Date: December 2003

UML Profile for CCM

Draft Adopted Specification

ptc/03-12-01

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR

WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE.
IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 250 First Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

The OMG Object Management Group Logo®, CORBA®, CORBA Academy®, The Information Brokerage®, XMI® and IIOP® are registered trademarks of the Object Management Group. OMG™, Object Management Group™, CORBA logos™, OMG Interface Definition Language (IDL)™, The Architecture of Choice for a Changing World™, CORBAservices™, CORBAfacilities™, CORBAmed™, CORBAnet™, Integrate 2002™, Middleware That's Everywhere™, UML™, Unified Modeling Language™, The UML Cube logo™, MOF™, CWM™, The CWM Logo™, Model Driven Architecture™, Model Driven Architecture Logos™, MDA™, OMG Model Driven Architecture™, OMG MDA™ and the XMI Logo™ are trademarks of the Object Management Group. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

ISSUE REPORTING

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents & Specifications, Report a Bug/Issue.

Table of Contents

1 Scope

Editorial Comment: Needs to be completed.

2 Conformance

(was section 2.4 - Relationship to the MDA)

This specification is compliant with the Model Driven Architecture (MDA) defined by the OMG. The UML Profile for CORBA Components specification was designed to provide a standard means for expressing CCM-based applications (PSMs) using UML notation and thus to support all kind of MDA model transformations such PIM→PSM or PSM→PSM and also work with MOF repositories.

Editorial Comment: Needs to be completed.

3 Normative References

The following normative documents contain provisions which, through reference in this text, constitute provisions of this specification.

Editorial Comment: Needs to be completed.

4 Terms and Definitions

For the purposes of this specification, the terms and definitions given in the normative references and the following apply.

Editorial Comment: Needs to be completed (or possibly eliminated).

5 Symbols

List of symbols/abbreviations.

Editorial Comment: Needs to be completed (or possibly eliminated).

6 Additional Information

6.1 Changes to Adopted OMG Specifications

There are no changes to existing OMG Specifications.

6.2 The Relationship to the UML 2.0

The CCM profile is defined as a UML 1.5 profile. In September, 2000, OMG started to work on the Release 2.0 major revision of the UML specification. UML 2.0 is tailored to MDA requirements, and is being proposed in four separate RFPs: UML Infrastructure, UML Superstructure, Object Constraint Language, and UML Diagram Interchange. It s excepted, that new UML 2.0 concepts (e.g. port, component, etc.) will simplify the modeling of component-based infrastructures like CCM or EJB. However, the necessity of the UML profile for CCM will not disappear and it will be a possible subject for a separated RFP in the future.

6.3 Acknowledgements

The following companies submitted and/or supported parts of this specification:

- Alcatel
- Fraunhofer Institute FOKUS
- IKV++ Technologies AG
- Laboratoire d'Informatique Fondamentale de Lille
- Technical University Berlin
- Thales

NOTE: The technology proposed by this specification is based on the work of the MASTER project (http://www.esi.es/Master) and the COACH project (http://www.ist-coach.org/) of the IST Program of the European Commission. The submitters would like to thank the participants of these projects for their contributions and review activities.

7 Overview

- Explain packages of the CCM MOF metamodel (ComponentIDL, CIF)
- Explain relations to BaseIDL package
- Explain relations to UML Profile for CORBA, which already covers BaseIDL

7.1 UML Subset Definition

The UML Profile for CCM specializes the UML Core package (formal/03-03-01) and the UML Profile for CORBA (ptc/00-10-01).

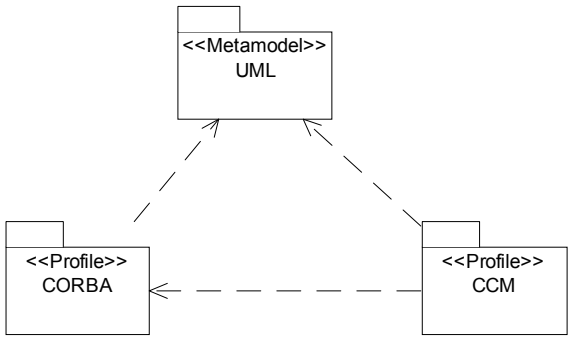

Figure 1 - Import dependencies between UML, CORBA and CCM packages

The following concrete metaclasses, and implicitly all super-metaclasses of these metaclasses, are used:

- Generalization.
- Association.
- Association Class.
- Class.
- Operation.
- Constraint.
- Package

The CCM metamodel is defined on top of the CORBA 2 metamodel. We have the same relationship at the profile level. The CCM profile uses the CORBA2 profile. It directly specializes the following stereotypes:

- «CORBAInterface».
- «CORBAValue».

