CAPE-OPEN

Delivering the power of component software
and open standard interfaces
in Computer-Aided Process Engineering

Open Interface Specification:

Collection Common Interface

www.colan.org

ARCHIVAL INFORMATION

Filename	Collection Common Interface.doc
Authors	CO-LaN consortium
Status	Public
Date	August 2003
Version	version 2
Number of pages	21
Versioning	version 2, reviewed by Jean-Pierre Belaud, August 2003
	version 1, Methods & Tools group, July 2001
Additional material	
Web location	www.colan.org
Implementation specifications version	CAPE-OPENv1-0-0.idl (CORBA) CAPE-OPENv1-0-0.zip and CAPE-OPENv1-0-0.tlb (COM)
Comments	

IMPORTANT NOTICES

Disclaimer of Warranty

CO-LaN documents and publications include software in the form of *sample code.* Any such software described or provided by CO-LaN --- in whatever form --- is provided "as-is" without warranty of any kind. CO-LaN and its partners and suppliers disclaim any warranties including without limitation an implied warrant or fitness for a particular purpose. The entire risk arising out of the use or performance of any sample code --- or any other software described by the CAPE-OPEN Laboratories Network --- remains with you.

CO-LaN is a non for profit organization established under French law of 1901.

Trademark Usage

Many of the designations used by manufacturers and seller to distinguish their products are claimed as trademarks. Where those designations appear in CO-LaN publications, and the authors are aware of a trademark claim, the designations have been printed in caps or initial caps.

Microsoft, Microsoft Word, Visual Basic, Visual Basic for Applications, Internet Explorer, Windows and Windows NT are registered trademarks and ActiveX is a trademark of Microsoft Corporation.

Netscape Navigator is a registered trademark of Netscape Corporation.

Adobe Acrobat is a registered trademark of Adobe Corporation.

SUMMARY

This document describes a *Common Interface* proposed by the Methods & Tools group: the Collection Common Interface. The Common Interfaces are interfaces and implementation models for handling concepts that may be required by any *Business Interfaces* specification.

The interface representing collections was already used by the UNIT specification (CO-CUNIT-1 Version 2.0), where this was included as an internal interface. The fact that the interface was used to hold two different types of entities (parameters and ports) and the generality of the collections concept shows that it can be re-used by any other CO packages requiring similar services.

ACKNOWLEDGEMENTS

CONTENTS

LIST OF FIGURES

1. Introduction

The aim of the Collection interface is to give a CAPE-OPEN component the possibility to expose a list of objects to any client of the component. The client will not be able to modify the collection, i.e. removing, replacing or adding elements. However, since the client will have access to any CAPE-OPEN interface exposed by the items of the collection, it will be able to modify the state of any element.

CAPE-OPEN Collections don’t allow exposing basic types such as numerical values or strings. Indeed, using CapeArrays is more convenient here.

Not all the items of a collection must belong to the same **class**. It is enough if they implement the same interface or set of interfaces. A CAPE-OPEN specification a component that exposes a collection interface must state clearly which interfaces must be implemented by all the items of the collection.

2. Requirements

2.1 Textual requirements

The requirement is basic in the case of this common interface: any client who has reached a CO collection wants

- to know the number of items the collection has,
- and to get a specific item using the name or the index of this item.

The collection (number of items, state of each item) can be changed at any time by any operation of any object within the PMC, such as by connecting a port etc. The only operations that are guaranteed not to modify the contents of a collection are ICapeCollection's operations.

This issue may be addressed later through the CAPE-OPEN event handling mechanism; for the moment the calling client is supposed to refresh its collection every time it wants to use it.

2.2 Use cases

Resulting from the previous requirement the straightforward UML Use-Cases are showed.

2.2.1 Actors

- **Client.** Any person or software component that decides to manage a collection.

