tComLabs

An introduction to TOOP

A Tcl extension for OO Programming

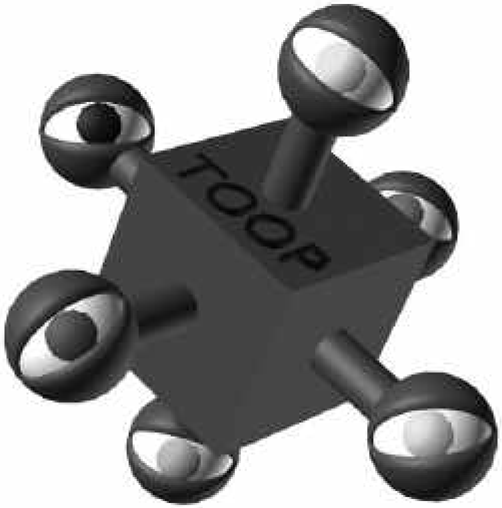
Overview

- What is TOOP
- TOOP Features
- Pro’s and con’s
- Conclusion

TOOP

- **What** :

  A Tcl extension (Tcl only 8.0 – x.x) that implements the object orientation paradigm to a great extent.



Introduces in Tcl:

A new proc TOOP
A library structure
A new file extension .tclClass

TOOP Space

TOOP Spaces are Tcl interpreters where the TOOP.tclClass file has been sourced.

```
% source TOOP.tclClass
```

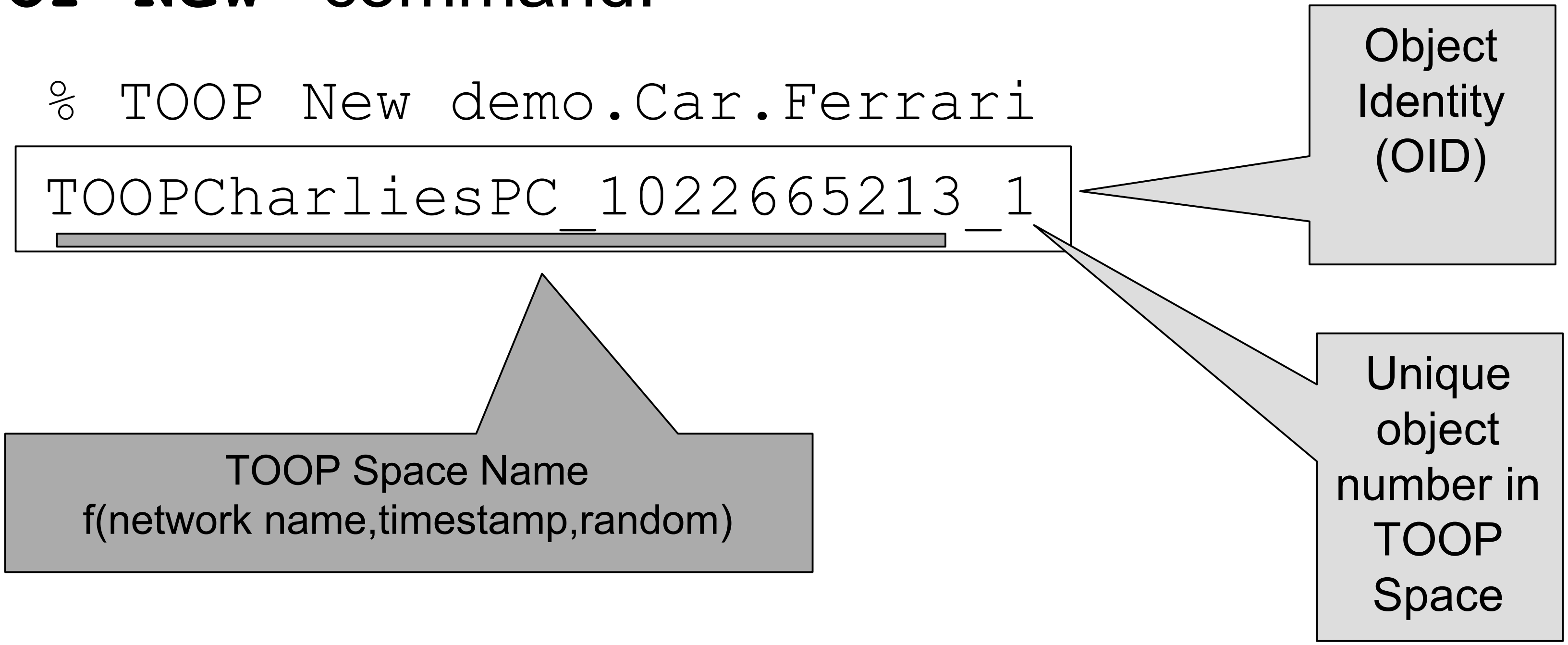
TOOP Spaces :

- Know where the TOOPLibrary can be found.
- Take care of creation of new TOOP.Objects
- Take care of the sourcing of tclClass files
- Take care of the duplication of objects
- Take care of serialization of objects
- Manage communication with other TOOP Spaces
- Have a unique name in time and space

TOOP Objects

New TOOP objects are made using the **`TOOP New`** command:

```
% TOOP New demo.Car.Ferrari
TOOPCharliesPC_1022665213_1
```



