Xtrace: a high-level extension of Tcl-trace

Florian Murr
Siemens AG - Corporate Technology
Otto-Hahn-Ring 6
D-81739 Munich, Germany
+49 (0)89 636-44949
florian.murr@siemens.com

Manfred Burger
Siemens AG - Corporate Technology
Otto-Hahn-Ring 6
D-81739 Munich, Germany
+49 (0)89 636-43209
manfred.burger@siemens.com

ABSTRACT
In this paper two Tcl-packages written in pure XOTcl [3] are presented. The first "Xcom" is yet another socket-communication package; the second "Xtrace" uses Xcom to provide an observer command across multiple processes called "`xtrace`" that is modeled along the lines of the well-known Tcl "`trace`" command, but much more high-level.

Keywords
Xtrace, Tcl-trace, observer-pattern, distributed applications, distributed user interfaces, model based.

1. INTRODUCTION

Tcl has much of the famous "observer pattern" built into the language via the well-known Tcl-"`trace`" command.

The *observer-pattern* consists of some entity that might be accessed and any number of observers who want to be informed about these accesses.

Tcl- "*`trace add variable`*" provides observer-functionality at core language level in a slightly modified way:

Any trace-callback-command, i.e. "observer", does not only get informed, but can intercept any attempted access and decide whether to let it pass, to modify, or to block it. This reflects the very low-level and immediate nature of Tcl's trace command.

Being so low-level "`trace`" is best suited for extending Tcl with new commands or control constructs.

The observer pattern on the other hand is often used in a distributed context and is much more high-level in spirit.

If an observer resides in a different process than the observed variable, intercepting an attempted access is out of question. Even getting informed of every change of the observed variable is sometimes too demanding.

Within a distributed observer pattern, consistency matters more than immediate callbacks! - Therefore Xtrace favors high-level comfort and consistency over direct access.

Let's first have a look at the mindset that spawned Xtrace and than at a simple example.

2. BASIC SETTING

The basic distributed setting we had in mind when developing Xtrace consisted of:

- A "model" composed of XOTcl objects that reside at the server.
- Multiple "clients" that are observers-of / actors-on this model, residing in different processes or even on different machines.

The clients are able to change the model, through setting of variables, or through method calls and these changes get propagated back to the clients, assuring that all clients mirror the current state of the model. The process in which the observed variables reside is henceforth called the "*model*" (or "*server*"). [This notion is not completely correct, since xtrace allows in principle that the model- objects / variables are distributed among multiple processes, but we not yet made use of that possibility.]

The "*clients*" ("*observers*") are typically interested in state-transitions of the model. Such a transition of one consistent state to the next often involves the change of more than one variable. A high-level feature like xtrace will therefore propagate the cumulated delta after all the simple variable changes are complete. Fortunately Tcl's event-loop gives a handle to find moments when a consistent state should have been reached.

2.1 EXAMPLE 1

Consider some facility with several workplaces.

Appliances to be checked travel through these workplaces and get different tasks performed upon them. Every workplace has a "current appliance" variable and every appliance has a variable for every task to be performed.

So in our example the situation at the model might be something like this:

```
% ::workplace7 set currAppliance
::app12
% ::app12 set pTest
::task42
% ::task42 set values
{123 234 345}
% ::task42 set state
ok
% ::task42 set remarks
{}
% ::task42 set worker
::person773
% ::person773 set firstName
John
% …
```

Some monitor may show the “id”, “state” and “remarks” variables of the current task on the current appliance at that workplace.

With Xtrace one may make use of

```
obj xtrace add ?-soft? chain vars cmd
```

and code this in a single command-line:

```
::workplace7 xtrace add \
        {currAppliance pTest} \
        {values state remarks} \
        [list ::monitorCallback]
```

2.2 WELL-KNOWN OBJECTS

Xtrace works in the object-oriented setting of XOTcl and any use of it is directed to some Xtrace-participating XOTcl-object.

Objects participate in Xtrace, when they provide a “toModel” method, which returns the peer-object, to contact its model. [There are some classes that may be used, either as base-class or as “mixin” that provide default implementations for clients and server.]

Some of these objects should be known at coding-time to any client of an application, to kick-start Xtrace-communication. These are the so called “*well-known objects*”. In example 1 “::workplace7” is treated as such a well-known object.

These objects only “live” fully on the server, but the client at least knows that on the server there exists some object with this name. – Usually the client also knows the type (classes) of these objects. Other objects may not be known at coding-time, but get delivered to the client during runtime. The client then could use another “xtrace add” to listen to variables of those objects, too.

