

XOTCL– an Object-Oriented Scripting Language

Gustaf Neumann

*Department of Information Systems
Vienna University of Economics
Vienna, Austria
gustaf.neumann@wu-wien.ac.at*

Uwe Zdun

*Specification of Software Systems
University of Essen
Essen, Germany
uwe.zdun@uni-essen.de*

First European Tcl/Tk User Meeting, 15/16th June 2000

Overview

◆ **XOTcl = Extended Object Tcl**

◆ **XOTcl is freely available from:**

<http://nestroy.wi-inf.uni-essen.de/xotcl>



◆ **Outline:**

- Scripting and object-orientation,
- XOTcl high-level language constructs,
- Example: design pattern-based design of an XML interpreter,
- xoComm HTTP implementation: performance comparison with Apache.

Tcl-Strengths

Important Ideas in Tcl:

- ◆ **Fast & high-quality development through component-based approach**
- ◆ **2 levels: “System Language” and “Glue Language”**
- ◆ **Flexibility through . . .**
 - dynamic extensibility,
 - read/write introspection,
 - automatic type conversion.
- ◆ **Component-Interface through Tcl-Commands**
- ◆ **Scripting language for glueing**



Motivation for XOTcl

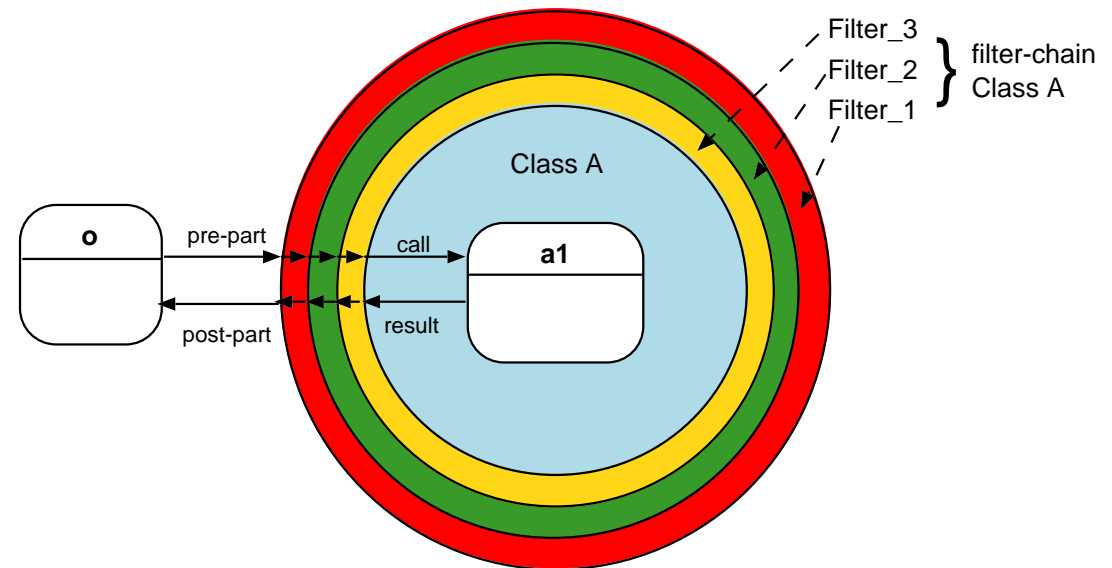
- ◆ **Extend the Tcl-Ideas to the OO-level.**
- ◆ **Just “glueing” is not enough! Goals are . . .**
 - Architectural support
 - Support for design patterns (e.g. adaptations, observers, facades, . . .)
 - Support for composition (and decomposition)
- ◆ **Provide flexibility rather than protection:**
 - Introspection for all OO concepts
 - All object-class and class-class relationships are dynamically changeable
 - Structural (de)-composition through *Dynamic Aggregation*
 - Language support for high-level constructs through powerful interceptors (*Filters* and *Per-Object Mixins*)

Filters

- ◆ A **filter** is a special instance method registered for a class C. Every time an object of class C receives a message, the filter is invoked automatically.

- ◆ Three parts, each optional:

- pre-part,
- call to **next**, invokes:
 - filter-chain,
 - actual called method.
- post-part.



- ◆ Filter-inheritance.

Example: Simple Filter

```
Class A                                ;# Class Definition
A a1                                   ;# Instance a1 of A

A instproc Filter-1 args {            ;# Filter instance method
  puts "pre-part of Filter-1"
  next
  puts "post-part of Filter-1"
}

A filter Filter-1                      ;# Filter registration

a1 set x 1                             ;# Method invocation
```

Applications: Trace facility, Composite Pattern, Proxy Pattern, Observers . . .



