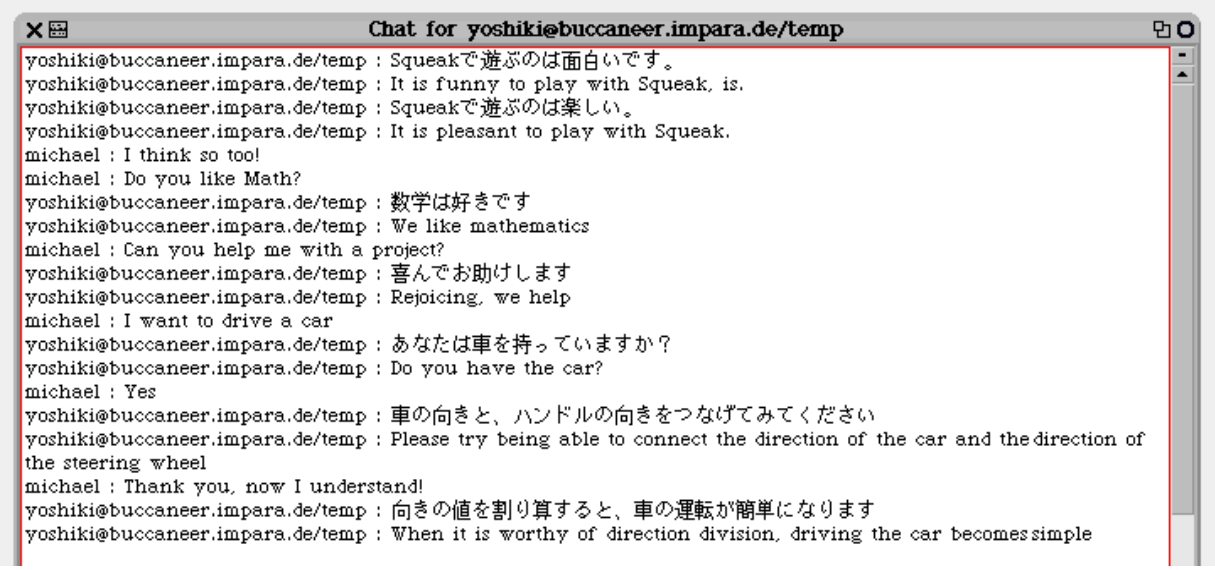

Figure 8:
An example Japanese-English dialog

visible in the English translation in the dialog seen in figure 8. Another is the obvious difficulty of dealing with referential expressions in sentences.

More on the amusing side: various literal translations of the word "Squeak".

6.Future Work

As we test more programs, we'd expect to gather more typical words and extend our own dictionaries for user-defined Squeak Etoys vocabulary. The translations could also be cached so they would be available offline. Human intervention: After an initial machine translation the result could be refined by the user and then cached for future reference.

7.References

[1] Norman Fisher, (Wired 1999: 134).
[2] http://squeak.org/
[3] http://lists.squeakfoundation.org/listinfo/squeak-dev
[4] http://squeakland.org/whatis/whatishome.html
[5] http://www.squeakersfilm.org/
[6] http://www.camp-k.com/
[7] Yoshiki Ohshima and Kazuhiro Abe.
The design and implementation of multilingualized squeak. In Proceedings of the Conference on Creating, Connecting and Collaborating through Computing (C5), pages 44–51. IEEE, 2002.
[8] http://www.jabber.org
[9] http://www.eamt.org/mt.html
[10] http://www.travlang.com/Ergane/
[11] http://traduki.sourceforge.net/
[12] http://linguaphile.sourceforge.net/
[13] http://translator.cx/
[14] http://www.google.com/language_tools
[15] http://babelfish.altavista.com/
[16] http://www.excite.co.jp/world/text
The translator at the excite site uses the technology licensed by Amikai, Inc. http: //www.amikai.com.
[17] Kakasi http://kakasi.namazu.org/
[18] David T. Lewis.
OSProcess package on SqueakMap.
`http://map1.squeakfoundation.org/sm/`
package/812c9d14-5236-4cad-82ea-`cc3e3837e30d`.
[19] Kazuhiro Abe and et. al. Project okiba.
`http: //swikis.ddo.jp/abee/3`.