The client version of an (well-known) object is just a stub. It will only have those variables that get mentioned in “xtrace add”. These variables will be created as needed. The stubs usually do not have the functional implementations for methods either. Methods just send there call to the server.

[Actually there are some Meta-classes of Xtrace that provide these functionalities. If you use

- “instproc” the method is implemented on the server and the client,
- “instprocModel” the method exists only on the server,
- “instprocDist” the method gets distributed in the sense above.

But to get the idea, we usually abstract away such technicalities in this paper.]

The model version of a well-known object is thought to be the real object, be fully functional and have all the necessary variables and methods.

2.3 VARIABLES

All the chain- and vars- variables exist in the client and on the server. Just setting a variable (on the client or server) suffices to let the change automatically be distributed. Since communication takes place “after idle”, one may change many variables until the next consistent state is reached and not until then the dissemination of the cumulated changes starts.

2.3.1 “Chain” variables

The "*chain*" in example 1 is the list "`{currAppliance pTest}`". Such a chain consists of variable names. These variables are supposed to hold object-names as values or to be empty. The idea behind "*variable chains*" is that starting with an already known object (e.g. "::workplace7" which is known à priori) which has a variable whose name is the first element in the chain ("currAppliance"). This variable has some object-name as value, which holds the next variable and so on.

2.3.2 “Vars” variables

The “*vars*”-list contains names of variables of the last object in the chain. Normally these variables are XOTcl- class attributes of some application specific class. They are not restricted to specific value-types, as the chain-variables are. Our practical experience has lead us to quite often use “`dict`” values [2]. (We used the Tcl8.4 backport of “`dict`”.)

2.3.3 Slow clients

“`xtrace add`” has an optional flag “-soft”, that is intended for slow clients. The client may be considered slow, because the connection, or the machine of the client are slow, or because the client has some time consuming task to perform every time he gets informed by xtrace.

As a guiding example consider some rapid changing image, i.e. some pseudo-movie and clients with different speeds trying to be in sync, i.e. to show the same image. The variable “currImage” on the server changes with some predefined frame rate. Since the server may not show the image at all, but only hold the “currImage” variable, speed is no concern there. If a client wants to show the image, that may take some time (especially with Tk). Without precautions in this respect, every change would be sent to this client. In other words, messages arrive faster on the socket, then the client can get them off and the socket would be jammed.

Here the “-soft” option comes to the rescue.

If a variable gets traced “soft”, then the server just sends an “are-you-ready” message to the client, whenever the variable changes. – The client responds when he is idle. When this response arrives at the server, the up to then cumulated changes get sent to the client. On the up-side the client avoids clogging of his socket and stays in sync, on the down-side, he might miss some intermediate changes and skip some of the image to be shown.

2.4 CALLBACK INTERFACE

When the callback-command “*cmd*” of “`xtrace add`” gets called in the client the name of a “parcel”-object gets appended to the command. This is an object of type “XtrcParcel” which provides methods to comfortably extract all the information concerning the changes that have happened.

Sometimes one is not really interested in getting the callback, but is content when the variables in chain and vars are kept up-to-date. This behavior is achieved by giving an empty callback-command to “`xtrace add`”.

In either case, should any variable in "chain" or "vars" have its value changed, then the change will be distributed to the clients that are interested in this variable; i.e. all the clients that have an “xtrace” registered for this variable and those whose chain happens to contain the variable in question.

On the client all the variables that contribute to any of the registered xtraces get mirrored from the server and get updated with the current values before the callback command is called. Additionally there is an interface on “parcel” that allows retrieving the previous value of any of the variables in “chain” or “vars”, so that the transition “oldValue→newValue” for any of these variables is available during the callback call.

Should for example "currAppliance" be set to "{}" the values of "pTest" and its vars are no longer valid too. All changes in the chain gets distributed and the callbacks called. The clients also need no longer listen for changes on variables of the previous “pTest”. Or should another appliance, with some other “pTest”, take place in the “::workplace7”, the new corresponding values get distributed. This is all handled automatically in Xtrace.

3. SYNOPSIS

The following commands are provided by Xtrace:

```
obj xtrace add ?-soft? chain vars cmd
obj xtrace remove chain vars cmd
obj xtrace info chain vars cmd
```

“obj” may be any well-known XOTcl object for Xtrace or an object which is actually in some chain traced by xtrace.