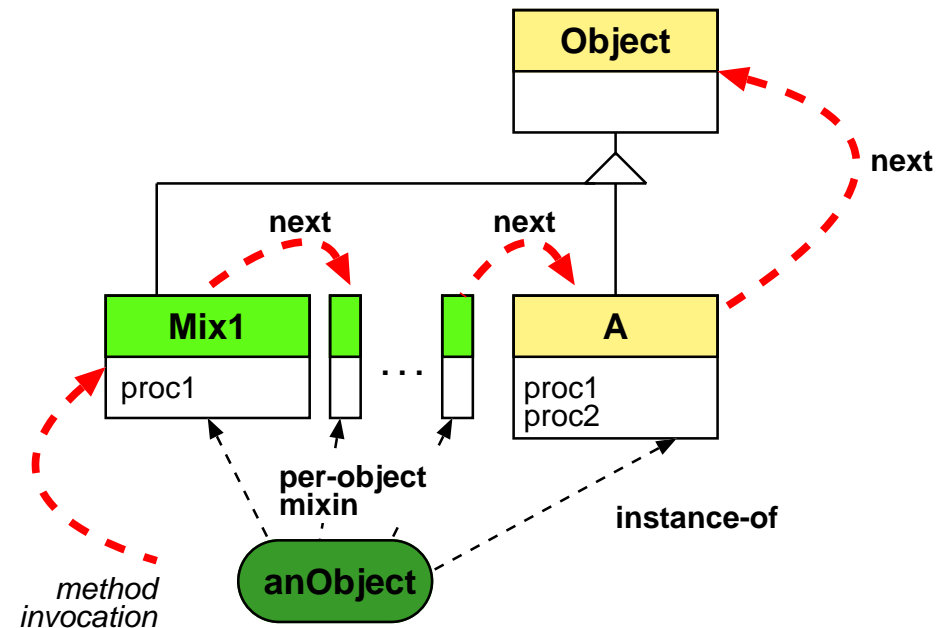
Per-Object Mixins

- ◆ A **per-object mixin** is a class which is mixed into the precedence order of an object in front of the precedence order implied by the class hierarchy.

Motivation:

- ◆ Model behavior of individual objects (decorator).
- ◆ Handle orthogonal aspects not only through multiple inheritance.
- ◆ Intrinsic vs. extrinsic behavior, similar to roles.

Applications: timing, statistics, persistence, life-cycle, chain of responsibility, adapter



Example Code for Per-Object Mixins

```
Class A                                ;# Class definition
A instproc proc1 {} {                 ;# Method definition
  puts [self class]; next
}
A instproc proc2 {} {                 ;# Method definition
  puts [self class]; next
}
Class Mix1                              ;# Class definition
Mix1 instproc proc1 {} {              ;# Method definition
  puts [self class]; next
}
A anObject                              ;# Instantiation of class A
anObject mixin Mix1                    ;# Mixin registration

anObject proc1                          ;# -> results in output "::Mix1 ::A"
anObject proc2                          ;# -> results in output "::A"
```



Dynamic Object Aggregations and Nested Classes

- ◆ **Nesting though namespaces: Classes and objects in XOTcl can contain other classes/objects.**
- **Dynamic Object Aggregation** resembles Part-of relationship in a dynamic and introspective fashion.
- **Nested Classes** reduce interference of independently developed program structures.
- ◆ **Class nesting and aggregation semantics are handled through XOTcl object system (including built-in methods for deep copy and deep move).**



Example Code: Nested Classes/Dynamic Object Aggregation

```
Class Agent                                ;# Class definition
Class Agent::Head                          ;# Nested classes
Class Agent::Body

Agent instproc init args {                ;# Constructor aggregates two
  ::Agent::Head [self]::head             ;# objects dynamically
  ::Agent::Body [self]::body
}

Agent myAgent                              ;# Object creation

puts "Children: [myAgent info children]"  ;# Output: head body

myAgent::head destroy                      ;# Agent loses his head

puts "Children: [myAgent info children]"  ;# Output: body
```