7.2 CCM package

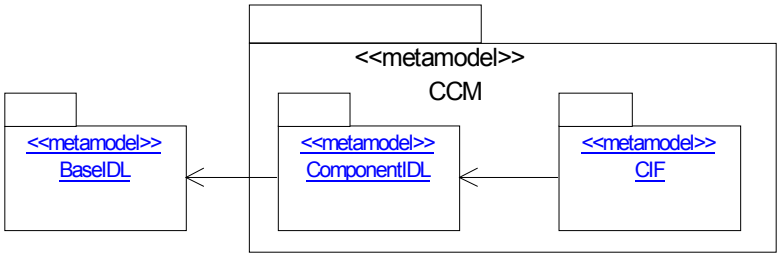

Figure 2 - CCM MOF metamodel

As shown in Figure 2 the whole CCM concept space represented by the package CCM with the stereotype <<metamodel>> consists of further packages ComponentIDL and CIF (Component Implementation Framework). The first Package ComponentIDL expresses the Component Model extensions. This package is related to the reference metamodel BaseIDL that is a MOF-compliant description of the pre-existing CORBA Interface Repository that is already specified in the UML Profile for CORBA (see Import dependencies between UML, CORBA and CCM packages). The CIF Package contains metaclasses and associations for definition the programming model for constructing component implementations, and is based on the reference ComponentIDL metamodel.

8 CCM Profile Definition

8.1 Component IDL Profile

8.1.1 IDL metamodel

Figure 3 gives the structure of the CCM external concepts. A more detail description of these concepts can be found in the CCM OMG standard definition.

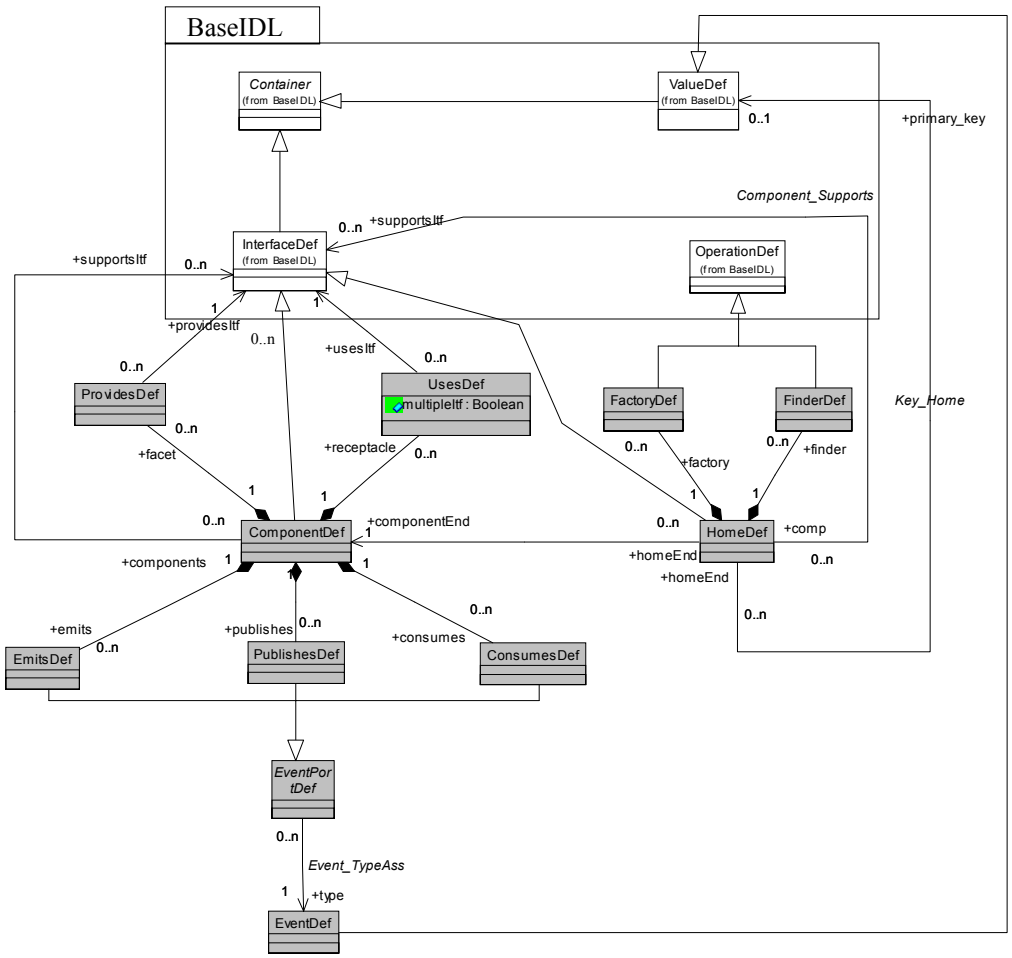

Figure 3 - IDL3 Metamodel

This abstract model is used to characterize CORBA Component interfaces. The IDL3 language has been defined to describe instances of this Meta-Model. It is a generalization of the OMG IDL language.

