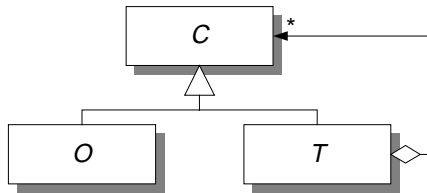


***The COM Pilot Application***  
*COT/3-15-V1.2*



Centre for Object Technology

Revision history:    10-08-98    V1.0    First version as report.  
                          10-08-98    V1.1    Pictures included.  
                          08-09-98    V1.2    First public version.

Author(s):            Aarhus University, Danish Technological Institute

Status:                Final

Publication:         Public

Summary:

<p>This report presents an application to be used for evaluating the COM support in four different development environments.</p>
--

© Copyright 1998 Aarhus University, Danish Technological Institute

# Table of Contents

<b>1. INTRODUCTION.....</b>	<b>4</b>
<b>2. DEMANDS FOR THE APPLICATION.....</b>	<b>4</b>
2.1 GENERAL DEMANDS .....	4
2.2 COM SPECIFIC DEMAND.....	4
<b>3. PILOT APPLICATION PROCESS SPECIFICATION.....</b>	<b>5</b>
3.1 PHASE 1 .....	5
3.1.1 Phase 1.1 - Create a component with an arithmetic interface .....	5
3.1.2 Phase 1.2 - Change the existing members of an interface.....	6
3.1.3 Phase 1.3 - Add a member to an existing interface .....	6
3.2 PHASE 2 .....	7
3.2.1 Phase 2.1 - Add an interface to Deep_Thought's Calc component .....	7
3.2.2 Phase 2.2 - Remove an interface from Deep_Thought's Calc component.....	7
3.3 PHASE 3 .....	7
3.3.1 Phase 3.1 - Add a component to Deep_Thought .....	7
3.3.2 Phase 3.2 - Remove a component from Deep_Thought.....	8
3.4 PHASE 4: CONTAINMEN.....	8
3.4.1 Phase 4.1 - Delegation .....	9
3.4.2 Phase 4.2 – Aggregation .....	9
3.5 PHASE 5 - CREATE THE DEEP_THOUGHT SERVER .....	9
3.6 PHASE 6 - CREATE A CLIENT WITH A GRAPHICAL INTERFACE.....	10
3.7 PHASE 7 - CHANGE DEEP_THOUGHT TO IMPLEMENT THE IDISPATCH INTERFACE .....	10
3.8 PHASE 8 - CREATE INSTALLATION DISKS .....	11
3.9 PHASE 9 - EXCHANGE IMPLEMENTATIONS.....	11

# 1. Introduction

This report presents an application to be used for evaluating the COM support in four different development environments. The summary of the evaluation as well as the list of reports documenting the results of the evaluation can be found in COT/3-4: *Evaluation of COM Support in Development Environments*.

To better understand the idea of the application in section 2 a number of general demands for the application is presented. In Section 3 then the application is presented as a number of steps to be performed by each of the evaluated tools.

## 2. Demands for the Application

### 2.1 General Demands

To model a "real" development history, the task must be divided into several "phases" with changing requirements between at least some of the phases. In real-life, the different phases will not be known until the previous phases are completed (at least in the case of changing of requirements). This cannot be simulated easily here, and the total list of phases are presented at once. One should solve the task of each phase without peeking at the forthcoming phases!

The pilot application should cover a broad spectrum of the COM characteristics already identified in the Taxonomy.

On the other hand, the main purpose of the pilot application is to provide a specific task to accomplish in the different development tools, that we are to investigate and categorise in the Taxonomy.

The pilot application process should be able to complete in approximately 2 man-months (per tool).

### 2.2 COM Specific Demand

Both home-made and standard components should be used.

If possible, the home-made components should be exchanged between the different work groups. The interfaces implemented should be specified completely in the pilot application, for this to be possible. The tool must be able to use an existing GUID, how else would an implementation of existing standard interfaces be possible?

Type libraries and object browsers should be generated/used.

**GUIDs:** It should be checked if predefined GUIDs can be used for implementing an interface (do the tool insist on generating one?). Preferably we should also test the tool's ability to do without a GUID (i.e. identification by name).

**Interface parameters and types:** This is an important issue, and we should both test conversion between simple and structured types (does the tool do it, or do we have to do it ourselves), conversion between languagespecific types and COM types.

**Interface parameters and memory handling:** Also an important issue, and we should both test reference-counting and allocation/deallocation in the context of in-, inout-, and out-parameters.

**Threading should be tested:** We should make sure, that certain interface-funktions are called simultaneously.

**Versioning of the same interface should be tested.** This is related to item 2, since registering a number of group's components on one machine will force the client to handle different versions of the same implementation.