Object Identity (OID)

Unique object number in TOOP Space

TOOP Space Name
f(network name,timestamp,random)

TOOP Object State

The Object array, a global array named OO

`::OO($OID,mySpeed)`

A Tcl array (hashtable) that stores all Object State in a TOOP Space

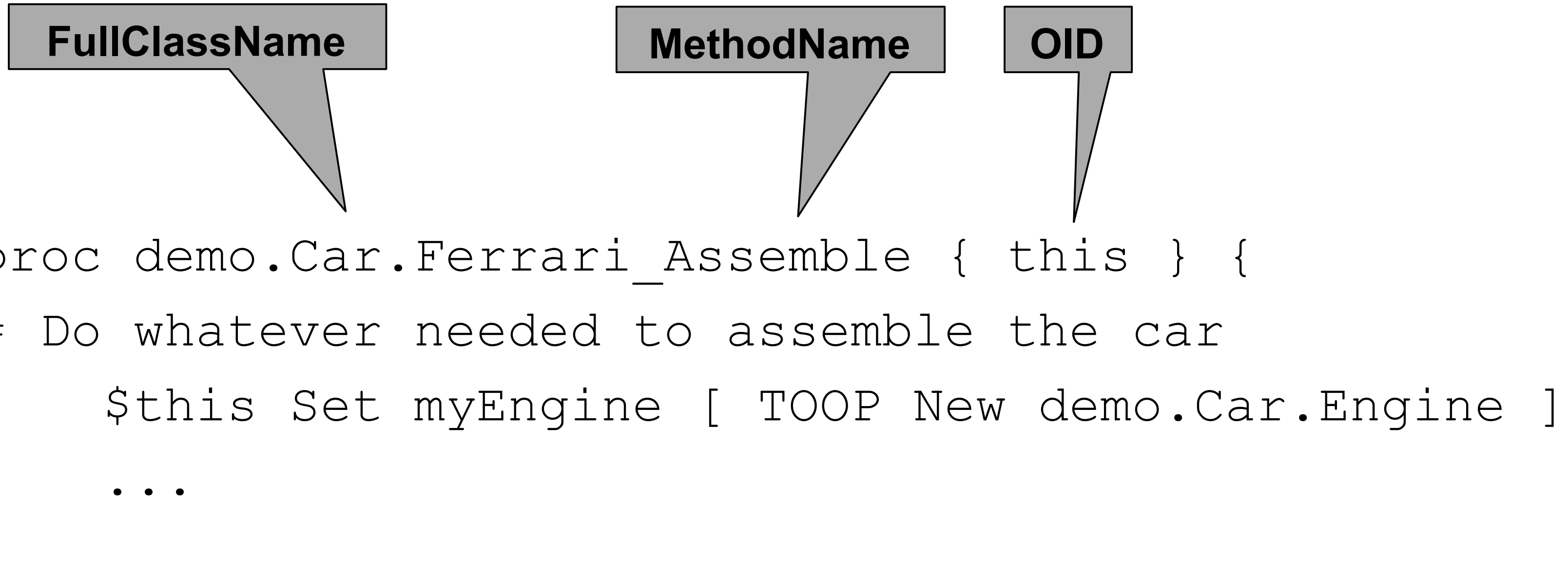
Changing Object State:

```
%set F [ TOOP New demo.Car.Ferrari ]
%$F Set mySpeed 20    OR    %set ::OO($F,mySpeed) 20
20                          20
%$F Get mySpeed       OR    %set ::OO($F,mySpeed)
20                          20
```

REMARK: there are NO notions of private and public data members!

TOOP Object Behavior - Methods

Method definitions are ordinary Tcl proc’s:

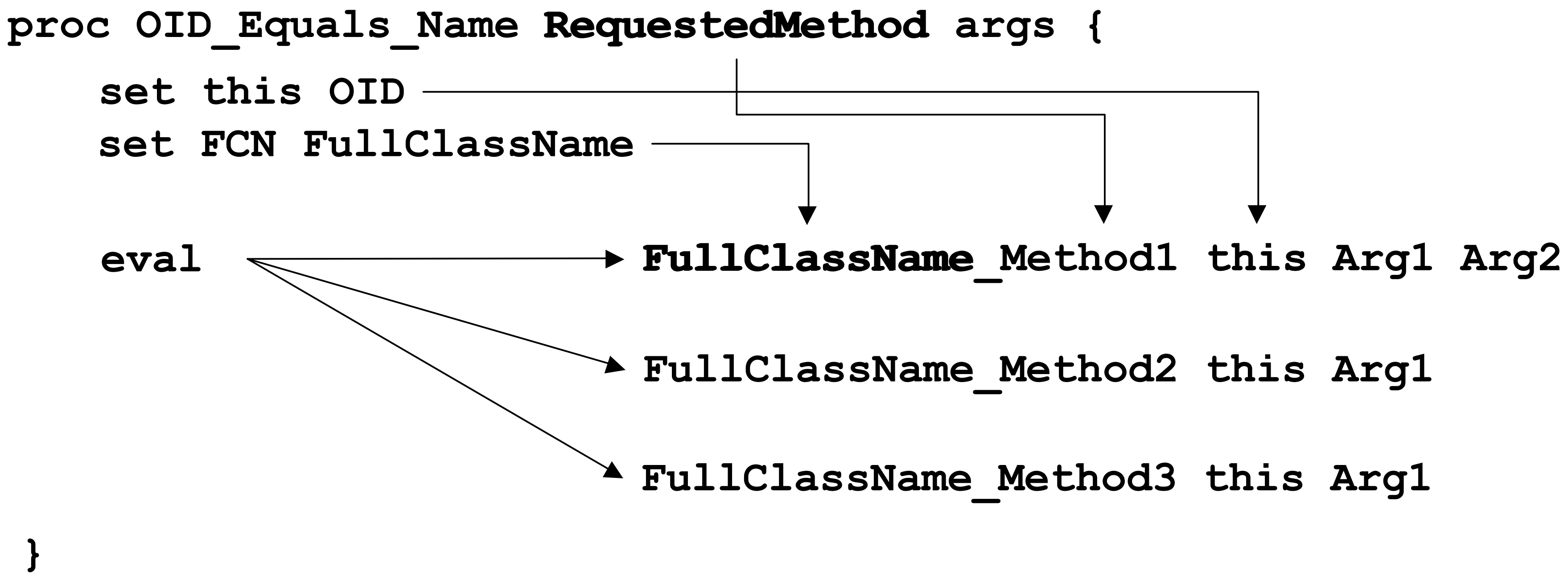


```
proc demo.Car.Ferrari_Assemble { this } {
# Do whatever needed to assemble the car
    $this Set myEngine [ TOOP New demo.Car.Engine ]
    ...
}
```

Note: the ‘this’ parameter must be provided, its value is the OID

TOOP Object Communication 1/2

Communication with an object happens through messages that are dispatched by its MDP (Message Dispatching Procedure)



```
proc OID_Equals_Name RequestedMethod args {
    set this OID
    set FCN FullClassName

    eval          FullClassName_Method1 this Arg1 Arg2

                  FullClassName_Method2 this Arg1

                  FullClassName_Method3 this Arg1

 }
```

TOOP Object Communication 2/2

Communication with objects results in
Object Behavior:

```
% set Obj [ TOOP New demo.Car.Ferrari ]

% $Obj Assemble
```

Speed loss by 5 times over ordinary Tcl due to MDP

```
   OR

% demo.Car.Ferrari_Assemble $Obj
```

No Speed loss over Tcl
(MDP is not used)
Flexibility is gone

TOOP Classes

Classes (.tclClass) files are templates for creation of TOOP objects that describe :

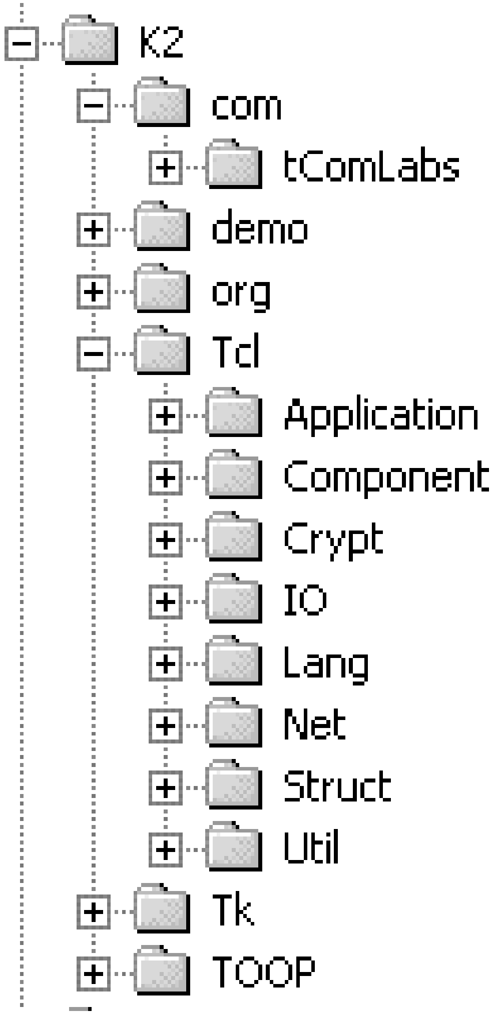
- (!) FullClassName e.g.: `Tcl.Application.vTcl.Main`
- Methods
- Static Fields
- Package Name

.\test\myClass.tclClass

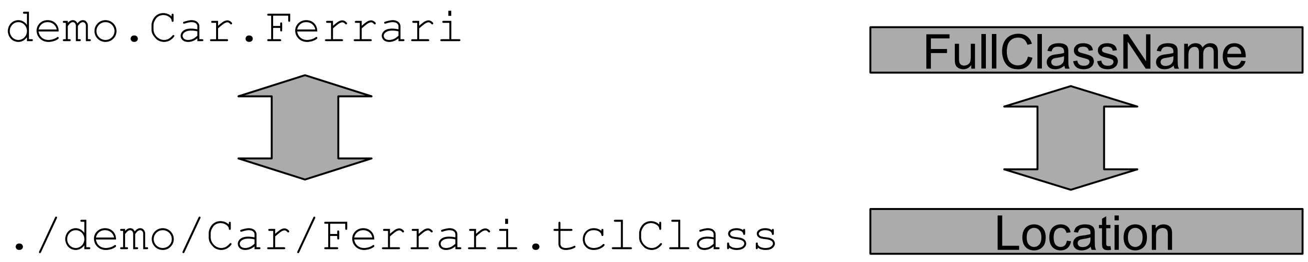
```
TOOP.Class Extend \
TOOP.Object test.myClass  {

TOOP.Class Method SayHello { }  {
      puts “ by the Beatles”
}

}
```

.tclClass files are pure .tcl files
They are evaluated in the interpreter in the end.