2.2.2 List of Use Cases

- **UC-001**: Get the number of items
- **UC-002**: Get an item

2.2.3 Use Cases Maps

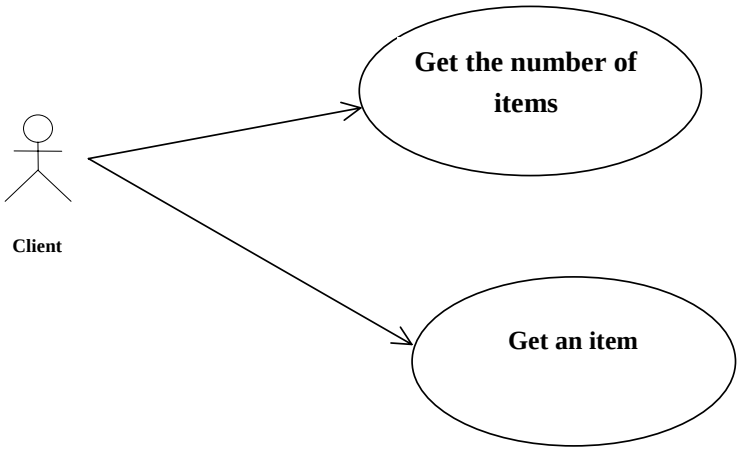

Figure 1 Use-Case map

2.2.4 Use Cases

This subsection lists the two Use Cases.

UC-001: GET THE NUMBER OF ITEMS

Actors: client Priority: high Classification: Context: the client deals with any objects which expose a collection. These objects identify clearly the nature of items the collection provides. Pre-conditions: the client has got the collection successfully from an object. Flow of events: the collection returns the number of items. If there is no item, the zero value is returned. Post-conditions: Errors: Uses: Extends:

UC-002: GET AN ITEM

Actors: Client Priority: High Classification:

Context: The client deals with any objects which expose a collection. These objects identify clearly the nature of items the collection provides.

Pre-conditions: The client has got the collection successfully from an object.

Flow of events: An item is returned according to a criterion. The client gives a criterion which characterises the item. The collection that uses its own searching procedure returns the resulting item.

Post-conditions:

Errors: Any item is found.

Uses:

Extends:

2.3 Sequence diagrams

None.

3. Analysis and Design

This chapter introduces the analysis models. That is independent from the distributed platform.

3.1 Overview

3.2 Sequence diagrams

None.

3.3 Interface diagrams

Basically the interface diagram exposes one interface called ICapeCollection.

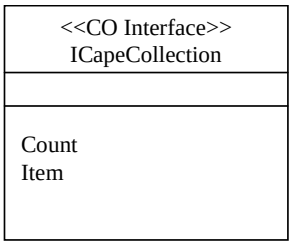

Figure 2 ICapeCollection interface

Within the following picture we can visualise the relations between implementation objects and CO interfaces.