**In-process and out-of-process servers should be tested.** If possible we should change strategy between two phases, to see how the tools handle the change from, e.g., out-of-process to in-process (the in-process (dll) requires certain extra entypoint to be generated: RegisterServer etc).

**Compile-time vs. early/late binding should be tested.**

## **3. Pilot Application Process Specification**

### **3.1 Phase 1**

#### **3.1.1 Phase 1.1 - Create a component with an arithmetic interface**

**Motivation:**

To examine the tool's ability to build a simple COM server.

**Task:**

Create a DLL-server Deep\_Thought consisting of a component Calc with an interface ICalc as described in DEEP1\_1.IDL:

```
HRESULT _stdcall Dad([in] wchar_t n1, [out, retval] wchar_t *n3)
HRESULT _stdcall Subtract([in] wchar_t n1, [in] wchar_t n2, [out, retval] wchar_t *n3)
HRESULT _stdcall Multiply([in] wchar_t n1, [in] wchar_t n2, [out, retval] wchar_t *n3)
```

```
HRESULT _stdcall Divide([in] wchar_t n1, [in] wchar_t n2, [out, retval] wchar_t *n3)
```

Create a type library for the server as the one defined in DEEP1\_1.IDL.

Build a simple client to check the ICalc interface.

### 3.1.2 Phase 1.2 - Change the existing members of an interface

#### **Motivation:**

To examine how the tool handles changes to an existing interface.

#### **Task:**

We have discovered that we have made errors in the specification of the members of the ICalc interface. So change the existing ICalc interface to the following:

```
HRESULT _stdcall Add([in] long v1, [in] long v2, [out, retval] long *res)
```

```
HRESULT _stdcall Subtract([in] long v1, [in] long v2, [out, retval] long *res)
```

```
HRESULT _stdcall Multiply([in] long v1, [in] long v2, [out, retval] long *res)
```

```
HRESULT _stdcall Divide([in] long v1, [in] long v2, [out, retval] long *res)
```

Create a type library for the server as the one defined in DEEP1\_2.IDL:

Adjust the existing simple client and use it to check the ICalc interface.

### 3.1.3 Phase 1.3 - Add a member to an existing interface

#### **Motivation:**

To examine how the tool handles additions to an existing interface.

#### **Task:**

Add a method Modulus to the ICalc interface:

```
HRESULT _stdcall Modulus([in] long v1, [in] long v2, [out, retval] long *res)
```

Create a type library for the server as the one defined in DEEP1\_3.IDL:

Adjust the existing simple client and use it to check the ICalc interface.

## 3.2 Phase 2

### 3.2.1 Phase 2.1 - Add an interface to Deep\_Thought's Calc component

**Motivation:**

To examine the tool's ability to handle components with more than one interface.

**Task:**

Add an IConv interface to the Calc component as described in DEEP2\_1.IDL:

```
HRESULT stdcall Hex2Dec([in] wchar_t v, [out, retval] long *res)
```

```
HRESULT stdcall Dec2Hex([in] long v, [out, retval] wchar_t *res)
```

Create a type library for the server as the one defined in ?.IDL.

Adjust the existing simple client and use it to check the ICalc as well as the IConv interface.

### 3.2.2 Phase 2.2 - Remove an interface from Deep\_Thought's Calc component

**Motivation:**

To examine the tool's ability to handle the removal of an interface.

**Task:**

Remove the IConv interface from the Calc component.

Create a type library for the server as the one defined in DEEP2\_2.IDL.

Adjust the existing simple client and use it to check the ICalc interface.

## 3.3 Phase 3

### 3.3.1 Phase 3.1 - Add a component to Deep\_Thought

**Motivation:**

To examine the tool's ability to create servers with more than one component.

**Task:**

Add a Conv component with the interface described in DEEP3\_1.IDL to Deep\_Thought.

Create a type library for the server as the one defined in DEEP3\_1.IDL.

Adjust the existing simple client and use it to check the ICalc interface on the Calc component as well as the IConv interface on the Conv component.

### 3.3.2 Phase 3.2 - Remove a component from Deep\_Thought

#### **Motivation:**

To examine the tool's ability to handle the removal of a component from a server.

#### **Task:**

Remove the Conv component from Deep\_Thought.

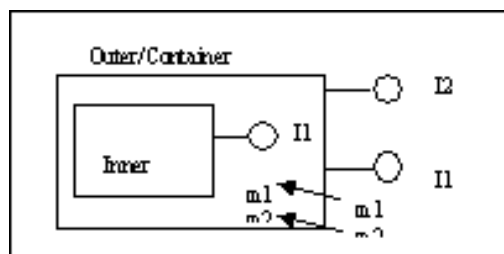
Create a type library for the server as the one defined in DEEP3\_2.IDL.

