# A Concept Analysis Framework

Christian Lindig

Technische Universität Braunschweig
Institut für Programmiersprachen
und Informationssysteme
D-38106 Braunschweig
Germany
Phone: +49 531 391 7465
Fax: +49 531 391 8140
Email: lindig@ips.cs.tu-bs.de

Concept analysis has been found a powerful data analysis tool since its invention over 10 years ago. More and more applications use concept analysis in a domain specific way. However, software designers are still forced to devote too much work on implementation of analysis techniques because most existing software can not be incorporated into new applications. This paper describes a software framework that provides domain independent abstractions for the two most common concept analysis structures: binary relations and concept lattices. The framework is implemented as an extension of the popular *Tool Command Language* (Tcl) for concept analysis operations. Tcl is an interpreted extension language which already provides abstractions for common application tasks. Using the framework the software designer develops software in a convenient interpreter environment and takes advantage of the provided abstractions for concept analysis. This can lead either to small "scriptable" tools, or, in conjunction with other extensions of Tcl, to full featured applications. As an example an interactive and graphical concept analysis workbench has been developed. The described software and its documentation is distributed via the world wide web.

# A Concept Analysis Framework

Christian Lindig

Abteilung Softwaretechnologie
Technische Universität Braunschweig
D-38106 Braunschweig
EMail: lindig@ips.cs.tu-bs.de

**Abstract.** Concept analysis finds its way into applications, but support for software developers is lacking: programmers must still concentrate on concept analysis techniques instead of application domains. TkConcept is a framework for concept analysis applications that offers domain independent support for concept analysis tasks. It is implemented as an extension of Tcl/Tk and provides abstractions for binary relations and concept lattices. Using this framework, concept based applications can be developed easier than with existing tools.

## 1 Introduction

Concept analysis was invented over 10 years ago by Wille [15] and has been found a powerful data analysis tool since then (cf. [3]). Although research is still going on (cf. [4, 3] for an overview), more and more concept based applications emerge: access control [11], software engineering [6, 2], and retrieval [7, 5] applications use concept analysis.

When creating a concept based application the designer realizes, that most existing concept analysis software is of little help for this task: it is either a specific application for a different domain, or an analysis tool which can not be used as a building block. So the designer has to devote too much work on mere implementation details of concept analysis techniques instead of the application domain. It would be helpful to use a framework that already provides all the common techniques in an abstract and domain independent way.

We propose TkConcept as such a framework to let the designer concentrate on the application. TkConcept is build as an extension of the increasingly popular Tool Command Language Tcl which already provides a rich set of abstractions for "real world" applications. TkConcept can be regarded either as a "scriptable" concept analysis tool with programmable in- and output formats or, together with other extensions, as full featured programming language for concept based applications.

## 2 TkConcept

### 2.1 The Tool Command Language

The *Tool Command Language* Tcl [9, 10] is a scripting language: it is interpreted and based on the only data type string. Tcl serves as a base for TkConcept—

together they are forming a framework. Tcl provides variables, control structures and access to many system features likes files and processes. It is, thus, similar to other shell languages like the Bourne Shell, but, unlike that, Tcl is explicitly designed to be extendible: the system is organized as C code library, implementing an interpreter together with a set of common commands. New commands, implemented in C, can be added as an extension to provide new domain specific abstractions. Once new commands are added they are indistinguishable from other built in commands. TkConcept is implemented as such an extension providing efficiently implemented operations for concept analysis.

### 2.2 Abstractions for Concept Analysis

Up to now, TkConcept supports the two basic structures in concept analysis: binary relations and concept lattices. To calculate the concept lattice of a context a relation must be created first. This is done by the `relation` command which turns the contents of an array into a relation. The relation then can be used to calculate the corresponding concept lattice. Runtime created commands represent relations and lattices; this leads to an object oriented flavor as the parameters passed to such commands can be regarded as methods passed to objects. The created commands permit access to all informations inherent to relations and lattices. In principle, a relation abstraction is not necessary to calculate concept lattices as relations can be represented by arrays. Nevertheless, having such an abstraction is efficient, convenient, and fits better to the model suggested by concept analysis theory. Among others, the relation abstraction offers the following features:

- Persistence. Relations can be written to and read from files.
- Attachments. Additional informations can be attached to objects and attributes. They help to establish connections between a relation and other program data structures.
- Query of common items. For lists of objects and attributes the lists of their common attributes and objects can be queried.
- Concept lattice calculation. From a binary relation its concept lattice can be computed. The lattice computing command creates a new command to represent the resulting lattice; the new command permits to access the lattice.

