Georg Heeg

Flexible Technologies to Visualize and Transform Terminological Representations
Modelling Representations instead of Programming using Smalltalk

Abstract
This paper discusses a software design approach to allow interchange of linguistic data. It focuses on the modelling of the linguistic concepts represented in the data and describes the transfer between exchange formats as a multi-tier interpretation/generation. These concepts are implemented in Smalltalk, a programming environment enabling flexible conversion of data between formats supported by Terminology Management Systems (TMS).

1 What is the Issue?
Most of today's software suffers from being inadequate in its innermost part. It is constructed from bits and algorithms.

Many papers in the workshop which have been documented in this issue of LDV Forum describe the tedious tasks to transfer contents from one terminology system to another. This issue is perceived as difficult because the domain of transfer requires an understanding of different domains at the same time.

This paper describes the use of Smalltalk to understand these different domains and to provide an implementation of the transfer problem at the same time. For Smalltalk, the key task is modelling instead of programming, and thus Smalltalk closes the gap between human thinking and its implementation in software.

The main question shifts from "How shall a particular feature be executed?" to "Who is responsible for a particular task?"

2 Modelling vs. Programming
2.1 Modelling in the Good Old Days
Before computers were invented in the mid 20th century, all computing was done by people, all algorithms were executed manually and it was important to represent knowledge in terms understandable to humans.

Where formalisms were needed, forms had been developed which resembled the thinking and terminology of the domain in question.

To give an example, assume you had a toothache in those days before computers were installed in dental practices. So you walked to your favourite dentist. She looked at you from her professional point of view and she recognized you as her patient. Additionally, she had a form which contained all her terminology produced by professional dentist publishers whose primary goal was "ease of use". This was realized by a good mapping of dentists' concepts on paper (see Fig. 1).

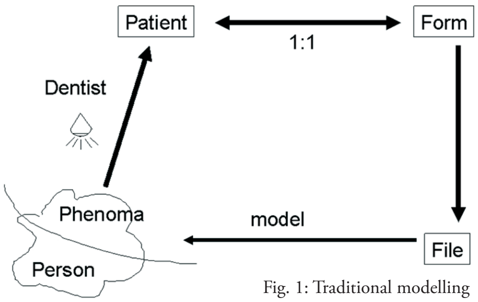

Fig. 1: Traditional modelling

The file which is essentially the filled form represented the patient's phenomena of interest to the dentist in an adequate and understandable form.

2.2 Modelling in the Computer Days

After computers were invented in the 1940s they were extremely expensive. In the 1970s, mainframe computers were 100 times slower and larger than today's PCs. One hour of usage of such a mainframe amounted to one monthly salary for a programmer. In this spirit most of computer technology was developed. The common mind set was and is that computers are expensive and thus it is mostly important to represent all information and procedures the computer way. And computers have two major components: CPU and Memory.

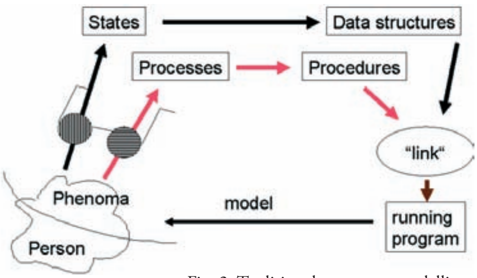

Fig. 2: Traditional computer modelling

This mind set influences still today many software projects and starts from the analysis phase. Figure 2 illustrates the traditional computer modelling process. From the very beginning the question is different: Instead of looking at the ontology of a domain, the viewpoint uses a filter to search for data and procedures. The following examples illustrate the limitations of this approach:

Example 1

1. Wooden body in the form of a cylinder with approx. 20 cm (8 inch) height and 6 mm, (1/4 inch) in diameter.
2. The centre of the cylinder contains a drilling of 1 mm filled with pressed graphite.
3. At one end, the cylinder is conically tapered.
4. The graphite can be transferred to other bodies by rubbing.

Example 2

1. Plastic tube in the form of a cylinder with approx. 20 cm (8 inch) height and 6 mm (1/4 inch) in diameter.
2. Inside is another plastic tube with 2 mm (1/12 inch) in diameter and at the top there is a metal ball.
3. The inner tube is filled with a viscous liquid.
4. The liquid can be transferred to other media with the help of the ball.

In both examples the first three issues describe state/information while the last one describes process/procedural aspects.