Adjust the existing simple client and use it to check the ICalc interface of the Calc component.

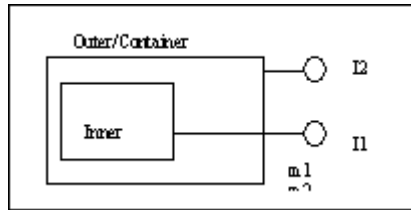
## 3.4 Phase 4: Containment

#### **Motivation:**

To examine how components can be contained and used when they reside inside other components. Containment and exposing can be done in two way: Either by delegation or by aggregation. When using delegation the contained component is exposed by adding a new interface to the container and then calling the inner interfaces methods, like:



When using aggregation the inner interface is exposed directly to the user, like:



Because the nature of the two implementation techniques are quite different both have to be investigated:

### 3.4.1 Phase 4.1 - Delegation

#### Task:

Create a new component called CalcDel. This component should implement the interface ICalc2 and aggregate the component Calc. The Interface for ICalc should be exposed through delegation (first figure).

Create a type library for the server as the one defined in DEEP4\_1.IDL.

Adjust the existing simple client and use it to check both the ICalc and the ICalc2 interface of the Calc component.

### 3.4.2 Phase 4.2 – Aggregation

#### Task:

Create a new component called CalcAggr. This component should implement the interface ICalc2 and aggregate the component Calc. The Interface for ICalc should be exposed through aggregation (second figure) as described in DEEP4\_2.IDL.

Use the existing simple client to check both the ICalc and the ICalc2 interface of the Calc component.

## 3.5 Phase 5 - Create the Deep\_Thought server

#### Motivation:

To construct a server with all the existing facilities for use in the following phases. And to examine how the tool handles the transition of existing servers from DLL to EXE servers and vice versa.

To enable the exchange of servers between different development tools (phase 9) all developers must use the same ProgID's.

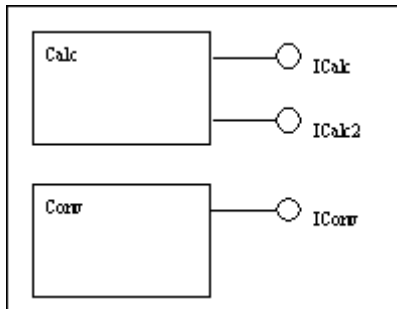
Use Deep\_Thought as servername for the DLL server.

Use Deep\_Thought\_EXE as servername for the EXE server.

**Task:**

Create a DLL-server Deep\_Thought with a matching type-library as the one described in DEEP5.IDL:

Deep\_Thought:



Create a client as described in phase 6 and use it to check all the interfaces of the DLL server.

Change the existing DLL server to an EXE server.

Use the existing client to check all the interfaces of the EXE server.

Change the existing EXE server to an DLL server.

Use the simple client to check all the interfaces of the DLL server.

## 3.6 Phase 6 - Create a client with a graphical interface

**Motivation:**

To construct a client that uses all the existing facilities in Deep\_Thought for use in the following phases.

**Task:**

Using standard controls for buttons and textdisplay, create a calculator client with a graphical appearance, using the interfaces provided by Deep\_Thought.

## 3.7 Phase 7 - Change Deep\_Thought to implement the IDispatch interface

**Motivation:**

To examine how dispatchable interfaces are handled by the tool.

**Task:**

Change both the EXE and the DLL version Deep\_Thought to implement the IDispatch interface instead of / as well as the custom interface specified above.

To enable the exchange of servers between different development tools (phase 9) all developers must use the same ProgID's.

Use Deep\_Thought\_Disp as servername for the DLL server.  
Use Deep\_Thought\_Disp\_EXE as servername for the EXE server.

A type library for the IDispatch based Deep\_Thought is described in ?.IDC.

Adjust the client appropriately and use it to check the Deep\_Thought server.

Test both IDispatched based versions of Deep\_Thought by specifying both UUIDs as well as ProgIDs.

## 3.8 Phase 8 - Create installation disks

**Motivation:**

To examine how the tool aides in the creation of installtion disks/programs.

**Task:**

Create installation disks for the Deep\_Thought servers (both the DLL and the EXE versions) from both phase 5 and phase 7 (a total of four versions) and distribute these to the other groups.

## 3.9 Phase 9 - Exchange implementations

**Motivation:**

To examine if there are any problems in using clients and servers implemented in different tools.

**Task:**

Use the tools objectbrowser (if any is available) to browse the type-libraries created by the other groups, and check whether there are any differencies from the component developed in the tool itself.

*Centre for  
Object Technology*

Test your graphical client against the other groups servers.