Figure 1 shows a small example for some relation operations. Micro processor architectures as objects are related to multi-platform operating systems as attributes. First, an array is filled with facts about architectures and processors. Then the `relation` command creates a relation from the array, that is, it creates a new command for accessing it, and returns the command's name which is stored in a variable `rel`. For every operating system the related architetures, just the objects related to the operating system, are reported.

For concept-based applications the concept lattice derived from a relation is naturally the most important information. The concept lattice abstraction provides the following operations:

```
set platform(sparc) { NextStep }
set platform(ppc) { MacOS OSF1 }
set platform(i486) { NextStep Linux }
set platform(m68k) { MacOS Linux }
set platform(alpha) { OSF1 }
# create relation,  and save command name for access and save
# it to a file
set rel [relation create platform]
$rel save "platform.rel"
# loop over all operating systems
foreach os [$rel attributes] {
    # find architectures related to $os
    set archs [$rel objects -attribute $os]
    puts "Operating System $os runs on architectures $archs"
}
```

**Fig. 1.** Example for binary relation abstraction

- Persistence. Concept lattices can be saved to files. This can lead to significant performance improvements of an application because the calculation of a concept lattice from a relation is computationally expensive. Loading a precalculated lattice avoids its recalculation.
- Lattice operations. Concepts can be queried, compared, met, and joined. Concepts are represented by short string handles. These handles are returned by some operations and can be used to investigate the represented concepts further.
- Attachments. Informations can be attached to concepts, serving as links between concepts and other data. This applies also to objects and attributes which are part of a lattice.
- Sub-/super concepts. For each concept its sub- and super-concepts can be obtained. The lattice abstraction distinguishes between direct sub/super concepts and the transitive closure. This makes it easy to create line diagrams of lattices.

Figure 2 shows, as a small example, some Tcl-code which loads a relation from a file, calculates its concept lattice, and traverses it from the top. For each concept the objects introduced by the visited concept are printed; to avoid duplicates each visited concept is marked.

The example in Figure 2 is somewhat hard to understand for readers unfamilar with Tcl's syntax and operational semantics. However, it shows that a few lines of code are sufficient to code a concept lattice traversal. There is an even simpler way to achieve the same result, as shown below. All concepts of a lattice are also returned by the `concepts` sub-command. So it is sufficient to

```
set rel [relation "platform.rel"]
# create concept lattice
set lat [$rel lattice]
# start at top concept
set top [$lat top]
# concepts holds the concepts still to visit - starting from
# the top concept
set concepts [list $top]
while {$concepts != {}} {
    # the actual concept is the first one of $concepts
    set concept [lindex $concepts 0]
    # remove the first concept one from concepts
    set concepts [lrange $concepts 1 end]

    # don't visit concept, if already visited
    if {[$lat attach concept $concept] != "visited"} {
        # mark concept visited
        $lat attach concept $concept "visited"

        # get all objects introduced by the actual concept
        set objs [$lat objects -new $concept]
        puts "new objects at $concept: $objs"
        # add direct subconcepts to list of concepts to check
        set concepts [concat $concepts\
                [$lat sub -direct $concept]]
    }
}
```

**Fig. 2.** Example for concept lattice traversal

query each of them for the introduced objects and to print them.

```
foreach c [$lat concepts] {
    puts "new objects at $c: [$lat objects -new $c]"
}
```

The abstractions for relations and concept lattices are domain independent and fit seamlessly into the Tcl system. Together Tcl and the concept analysis extension form a framework for all kinds of concept analysis tasks.

### 2.3 Why extending Tcl?

Extending Tcl to build a framework is not the only imaginable solution and the question arises why Tcl was chosen. Indeed, Tcl has some severe limitations: it

lacks user defined types, compilation and sophisticated scoping rules. On the other hand, Tcl is explicitly designed as an extension language. Currently, there are more than hundred extensions for all kind of tasks in "real world" applications: extensions for database access, distributed computing, network management, or object orientation which all can be combined. The Tk-extension for Motif style user interfaces provides a uniquely powerful abstraction which makes the development of such user interfaces a lot easier. Tcl and its extensions are portable between different computer platforms. Development in an interpreter environment is convenient and secure as errors are traped and do not lead to unrecoverable crashes. Because more complicated operations are implemented as C code extensions they do not cause performance loss. Tcl is an increasingly popular programming system—there is excellent documentation for programmers and extension authors available. This does not necessarily compensate its programming language limitations but addresses problems which are also important for all kinds of applications and which are often neglected by other systems.