Common Sense

Reading above descriptions a normal (non-computer) person will easily recognize that the first is a strange description of a pencil while the latter is a not less strange description of a ball pen.

When I ask a normal person (or even better a child) they will come up with a totally different description of pencil and ball pen: They serve to draw and write and the main difference is that using a pencil you can easily use an eraser to rub out.

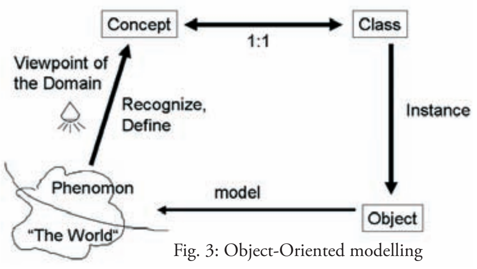

Fig. 3: Object-Oriented modelling

2.3 Modelling like in the Old Days

The basic idea of the programming language Smalltalk is to go back to this "naïve" understanding and to map concepts directly into software (see Fig. 3). The key idea resembles the strategy in the pre-computer times: understandability and adequacy.

2.4 Objects and Data

Objects in object-oriented languages like Smalltalk primarily care about the services they provide. Data are encapsulated inside the objects and are not visible from the outside.

3 Applying Objects to Transfer Problems

Now let us start to apply this approach to the exchange of lexical and terminological resources.

To do so, we first analyse the problems in this area and categorize them into different kinds of problems.

Similarly to communication levels in general, one can observe in our domain three different types of problems: Syntactic, structural and content mismatches.

3.1 Syntactic Mismatch

Some systems produce exchange files in plain text files, one line per entry, separated by special delimiters like "@@@". If this delimiter is a comma, these text files are often called CSV files (comma separated values); if it is a tabulator character these files are called TSV files (tab separated values). As these two file types can be read with Microsoft Excel, they are often called Excel files.

Another file format, which has become very common recently, is XML (eXtensible Mark-up Language). XML files are annotated trees represented in a linear fashion. Obviously, this approach offers more flexibility than plain text or Excel files, as entries with differently structured information can be stored in the same file.

3.2 Structural Mismatch

Let us assume we have a match in the syntactic form of a file, we can still have strong incompatibilities. The structures of the files do not match.

Examples are: Excel file columns do not match; XML files have different DTDs (DTDs and schemata describe the structure of XML files).

3.3 Semantic Mismatch

File contents do not necessarily match even if all structures match. There is still room for incompatibilities. Examples are: Some systems allow importing language pairs only; others allow importing many languages at the same time. Some allow multiple entries to represent homonymic terms, other require special entries.

4 SSS-TTT

As we found three levels of problems we will also start with three levels (also called tiers) of solutions: our architecture implements a syntactic, structural, semantic three-tier transfer (SSS-TTT).

Additionally, we have the desire to get a universal converter which can convert any input form to any output form. Universal converters are much easier to design in a so-called star architecture which consists of a common internal representation which can be filled by any input converter and can output to any destination format.

In this particular case of a layered problem description universal converters can be developed for each tier reducing the number of converters even further providing standardized interfaces between the tiers.

The syntactic tier converter reads the files and maps keys to values. Keys can be column numbers, column headers or XML entity labels, values normally are strings, for inner XML nodes values are trees.

Converters of the structural tier map logical attributes onto (potentially multiple keyed) values.

These values are transformed in the semantic tier into "meaningful" objects.

If needed, transformation and filtering is done on the level of meaningful objects, the result of the semantic layer.

For each converter there is a generator which operates in the opposite direction.

4.1 Implementation in Smalltalk

For the syntactic tier Smalltalk provides predefined classes. in particular an XML Parser/Generator and a CSV/TVS Reader/Writer.

Additionally, there is a bunch of technologies available to communicate directly to language software. COM Connect and .NET Connect use Microsoft inter-process communication, WebServices and Corba over IIOP connect into the Java world.

In Smalltalk, everything is in source and everything can be enhanced, it can be changed and adapted whenever needed.

The structural and semantic tiers map linguistic theories easily into software artefacts.

4.2 Smalltalk Development Process

Smalltalk always gives you immediate feedback, thus it fully supports agile programming. Thus it is good practice to interleave programming and testing all the time: "Make it work half way", "Try it out", "Make it work a little bit more", "Try it out again".