“-soft” allows for slow clients.

"chain" is a list of variable names. The corresponding variables must contain an object name or be empty.

"vars" is a list of variable names, of the last object in the chain.

"cmd" is a Tcl or XOTcl callback command.

Further details of the meaning of the different arguments of “`xtrace add`” are described in chapter 2.

“`xtrace remove`” and “`xtrace info`” are most of all self–explanatory. We do not describe it here.

4. REMARKS

Xtrace goes especially well in combination with the "`dict`" command. Since a dict is a value, it may be the value of any of the "vars" in an xtrace. Therefore one can bundle values in a dict at the model, to facilitate observation.

The Tcl event loop has extraordinary power in synchronizing asynchronous calls! Since xtrace (interprocess-) communication takes only place when the process get "idle", some quasi-simultaneous changes from different clients get synchronized by the Tcl-event-loop. When the process becomes "idle" again, xtrace assumes that a consistent state has been reached, that gets distributed to the clients. - It is of course possible to program against this assumption, but with only moderate consideration it is possible to develop very stable and consistent distributed applications.

The xtrace interface is very high-level, since the user just focuses on the logical "chain of variables" and all the work of keeping the corresponding values up to date gets done by the xtrace package.

Another feature that makes Xtrace high-level is the allowance of slow clients. In that case Xtrace doesn't force the frequent changes upon the client (which would clog the communication socket), but informs that there has been some change and waits then to be requested by the client to send the current cumulated changes.

In very complex applications one even might consider not to have only one model-process, but to distribute the model over multiple machines. This should be no problem to xtrace, since every one of the "well-known" objects knows how to contact its model and these may well be in different processes for different objects.

5. COMPARISON OF “trace” AND “xtrace”

Xtrace is rather similar in spirit to the low-level Tcl-“`trace add variable`”. Here is a list of similarities and differences between "trace" and "xtrace:

- “trace” is low-level monitoring, intended to be able to extend Tcl with new infrastructure. That is exactly what xtrace uses Tcl-trace for!
- “xtrace” is intended for high-level observing. The new structures support monitoring of consistent changes of different objects and variables.
- "xtrace" is written in XOTcl, whereas "trace" is a Tcl core feature.
- "xtrace" allows to observe more than one variable in one call, it even allows so called “*variable-chains*” to be observed.
- "xtrace" does not call the callback command, the very moment the variable is accessed, but distributes the cumulated changes when the process becomes idle.
- "xtrace" specializes on "write"-access to variables. Read-only access of variables is not in its scope.
- "xtrace" may observe variables in objects in different processes.
- "xtrace" is bidirectional. Variables get mirrored in the observing client and changes get propagated in both directions.
- "xtrace" callback gets delivered a "parcel" that contains both: the old value and the new value of the variable.
- "xtrace" communicates using the "xcom"-package that uses "sockets" for communication.
- "xtrace" allows clients that are quite slow to participate. ("-soft" option)
- "xtrace" allows the callback to be empty. - Only the variable values keep getting synchronized.
- "xtrace" allows dynamically to extend the observed objects. The chain of variables need not exist.

6. THE “Xcom” PACKAGE

Xcom is yet another socket communication package, similar to the well known “comm” package [1].

Unfortunately "comm" had some limitations that lead us to reimplement its functionality using XOTcl. (Some conditions

dubbed "race-conditions" in comm are quite natural in xtrace and handled gracefully there.)

Xcom allows file-transfers to be triggered and a callback is executed once the file-transfer is complete.

The request for some file lets the called partner create a temporary file-server for this request and reply to the requesting-partner, the host and port of this file-server. The requesting-partner gets the file from the server and the callback command is evaluated. - This works in both directions, from Xcom-server to Xcom-client, or from client to server.

Xcom has some utility functions built-in for xtrace.

XOTcl allows to "mixin" classes dynamically, which makes it very easy to account for clients "speaking different languages". One client may speak "Tcl", just plain Tcl commands, another client might use "XML", say a Flash-UI using ActionScript-XMLsocket, still another client wants encrypted messages. Xtrace just mixes in the appropriate encoding classes. - As a result, Xtrace can communicate with each client in its preferred communication language.

7. REFERENCES

[1] Comm., http://tcllib.sourceforge.net/doc/comm.html

[2] dict, http://www.tcl.tk/man/tcl8.5/TclCmd/dict.htm

[3] XOTcl, http://www.xotcl.org/