TOOP Library



- K2
  - com
    - tComLabs
  - demo
  - org
  - Tcl
    - Application
    - Component
    - Crypt
    - IO
    - Lang
    - Net
    - Struct
    - Util
  - Tk
  - TOOP



`demo.Car.Ferrari`

`./demo/Car/Ferrari.tclClass`

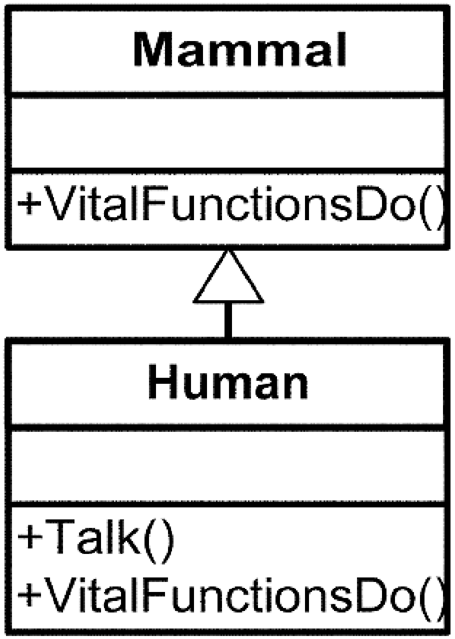
Root of the tree is defined as location of the TOOP.tclClass file

Auto-source of class definitions:

- Find file in local [pwd] directrory
- Find file in TOOP Library
- Find file using the self defined ClassLoader (option)

TOOP Inheritance

The story of parents and children... Humans extend from mammals



Mammal

+VitalFunctionsDo()

Human

+Talk()
+VitalFunctionsDo()

```
proc Mammal_VitalFunctionsDo { } {
   puts “Eating, breathing, ...”
}

proc Human_VitalFunctionsDo { } {
   return [ Mammal_VitalFunctionsDo ]
}

proc Human_Talk { } {
   puts “We say Hello”
}
```