## 3 Experiences

TkConcept was born out of negative experiences with previous concept analysis implementations: simple batch oriented concept analysis tools with fixed input and output syntax. It turned out that nearly every application required a change of the input and output syntax and, thus, required a specialized version of the analysis tool. Although most users prefer graphical user interfaces, a batch analysis is still handy and, so, one of the first applications built with TkConcept was such a tool. It computes a concept lattice from an input file, and it is controlled by command line options. Unlike the previous version, changes to its behavior do not require changes to C code and recompilation but changing a small Tcl source file.

A predecessor of TkConcept has been used to implement a search tool [8] for software component documentation based on a concept analysis of the documentation. The flexibility gained over a previous C++ coded version of the search tool was the motivation to build a general framework for concept analysis applications. It turned out that dividing the application into a concept analysis Tcl-extension and application-specific Tcl-code made the development easier. Especially, testing the extension and application was improved by using an interpreter environment.

To show the usefulness of the TkConcept framework a *Concept Analysis Workbench* was developed which may be interesting for the concept analysis community independent of its implementation technique. The following list shows its main features:

- Completely interactive, graphical user interface.
- Objects, attributes, and relations can be entered, edited, saved, and loaded easily.
- For a relation its concept lattice can be computed; it is displayed as a diagram in a graph editor.

– The graph editor permits to layout the diagram automatically or manually. Concepts can be joined or met and their contents can be investigated.

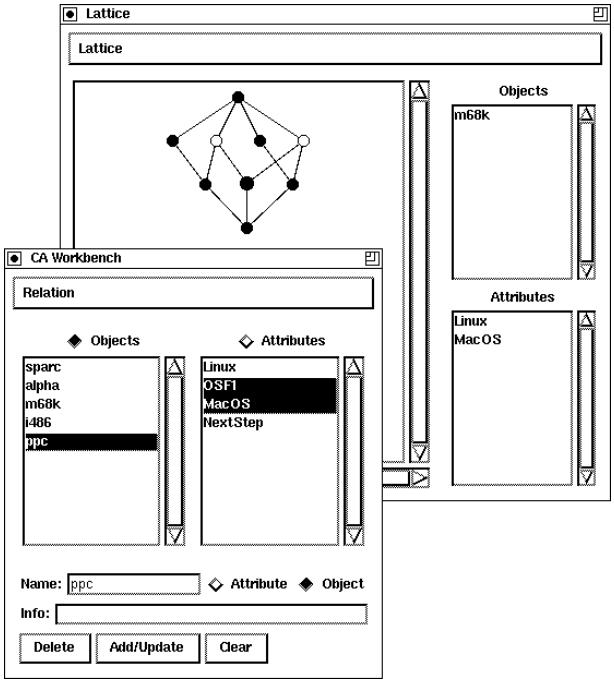


**Fig. 3.** Concept Analysis Workbench

Figure 3 shows a screen shot of the workbench. It is implemented within about 2000 lines of Tcl code and uses the commands provided by the TkConcept extension. The automatic lattice layout is realized using the external, batch oriented graph layouter *graphplace* [12].

## 4 Related Work

Toscana [14] is a graphical and interactive concept analysis tool developed by Wille's group. It runs on Microsoft Windows platforms and gets its input from a SQL database. Toscana is intended for scaled contexts which result in nested line diagrams. The diagram layouts are combined from predefined lattice layouts to achieve good results. So, it is not a framework or a building block that helps to create new applications but is itself a generic application similar to the *Concept Analysis Workbench*. In comparison, Toscana is more sophisticated, in some sense, as it can analyze scaled contexts which stem from multi valued contexts,

and can display them nicely. On the other hand, it is not user extendible, nor portable, nor does it provide fully automatic layouts, and it requires a database to read the input from.

The functional kernel of Toscana is the *Concept Analysis Library*, a C++ library [13]. As it provides some basic functionality for concept analysis it is directly related to TkConcept. It differs in the analysis technique (scaled contexts in contrast to binary contexts), the chosen language, and the typical environment applications are developed in. The latter is partly a matter of taste: while C++ offers object orientation, compilation, and type-checking, TkConcept provides an interpreter that shortens the edit-run-edit cycle. For both languages a rich set of toolkits or extensions exists which make application development a lot easier. Because TkConcept is divided in a C coded extension and the interpreted user level it provides a somewhat more comfortable environment for unexperienced users.