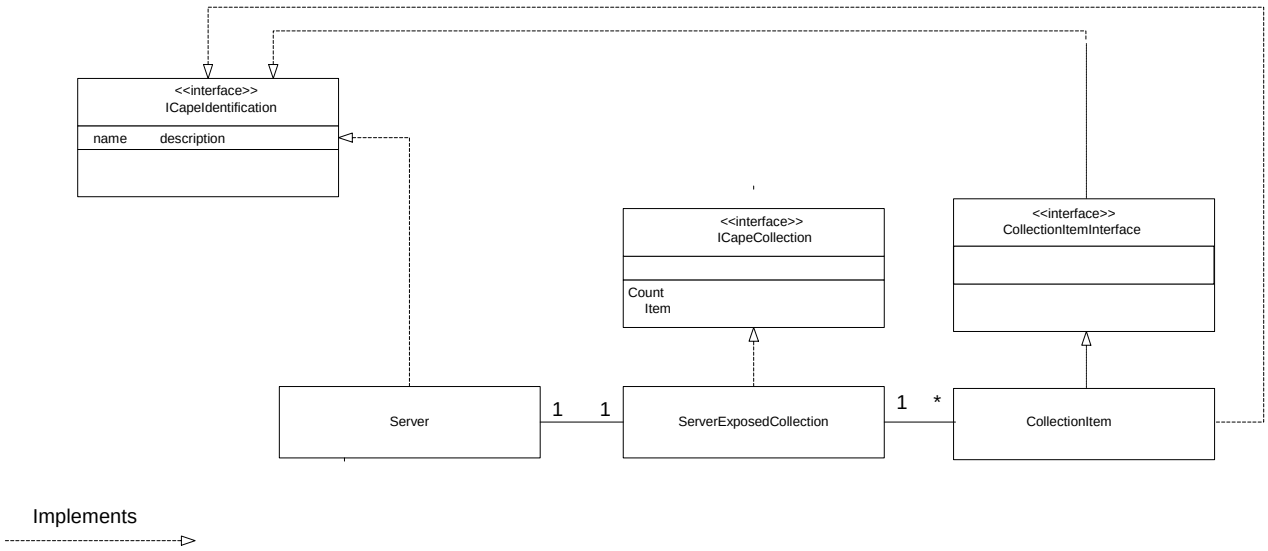

Figure 3 Interface diagram

When a server wants to expose a collection of items, it must have a property (let's call it in this document getCollection) that returns an object that implements the ICapeCollection interface

Similarly, the Item method in ICapeCollection can be used to obtain a pointer to each of the items of the collection. Since the client will know the meaning and type of the items of the exposed collection, it will convert this object to the desired interface. Normally, the client will also want to query to ICapeIdentification, in order to obtain information about the identity of a specific item.

3.4 State diagrams

This section presents one State Diagram.