Single inheritance and multiple (interfaces) are possible in TOOP

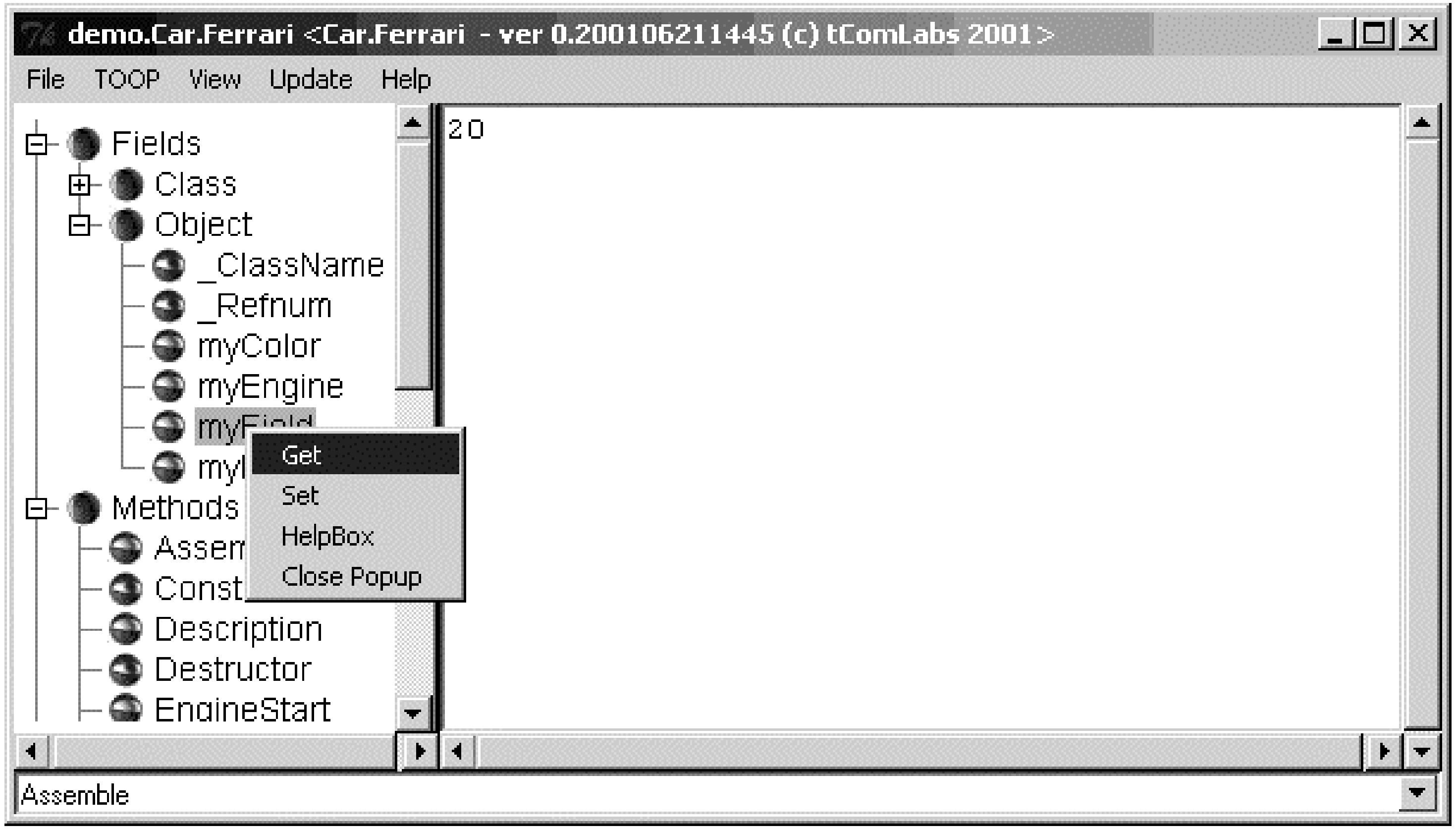
TOOP Introspection 1/2

Since all TOOP classes extend from the base class `TOOP.Object` a uniform approach to all objects is possible using the `Get, Set` and `Info, ...` Methods.

```
%$AnObject Info -methods
{Assemble this args} {Constructor this arg ...

%$AnObject Info -fields
{ClassCount Description ImplementList Pa ...

%$AnObject Info -documentmethod Info
# Reflection/Introspection of every TOOP object...
# <-methods> : returns MethodNameList of <this> ...
# <-fields>  : returns a 2 element list { StaticF ...
```

TOOP Introspection 2/2

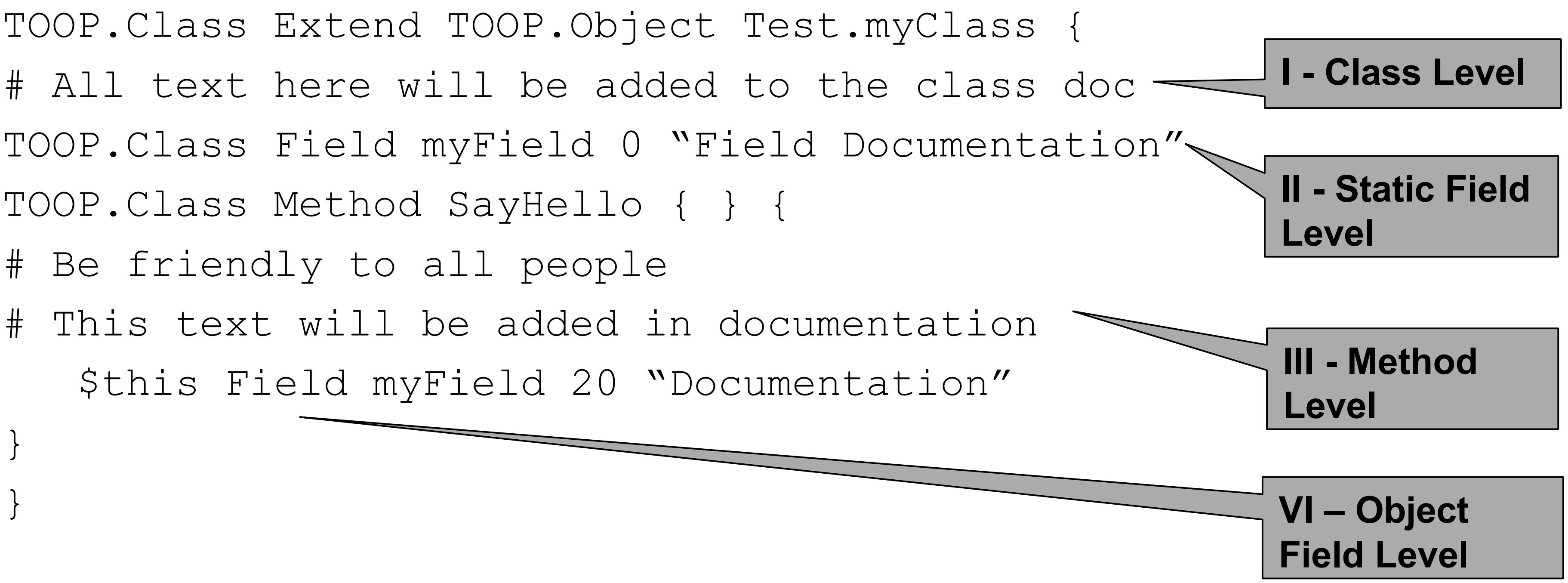
```
%$AnObject Tk
```



demo.Car.Ferrari <Car.Ferrari - ver 0.200106211445 (c) tComLabs 2001>

```
20
```