The predecessor of all concept analysis tools is conimp [1] which has been developed at Wille's group. It is a many featured, interactive, and terminal-oriented analysis tool with fixed input and output formats. Its analysis techniques go much beyond those provided by Toscana or TkConcept, but it lacks a visual presentation of the results and customization. Intended for interactive use it is difficult to be used as a building block in concept analysis applications. Because of this we developed a batch analysis tool for basic concept analysis. The inflexibility of input and output formats and the difficulties to use batch tools as parts of interactive applications led to the development of TkConcept—as mentioned previously.

## 5 Conclusions

Concept analysis has been found a valuable data analysis technique and finds its way into more and more domain specific applications. Nevertheless, support for building concept analysis based applications is lacking. Most concept analysis tools can not be incorporated into new applications and, thus, force the designer to re-implement the concept analysis techniques each time. TkConcept is a domain independent framework that provides ready to use abstractions for concept analysis. It helps the designer to focus on his or her domain instead of analysis techniques. Implemented as an extension of Tcl/Tk, it extends a free, portable, and popular system that already provides solutions to many common application tasks. The user of the framework takes advantage of the fact that Tcl is an interpreter but extensions as TkConcept are implemented in compiled C code. First experiences show that the gained flexibility indeed leads to noticeable productivity improvements. There are surely other ways beside TkConcept to create a framework; however, we believe that providing a framework for concept based applications is a step towards the right direction.

TkConcept, its documentation, and the demo applications are distributed via the world wide web: `http://www.cs.tu-bs.de/softech/tkconcept`.

**Acknoledgements.** Franz-Josef Grosch and Bernd Fischer provided valuable comments on earlier versions of this paper.

## References

1. Peter Burmeister. Programm zur Formalen Begriffsanalyse einwertiger Kontexte. Software, TH Darmstadt, Fachbereich Mathematik, Darmstadt, Germany, 1987.
2. Petra Funk, Anke Lewien, and Gregor Snelting. Algorithms for concept lattice decomposition. *Int. Conf. on Conceptual Knowledge Processing*, Darmstadt, February 1996. Submitted.
3. B. Ganter, R. Wille, and K. E. Wolff. *Beiträge zur Begriffsanalyse*. BI Wissenschaftsverlag, Mannheim, Germany, 1986.
4. Bernhard Ganter. Lattice theory and formal concept analysis. FB4-Preprint AL-2-1994, Technische Universität Dresden, Dresden, Germany, 1994.
5. Robert Godin, Rokia Missaoui, and Alain April. Experimental comparison of navigation in a galois lattice with conventional information rertrieval methods. *Int. J. Man-Machine Studies*, 38:747–767, 1993.
6. Maren Krone and Gregor Snelting. On the inference of configuration structures from source code. In *Proc. 16th International Conference on Software Engineering*, pages 49–57, Italy, May 1994. IEEE Comp. Soc. Press.
7. Christian Lindig. Concept-based component retrieval. In Jana Köhler, Fausto Giunchiglia, Cordell Green, and Christoph Walther, editors, *Working Notes of the IJCAI-95 Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs*, pages 21–25, Montreál, August 1995.
8. Christian Lindig. Komponentensuche mit Begriffen. In *Softwaretechnik '95*, Braunschweig, October 1995.
9. J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison Wesley, Reading, MA, 4th edition, 1994.
10. John Ousterhout. Tcl: An embeddable command language. In *Proceedings of the Winter 1990 USENIX Conference*, 1990. see also `http://www.sunlabs.com:80/-research/tcl`.
11. Herrmann Strack. *Sicherheitsmodellierung und Zugriffskontrolle in verteilten Systemen*. Dissertation, Universität Karlsruhe, Fakultät für Informatik, Germany, 1996. To appear.
12. Jos van Eijndhoven. Graphplace – a graph layouter. Software, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands, August 1994. distributed by anon ftp from `ftp.ele.tue.nl`.
13. Frank Vogt. The formal concept analysis library. Software library, TH Darmstadt and Ernst Schröder Zentrum, Darmstadt, Germany, 1995.
14. Frank Vogt and Rudolf Wille. Toscana – a graphical tool for analyzing and exploring data. In Roberto Tamassia and Ioannis. G. Tollis, editors, *Graph Drawing*, number 894 in LNCS, pages 226–233, Princeton, USA, October 1994. Springer-Verlag.
15. Rudolf Wille. Restructuring lattice theory: An approach based on hierarchies of concepts. In I. Rival, editor, *Ordered Sets*, pages 445–470. Reidel, 1982.

This article was processed using the LaTeX macro package with LLNCS style