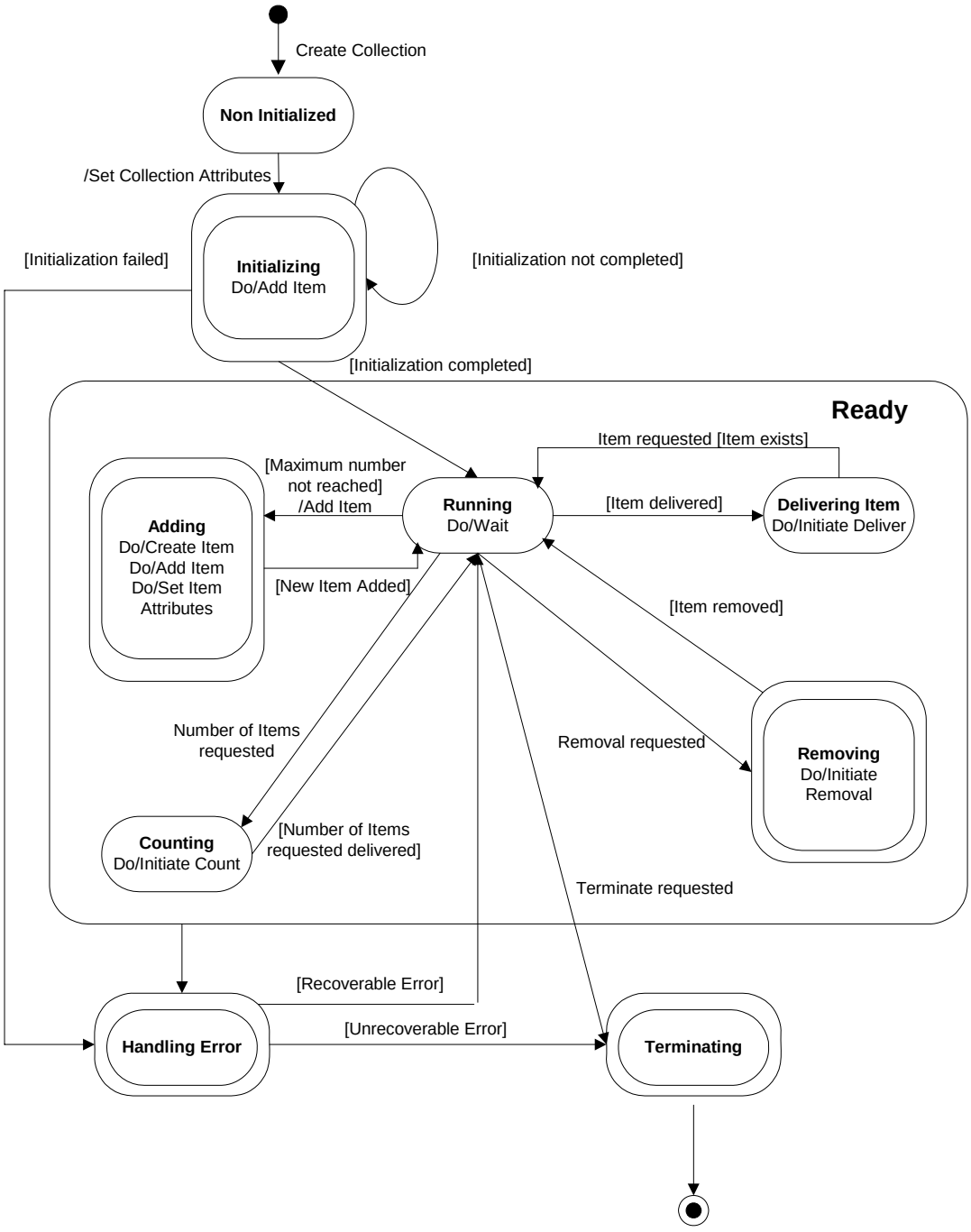

Figure 4 State diagram

3.5 Other diagrams

3.6 Interfaces descriptions

The ICapeCollection interface provides a means of collecting together lists of CAPE-OPEN items/entities (eg. parameters, ports, …).

3.6.1 ICapeCollection

Interface Name	ICapeCollection
Method Name	Item
Returns	CapeInterface

Description

Return an element from the collection. The requested element can be identified by its actual name (e.g. type CapeString) or by its position in the collection (e.g. type CapeLong). The name of an element is the value returned by the ComponentName() method of its ICapeIdentification interface. The advantage of retrieving an item by name rather than by position is that it is much more efficient. This is because it is faster to check all names from the server part than checking then from the client, where a lot of COM/CORBA calls would be required.

Arguments

Name	Type	Description
[in] id	CapeVariant	Identifier for the requested item: name of item (the variant contains a string) position in collection (it contains a long)

Errors

ECapeUnknown, ECapeFailedInitialisation, ECapeOutOfBounds, ECapeInvalidArgument

Interface Name	ICapeCollection
Method Name	Count
Returns	CapeLong

Description

Return the number of items in the collection.

Arguments

No arguments required

Errors

ECapeUnknown, ECapeFailedInitialisation

3.7 Scenarios

None.

4. Interface Specifications

4.1 COM IDL

```
// You can get these intructions in Common.idl file from CAPE-OPENv1-0-0.zip
```

4.2 CORBA IDL

```
// You can get these intructions in CAPE-OPENv1-0-0.idl within the
CAPEOPEN100::Common::Collection module
```

5. Notes on the interface specifications

On the CORBA side, in order to avoid working with CapeVariant (type any), two operations corresponding to Item() are specified: ItemByIndex(CapeLong) and ItemByName(CapeString).

So we get the following codes:

```
interface ICapeCollection : Identification::ICapeIdentification{
	Types::CapeLong Count() raises (Error::ECapeUnknown,
Error::ECapeFailedInitialisation);
	Types::CapeInterface ItemByIndex(in Types::CapeLong index) raises
(Error::ECapeUnknown, Error::ECapeInvalidArgument, Error::ECapeFailedInitialisation,
Error::ECapeOutOfBounds);
	Types::CapeInterface ItemByName(in Types::CapeString name) raises
(Error::ECapeUnknown, Error::ECapeInvalidArgument, Error::ECapeFailedInitialisation,
Error::ECapeOutOfBounds);
};
```

6. Prototypes implementation

7. Specific Glossary Terms

8. Bibliography

8.1 Process simulation references

8.2 Computing references

8.3 General references

9. Appendices