TOOP Auto-Documentation 1/2



```
TOOP.Class Extend TOOP.Object Test.myClass {
# All text here will be added to the class doc
TOOP.Class Field myField 0 “Field Documentation”
TOOP.Class Method SayHello { } {
# Be friendly to all people
# This text will be added in documentation
   $this Field myField 20 “Documentation”
}
}
```

I - Class Level

II - Static Field Level

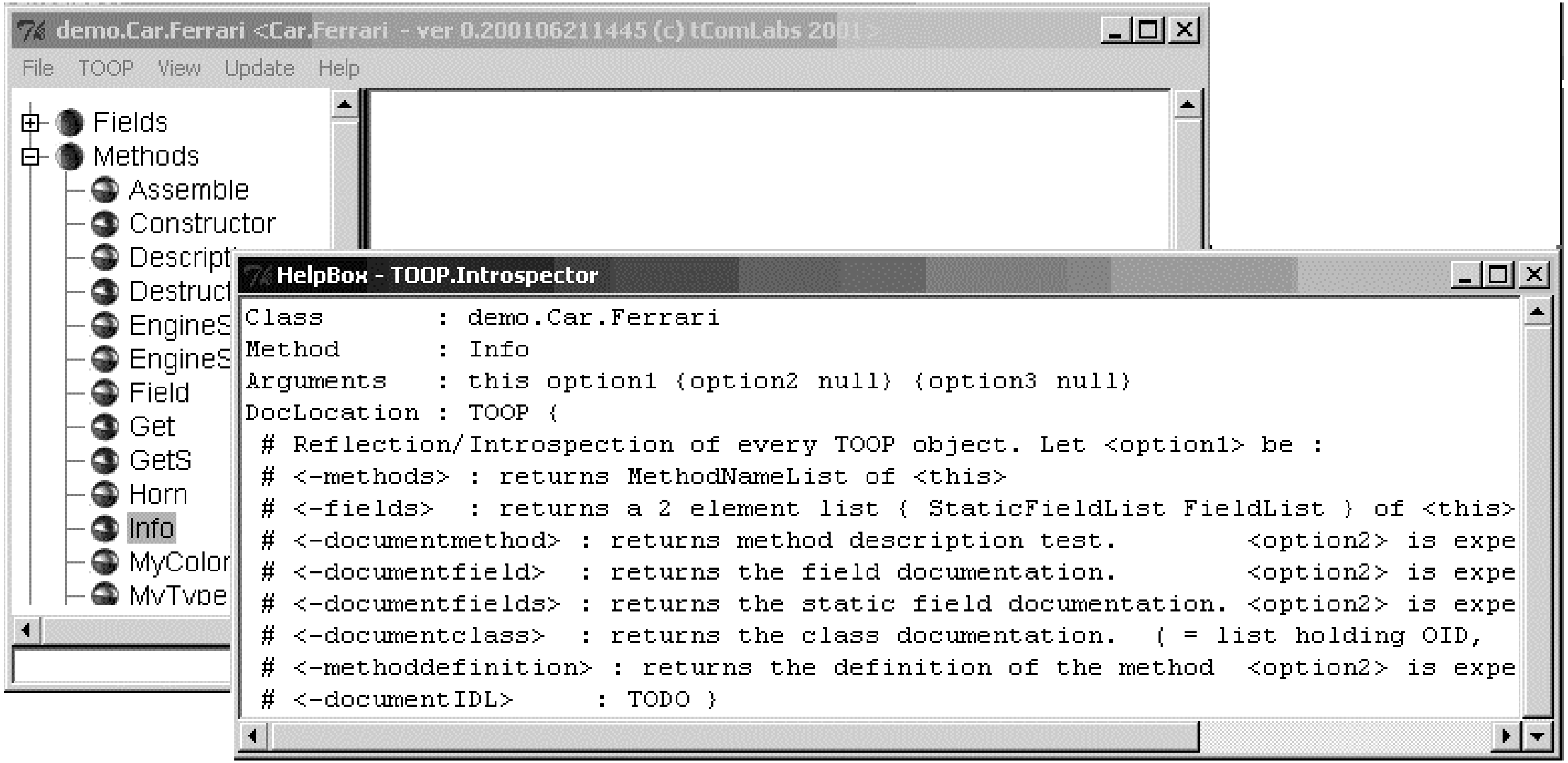
III - Method Level

VI – Object Field Level

Rule N° 1 : Write first the documentation and then the code

TOOP Auto-Documentation 2/2

%$AnObject Tk

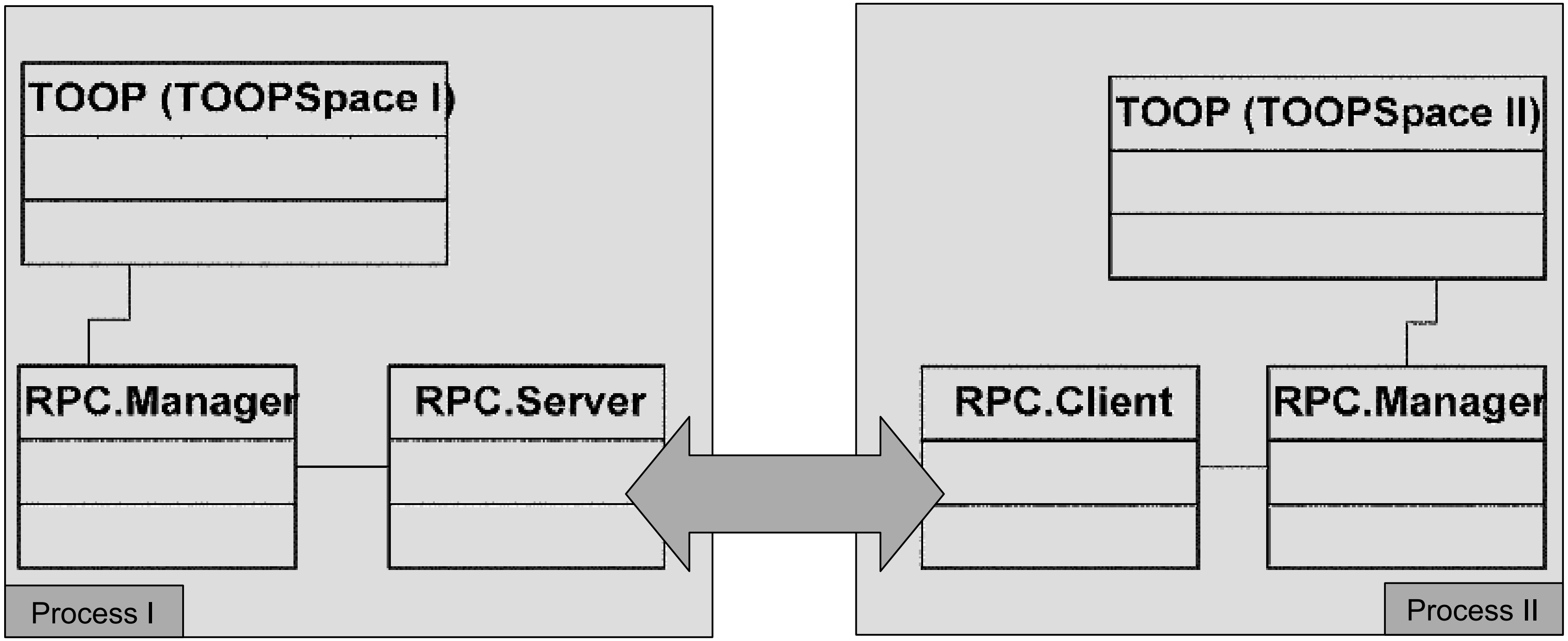


```
Class       : demo.Car.Ferrari
Method      : Info
Arguments   : this option1 {option2 null} {option3 null}
DocLocation : TOOP {
 # Reflection/Introspection of every TOOP object. Let <option1> be :
 # <-methods> : returns MethodNameList of <this>
 # <-fields>  : returns a 2 element list { StaticFieldList FieldList } of <this>
 # <-documentmethod> : returns method description test.        <option2> is expe
 # <-documentfield>  : returns the field documentation.        <option2> is expe
 # <-documentfields> : returns the static field documentation. <option2> is expe
 # <-documentclass>  : returns the class documentation.   ( = list holding OID,
 # <-methoddefinition> : returns the definition of the method  <option2> is expe
 # <-documentIDL>     : TODO }
```