Further Functionalities provided in XOTcl

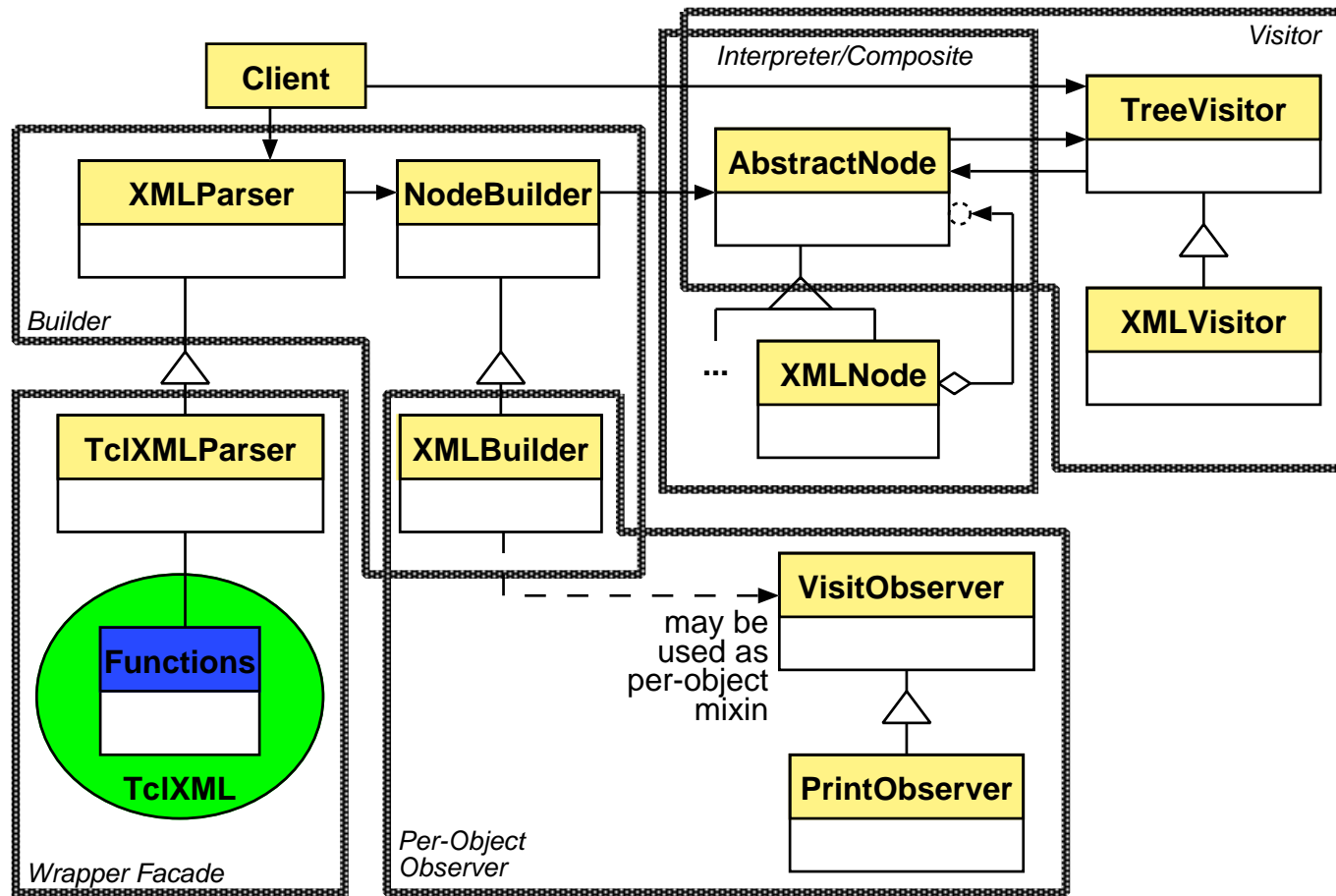
- ◆ **Assertions reduce interface and reliability problems caused by dynamic typing:**
 - Design by contract: invariants and pre-/post-conditions for methods,
 - per-class and object-specific assertions.
- ◆ **Meta-Data enhances self-documentation of objects and classes.**
- ◆ **Automatic Name Creation with `autoname`.**
- ◆ **Abstract Classes,**
- ◆ **Parameters.**



Example: XML Parser/Interpreter

- ◆ Constructs a composite object structure from XML documents
- ◆ OO-implementation using design patterns, based on TclXML, around 120 lines (including example visitors and reusable pattern)
- ◆ Changeability and Adaptability through:
 - dynamics,
 - introspection,
 - patterns in hot spots,
 - interceptors per-object and filter,
- ◆ **Patterns:** Wrapper Facade, Builder, Composite, Interpreter, Visitor, Observer, ...
- ◆ Extensibility through new visitors, observers

