development effort for OBST applications and by opening the way to functionality already available as Tcl extensions, most notably Tk and related tools. With tclOBST the full power of OBST is provided for convenient and explorative use in an interpretative environment.

In terms of generality, development effort, and the ease with which tclOBST can be combined with other functionality using a Tcl/Tk infrastructure, tclOBST is unique compared to the interactive interfaces offered for other object-oriented database systems [8].

Tcl/Tk proved to be an almost ideal basis for the described development effort. Its probably greatest asset is difficult to quantify: the experience that things fit together well and that solutions are accomplished fast and work well.

With larger tclOBST applications in mind, it might be about time for the Tcl/Tk community to reach consensus on larger grained programming abstractions than Tcl procedures.

References

[1] S. Delmas, *XF - Design and Implementation of a Programming Environment for Interactive Construction of Graphical User Interfaces*, Master's Thesis, Technical University of Berlin, Institut für Angewandte Informatik, March 1993.

[2] A. Lampen, *Advancing Files to Attributed Software Objects*, Proc. Winter USENIX Conf. 1991, pp. 219-229.

[3] C. Lewerentz, E. Casais, *STONE: A Short Overview*, STONE Tech.Rep. FZI.40.1, Forschungszentrum Informatik (FZI), Karlsruhe, May 1992.

[4] J. Ousterhout, *Tcl: An Embeddable Command Language*, Proc. Winter USENIX Conf. 1990, pp. 133-146.

[5] J. Ousterhout, *An X11 Toolkit Based on the Tcl Language*, Proc. Winter USENIX Conf. 1991, pp. 105-115.

[6] B. Schiefer, *An Environment for Supporting Schema Evolution in Object-Oriented Databases*, Tech. Rep., Forschungszentrum Informatik (FZI), Karlsruhe, in preparation (in german).

[7] C. Schürmann, *How to use the OShell in STONE*, STONE Tech.Rep. FZI.48.0, Forschungszentrum Informatik (FZI), Karlsruhe, October 1992.

[8] V. Soloview, *An Overview of Three Commercial Object-Oriented Database Management Systems: ONTOS, ObjectStore, and O_2*, SIGMOD Record, Vol. 21, No. 1, March 1992, pp. 93-104.

[9] D. Theobald, *The Design of a Tcl Interface to OBST*, STONE Tech.Rep. FZI.47.1, Forschungszentrum Informatik (FZI), Karlsruhe, April 1993.

[10] J. Uhl, D. Theobald, B. Schiefer, M. Ranft, W. Zimmer, J. Alt, *The Object Management System of STONE - OBST Release 3.3*, STONE Tech.Rep. FZI.27.2, Forschungszentrum Informatik (FZI), Karlsruhe, March 1993.

Appendix: How to get OBST / tclOBST

The current version of OBST and OBST applications such as tclOBST are freely available via anonymous ftp from ftp.fzi.de (141.21.4.3), directory /pub/OBST/OBST3-3.

The directory /pub/OBST/OBST3-3/psfiles holds postscript versions of the documentation which is as well contained in the distribution files.