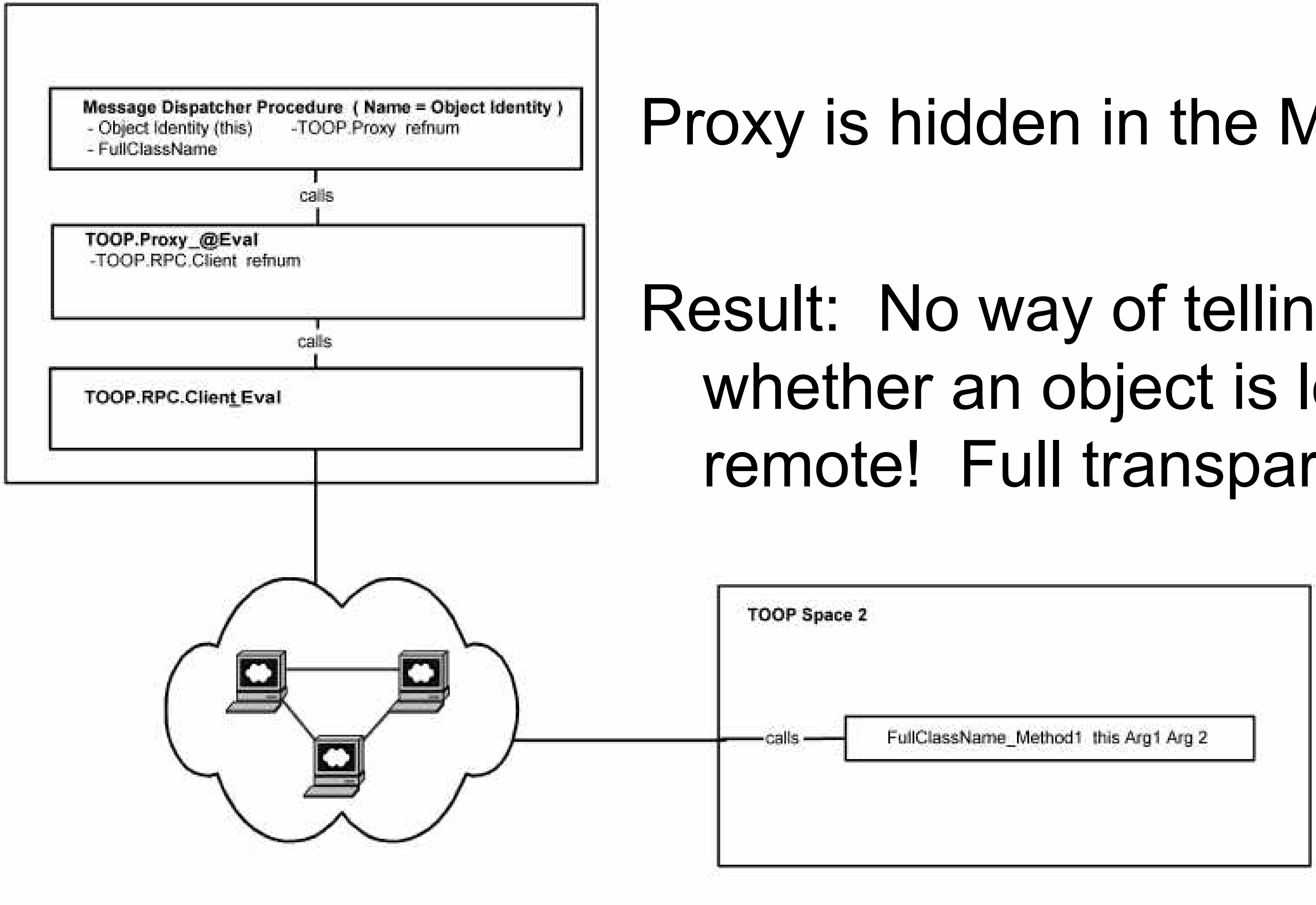
TOOP Distributed Operation 1/2

TOOP implements a transparent system that allows communication with objects in other TOOP Spaces

```
% TOOP New demo.Car –TOOP {-distributed OtherSpace }
```



TOOP Distributed Operation 2/2



Message Dispatcher Procedure (Name = Object Identity)
- Object Identity (this) -TOOP.Proxy refnum
- FullClassName

calls

TOOP.Proxy_@Eval
-TOOP.RPC.Client refnum

calls

TOOP.RPC.Client_Eval

TOOP Space 2

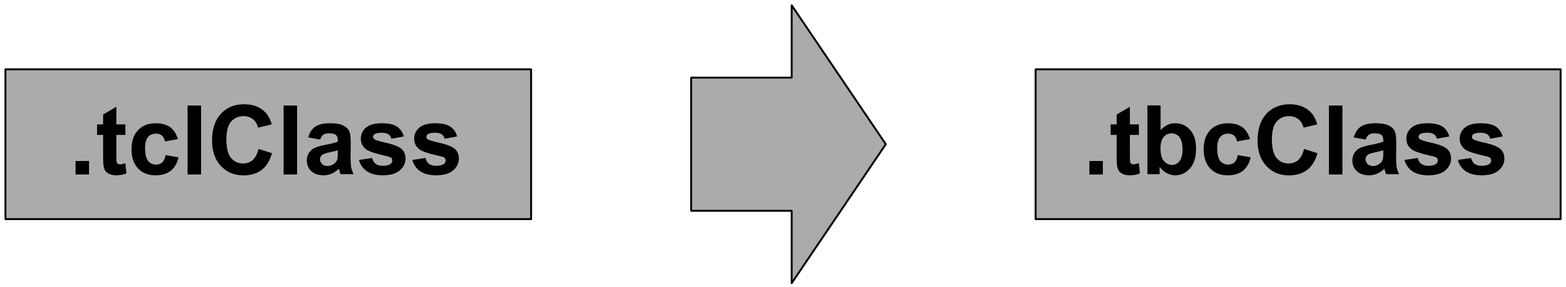
calls

FullClassName_Method1 this Arg1 Arg 2

Proxy is hidden in the MDP

Result: No way of telling whether an object is local or remote! Full transparancy

TOOP Compilation



.tclClass

.tbcClass

A .tbcClass file:

- Contains the compiled (Tcl ByteCode) version of a .tclClass file
- Is interchangable with a .tclClass file

TOOP - Object Orientation

- Abstraction
- Objects
- Object Identity
- Object State
- Object Behavior
- Messages
- Object Classification
- Object Copy
- Sharing (Inheritance)
- Polymorphism
- Encapsulation
- Object Lifetime
- Distributed Objects
- Class Library
- Introspection
- Object Serialization
- ...

TOOP Con’s

- Requires basic OO knowledge (What is an object)
- Speed reduction by 5 times over plain Tcl (However can be overcome to 0 times!)
- All objects are stored in one array OO
- No notions of public/private

TOOP Pro’s

- Is being used in mission critical software by tComLabs during the Euro-DOCSIS certification test process
- Introduces all advantages of OO design (UML, design patterns, reusability,...)
- Tcl only
- Compilable (Tcl Byte Code)

TOOP Conclusion

TOOP:

- Another OO extension for Tcl
- Standard GUI on all objects
- Framework for distributed applications
- Self descriptive