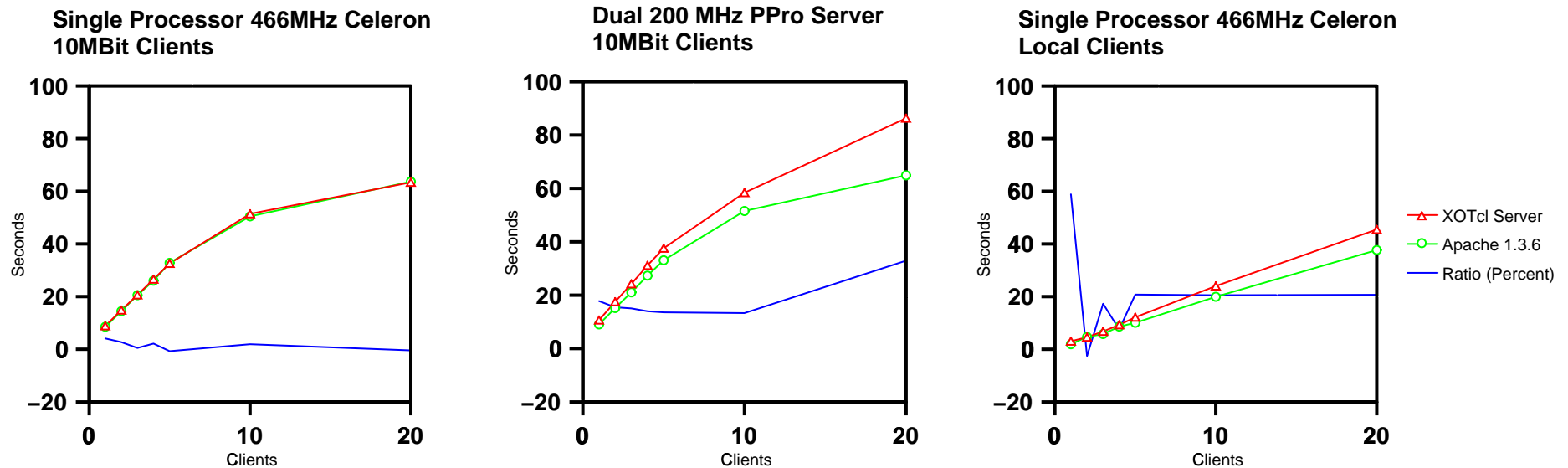
Partial Design of the XML Parser/Interpreter



Assessments

- ◆ **size 73 lines (including two more visitors),**
- ◆ **+ 22 lines for the Wrapper Facade and 25 lines for the Composite,**
- ◆ **Reuseable Composite implementation and reuseable TclXML wrapper,**
- ◆ **Changeability and Adaptability through:**
 - dynamics,
 - introspection,
 - patterns in hot spots,
 - interceptors per-object and filter,
- ◆ **Extensibility through new visitors.**

Speed Comparison: XOTcl based HTTP Server vs. Apache



- ◆ Basic functionality of HTTP/1.1 in around 250-400 lines of XOTcl code
 - ◆ 1-20 simultaneous client sessions issuing each 76 HTTP requests.
- Modern CPUs are fast enough to execute even complex applications in object-oriented scripting language with sufficient speed.

Summary and Conclusions

- ◆ **XOTcl provides high-level language constructs for software composition,**
- ◆ **tailored for the use with scripting applications:**
 - dynamics,
 - (read/write) introspection,
 - flexible glueing of (preexisting) components.
- ◆ **Combination of: scripting, object-orientation, and high-level language constructs**
 - ⇒ Flexible composition of software systems.
 - ⇒ Coping with changes at runtime/design time.

More XOTcl Material

- ◆ Gustaf Neumann, Uwe Zdun: *Filters as a Language Support for Design Patterns in Object-Oriented Scripting Languages*, Proceedings of the 5th Conference on Object-Oriented Technologies and Systems (COOTS '99), San Diego, May 3-9, 1999.
- ◆ Gustaf Neumann, Uwe Zdun: *Enhancing Object-Based System Composition through Per-Object Mixins*, Proceedings of Asia-Pacific Software Engineering Conference (APSEC'99), Takamatsu, Japan, December 6-10, 1999.
- ◆ Gustaf Neumann, Uwe Zdun: *Towards the Usage of Dynamic Object Aggregations as a Form of Composition*, Proceedings of Symposium of Applied Computing (SAC'00), Como, Italy, March 19-21, 2000.
- ◆ Gustaf Neumann, Uwe Zdun: *High-Level Design and Architecture of an HTTP-Based Infrastructure for Web Applications*, Word Wide Web Journal, Baltzer, early 2000.
- ◆ More on <http://nestroy.wi-inf.uni-essen.de/xotcl>