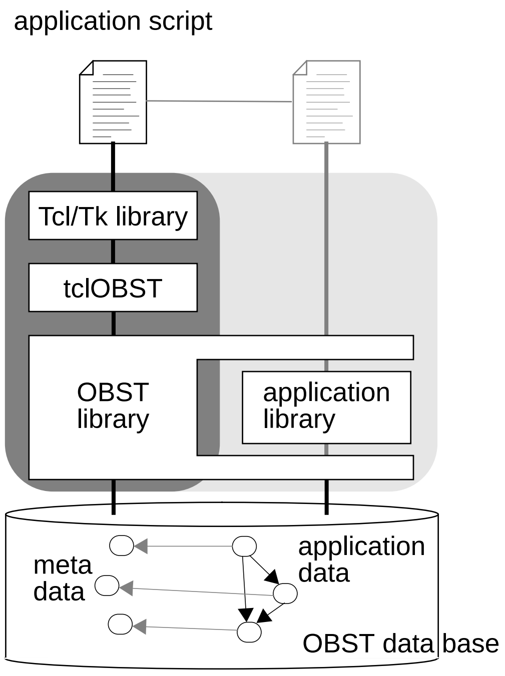

Figure 3 tclOBST application

3. Status and Future Work

Since the first release at the start of this year, a number of small applications were implemented based on tclOBST. tclOBST was furthermore used for prototyping and testing OBST code which was later on transfered to C++. These translations required only minor effort if Tcl specifics such as sophisticated string manipulations were used only sparsely.

Although we conducted no comparative studies, we feel development times to be significantly shorter based on experiences with previous C++ implementation tasks. This does in particular hold if the application includes a graphical user interface (e.g. a graphical browser for OBST meta data).

Some benchmarking results and usage experiences showed the overhead incurred by tclOBST - in comparison to programming an OBST application in C++ - to be acceptable and in most practical cases not even noticeable. However, this required substantial additions to the C interface described above for caching OBST meta data and hence reducing database lookups.

The total development effort for tclOBST was 3 person months, in which most time was spent in deriving the conceptual mapping of the OBST data model and implementing the C interface - the actual embedding in the Tcl framework took about two weeks.

Currently, tclOBST has reached a rather stable state and work is under way to implement larger applications based on it, e.g. a support environment for the evolution of OBST database schemas [6]. tclOBST will also serve as a base component for the integration of heterogeneous object bases: OBST and AtFS, an object base for file objects [2]. Integration will be achieved by drawing on existing Tcl embeddings for both object bases, i.e. OBST by tclOBST and AtFS by another interface called tclAtFS. On top of these there will be an integration layer implemented in Tcl (see fig. 4).

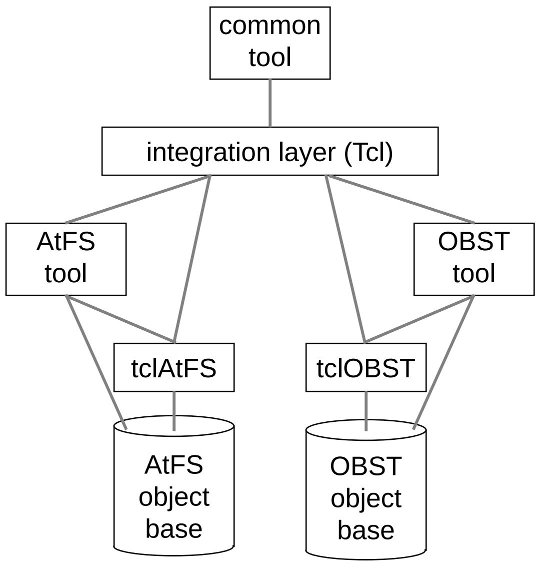

Figure 4 heterogeneous database integration with tclOBST

4. Conclusion

We presented the architecture of tclOBST, a Tcl interface to the OBST object-oriented database management system, as well as some experiences. The interface significantly raised the usability of OBST by considerably reducing the oriented way - as required by Tcl: The object structure is traversed by invoking the attribute access methods until a scalar value is reached. This value can then be converted into its string representation and vice versa.

A prerequisite for implementing such a traversal in a generic fashion is the meta protocol of OBST which does not only make the description of OBST types available, but does also provide access to the associated code, i.e. the implementation of methods and conversion operations for scalar types.

2.2 Architecture of the Interface

The intended interface has to be complete in the sense that any data described in the OBST data model can be handled. It should furthermore be generic in that no recompilation of tclOBST is required to handle newly defined OBST types. Instead, it should be sufficient to link with the code associated to those types.

tclOBST was built in a three layered approach: the first layer is made up of the OBST library, the second layer is a C interface which embedds OBST in C, and the third layer is the actual embedding in Tcl. Here, we only present the most important aspects of this interface. A detailed description can be found in [9].