A well known development strategy in Smalltalk is:

1. Make it work
2. Make it right
3. Make it fast (if needed)

The main technique in steps 2 and 3 is called refactoring. "Refactoring is the process of rewriting a computer program or other written material to improve its readability or structure with the explicit purpose of keeping its meaning or behaviour" (Wikipedia 2006).

5 Linguistic Smalltalk Experiences

In cooperation between the Software Localization Group of Anhalt University of Applied Sciences and Georg Heeg eK (*http://www.heeg.de/*) several Smalltalk projects have been successfully developed. They fall into three categories: Small transfer tools to get data into professional MT, CAT and TMS systems, tools for Software Localization education and Software Localization research. One of these projects, which consisted of making Microsoft glossaries accessible for Software tools, demonstrates typical problems and their solutions. Therefore, we want to describe this project in more detail.

Microsoft provides its products in many languages. As described on the web page *http://www.lai.com/microsft.html* Microsoft provides its translation catalogs in 24 languages on ftp server *ftp://ftp.microsoft. com/developr/msdn/newup/glossary*.

When you unzip all files you will get 5.8 GB *.csv files. Most of them are too big to open them in Microsoft Excel. So other tools are needed to get access to this very rich resource.

When you try to import these glossaries you will see additional problems: From language to language the number of columns differs; several files contain no headers at all.

Mostly the filename of the glossary files indicates the language and country of the translations, but some language codes are represented with 3 characters, as German in "deu-deu-Access2003.csv", others with 2 characters, as Czech in "cz_vb50.csv", which are different ISO standards.

All of these problems have two things in common: There is no description at all and you step over them just by accident. So it is excellent to have an open flexible tool like Smalltalk with full control for the developer.

To read the files we started with subclassing CSVReader. In some of the CSV files the entries are separated by commas, in others by tabulators. We made CSV Reader doing the right guess.

Then we saw that the number of columns ranges from 8 to 255; we looked at the data and guessed the intention.

Some of the files had no columns headers at all, so we added guessing the column structure in CSVReader subclass.

Some of the files have a comment in the first line; our guess was easy: if the number of columns is 1, it is a comment.

As already mentioned, languages are indicated in filenames using different standards for the language codes, like in "deu-deu-Access2003.csv" or "cz_vb50.csv", so that the codes had to be transferred to homogeneous representations.

All files for the largest language pair (English -German) could be made available in a Smalltalk system, but all files for all languages (5.8 GB) cannot be loaded into current 32 bit VisualWorks systems. This requires a 64 bit version or an object database.

After reading you have a collection of objects representing the contents of all files read. These objects can be sorted by any sorting criteria, filtered anyhow, or matched against any input in a translation memory manner.

Last but not least these objects can be exported to any desired format. This allows loading subsets of the Microsoft glossaries into any translation memory or terminology management system. Examples are TMX and TBX files.

Certainly, the glossary tools can also be used in VisualWorks language tools developed at Georg Heeg eK and Anhalt University like LioN (Localizer for VisualWorks applications, see Lannatewitz 2003 and Haase 2005) and Web-TCM (Localizer for HTML-Pages; see Seewald-Heeg 2001).

6 Conclusion

Smalltalk serves as an ideal technology for linguistic tasks. It enables to create transformations between different terminological representations and modify them to get ad-hoc problems solved instantaneously. It is easy to try out new ideas and to observe the execution in graphic user interfaces immediately.

The main thing is "modelling instead of programming" to keep the entire software totally understandable.

References

Haase, C. (2005). "Lokalisierung einer Entwicklungsumgebung mit einem Nicht-Standard-System am Beispiel von LioN". Diploma Thesis, Anhalt University of Applied Sciences. *http://www.heeg.de/downloads/vortraege/DiplomArbeit-ClaudiaHaase-2005.pdf.*

Lannatewitz, D. (2003). " Entwicklung und Implementierung eines Lokalisierungswerkzeuges für VisualWorks". Diploma Thesis, Anhalt University of Applied Sciences, manuscript.

Seewald-Heeg, U. (2001). "Entwicklung und Einsatz von Lokalisierungswerkzeugen. Informatik-, Computerlinguistik-, Fachsprachenkompetenz". *http://www.heeg.de/~uta/PPT/Web-TCM/LokalisierungmitWeb-TCM.ppt.*

Wikipedia:Refactoring (2006). Refactoring – Wikipedia, The Free Encyclopedia, *http://en.wikipedia.org/wiki/Refactoring, accessed May 2006.*