The C interface basically provides an opaque handle type[2] which can refer to any OBST object, two functions to invoke methods specified by name, and a means to capture errors raised by OBST - in particular type errors.[3] Furthermore, there are accessor functions to the part of the OBST library for which there is a C++ interface but which is not expressed in the OBST data model. Additional support is provided for the data container classes of the OBST library such as Set, List, ...

The C interface is as well generic as complete in the above sense. It is generic, since there is a fixed set of functions and types which suffices to handle any data which is defined in the OBST data model.

These properties of the C interface carry over to the Tcl embedding which defines operations to convert between C object handles and Tcl object handles. These Tcl object handles are fixed size identifiers which do not incorporate process specific data and may hence be used across process boundaries. The accessor functions and container class support of the C interface are mirrored by Tcl commands, in particular a loop command for scanning such container objects.

Furthermore, a Tcl command for invoking methods was defined which transparently handles the appropriate conversion of method arguments and result values: class instances are given and returned, respectively, as object handles, and scalar values in their string representation (e.g. ”6” for an integer type). Finally, a natural syntax is achieved by transparently binding object handles to this Tcl command, i.e. object handles become Tcl commands themselves. Hence, the application of a method ”get_sources” to an object handle ”$module” and the following application of the method ”card” on the resulting object (cf. fig. 2), read as:

```
set no_of_sources [[$module get_sources] card]
```

Fig. 3 shows an architectural overview of a tclOBST application: the shaded area represents the code of a customized Tcl interpreter which is capable of interpreting a script of Tcl(OBST) commands. It consists of a standard part which will appear in all such interpreters - namely the core OBST, tclOBST and Tcl/Tk libraries, plus the additional code which is associated to the (OBST) data types used in the application. tclOBST is built on top of the core OBST library whereby access to the application specific code and its data is provided by the OBST (run time) system and by interpreting the meta objects holding the description of application objects. As symbolized by the dimmed script, tclOBST will be mostly transparent to the writer of an application script which will think and program in terms of application objects.

2. This opaque C handle is a fixed size byte array.
3. A type error will in particular be raised and caught when trying to invoke a method which is not defined for a given object.

ble, but yet general mechanisms for tool integration in STONE which are applicable to as well OBST based as other tools.

2. The tclOBST Interface

Starting from the above listed experiences we looked for a way to provide flexible, interpretative access to an OBST database. An existing interpreter for a lambda calculus based language [7] proved to be dissatisfactory in terms of user friendliness, extensibility and integration in a X/UNIX environment.

2.1 Ingredients

Tcl/Tk was chosen as the technical basis for our solution because its properties matched our requirements: Tcl [4] works well together with UNIX/C/C++ based applications and the functionality of such an application can be easily integrated into Tcl as soon as the application's data can be represented as strings and its functionality can be expressed by the combination of a few basic commands. A C/C++ or UNIX shell programmer will accustom fast to the Tcl language and the language is lean in providing just the basic processing capabilities which the experience of UNIX shell programming has shown to be sufficient for the applications we had in mind: standard control structures, string handling, basic arithmetics, file handling, and program execution. A particular asset of Tcl/Tk is of course the embedded X toolkit Tk [5], whereby the existence of powerful interactive interface builders [1] was probably even more attractive to us than Tk itself. Tcl and in particular Tk provide good performance. They exhibit a very dynamic nature in that code and most data elements can be accessed and modified at any time. A very important criterion was that Tcl/Tk and most of the published additions can be used without any licensing restrictions, since any outcome of STONE is to be distributed in the public domain. Last not least, Tcl/Tk and the so far published extensions are used and constantly pushed forward by an active user community.

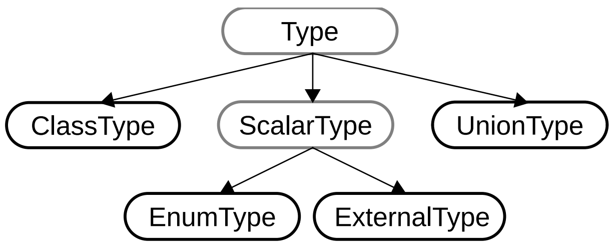

Figure 1 OBST type hierarchy

The OBST data model is hybrid by differentiating between class types and basic, so-called scalar types (see fig. 1). Class types comprise both structural and behavioral aspects, i.e. attributes and methods. Attribute domains may be class types as well as scalar types whereby an attribute with a class domain will contain object identifiers as values. Thus any, even complex data structure is recursively built from scalar values and object identifiers.

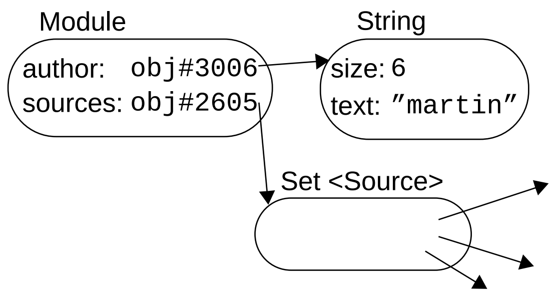

Figure 2 OBST data structure example

For an example consider figure 2 which contains a subset of a hypothetical OBST data base: the Module object contains two attributes author and sources with class type domains String, and Set<Source>, respectively. Both attributes hold object identifiers. The referenced String object contains the attribute size with an integer domain type. This integer type is an external scalar type according to the OBST data model (see fig. 1).

All accesses to the state of an object, i.e. reading and writing an attribute, are performed by invoking attribute access methods generated by the OBST system. OBST provides furthermore a standard conversion between scalar values and string representations. This forms the basis for processing OBST object structures in a string-

Interfacing an Object-Oriented Database System from Tcl

Dietmar Theobald

Forschungszentrum Informatik (FZI)
Haid-und-Neu-Straße 10-14
D-76131 Karlsruhe
Germany
email: {theobald,stone}@fzi.de

May 1993

Abstract

We present an extension to Tcl which realizes a generic interface to an object-oriented database system.[1] This interface provides flexible access to the database system by drawing on Tk/Tcl's ability as a scripting language promoting rapid prototyping and the development of graphical user interfaces. Ongoing work investigates the suitability of the interface as a means for application development and tool integration.

The first chapter introduces the context of this work and summarizes its starting point. Then the implementation and the architecture of the interface are described. Usage experiences and an outlook on future work concludes the presentation.

1. Starting Point

The Forschungszentrum Informatik (FZI) participates in the project STONE („Structured and Open Environment“) which aims at the development of a software engineering environment for the educational domain [3]. A major contribution of FZI to this project is the object-oriented database management system OBST, a core component which serves as the main persistent store for the tools in a STONE environment.

OBST features an object-oriented data model which supports core concepts found in major object-oriented languages [10]. OBST is targeted at a UNIX environment where workstations are coupled in a local area network. The system employs C++ as host language for writing method implementations. Applications access the database via a programming interface, i.e. they are implemented in C++ and linked with the database library.

For larger applications where the application domain is well understood and/or efficiency is at premium, C++ based development is the method of choice. This does not necessarily apply to prototypical implementations such as concept studies or for applications where much emphasis is placed on the user interface part. Furthermore, in case of small applications performing e.g. administration tasks the overhead in terms of disk space and development effort might not merit their realization as a C++ program.

Another area of application development demanding specific support is the integration of OBST based tools into a possibly heterogeneous environment: There should be simple and flexi-

1. The work reported here is funded by the German Ministry of Research and Technology (BMFT) as part of the project „STONE“.