A component type defines attributes and ports. The attributes are used to configure the component. By using ports, components can use or provide a set of services (typed with a CORBA interface). There are four kinds of ports:

1. A facet is a component provided interface. It is a synchronous communication mechanism.
2. A receptacle is a component required interface. It is a synchronous communication mechanism.
3. An event sink is a component provided interface. It is an asynchronous communication mechanism.
4. An event source is a component required interface. It is an asynchronous communication mechanism.

A component is a kind of interface. A component can inherit from another one and support one or more interfaces. A component cannot inherit from several components at the same time. Multiple inheritance is only possible for interfaces.

A facet represents a component's role. It is described using an interface. A facet is the only visible part for clients. It is only a declarative part. Clients have no access to the implementation part. Facet implementations are hidden inside the component. Facets and components have the same lifecycle. Each facet has its own reference.

With a receptacle a component can use a reference. This relationship is called a connection. Connections are used for component assembly. There are two receptacle kinds. Simple receptacle can only use a single reference. "Multiple" receptacle can use several references.

The CCM also provides a provider/consumer event model. They are two kinds of event ports: event source and event sink. An event source can be either an emitter (only one consumer) or a publisher (several consumers). Event sources are used to send events; event sinks are used to receive events.

CORBA Components are managed by homes. A component home provides component factory operations. It can also provide component finder operations. "Home" supports single inheritance.

8.1.2 Profile definition

Component

A CORBA Component is defined using a UML "CORBAComponent" stereotyped class. A "CORBAComponent" can inherit from another one (single inheritance) using the UML generalization. It can also inherit from a set of CORBA interfaces. These relationships are materialized with "CORBASupports" stereotyped generalizations.

Table 1 describes CORBA Component stereotypes. Table 2 defines the associated tagged values.

Table 1 - UML representation of a CORBA Component

Metamodel element	Stereotype name	UMLBase Class	Parent (from Metmodel)
ComponentDef	CORBAComponent	Class	InterfaceDef
Generalization between InterfaceDef and ComponentDef	CORBASupports	Generalization	N/A

Table 2 - Tagged value definition for a CORBA Component

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAComponent	None	N/A	N/A	N/A
CORBASupports	None	N/A	N/A	N/A

<<CORBAComponent>> Constraint

A « CORBAComponent » is a kind of « CORBAInterface ». Each “CORBAComponent” must respect the “CORBAInterface” constraints. It must also respect the following additional constraints:

- A «CORBAComponent» cannot own operations:

  self.feature→forAll(not oclIsKindOf (behavioralFeature))

- A «CORBAComponent» can only inherit from a «CORBAComponent» or a «CORBAInterface»:

  self.generalization→forAll (g : Generalization | g.parent.isStereotyped ("CORBAComponent") or g.parent.isStereotyped("CORBAInterface"))

- Only single inheritance is possible between «CORBAComponent»:

  self.generalization→
  select(parent.isStereotyped("CORBAComponent"))→size <= 1

- Each «CORBAComponent» inheritance from a «CORBAInterface» must be stereotyped «CORBASupports»:

  self.generalization→forAll (g : Generalization | g.parent.isStereotyped("CORBAInterface") implies g.isStereotyped("CORBASupports"))

Example

The CCM Component inheritance model is the UML counterpart of the following IDL3 declaration:

```
interface I1 { } ;
interface I2 { } ;
component C1 supports I1, I2 { } ;
component C2 : C1 { } ;
```

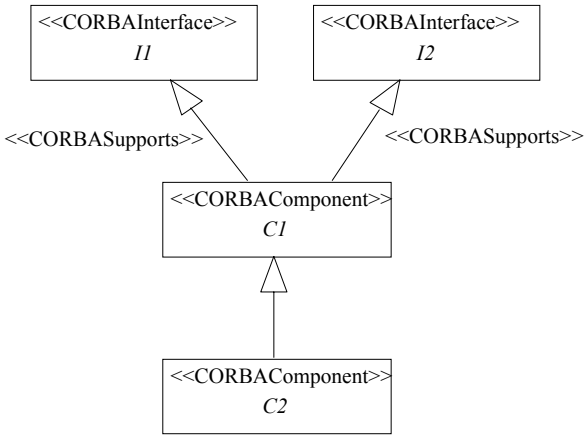

Figure 4 - CCM Component inheritance

Facet and Receptacle

The facets are described using a composition association between a CORBAComponent and a CORBA interface. This association must be stereotyped “CORBAProvides”. The role name of the interface AssociationEnd gives the facet name. The AssociationEnd cardinalities are [1..1] for both association sides.

The receptacles are described using a composition association between a CORBAComponent and a CORBA interface. This association must be stereotyped “CORBAUses”. The role name of the interface AssociationEnd gives the receptacle name. The component side AssociationEnd cardinality must be [1..1]. The receptacle side AssociationEnd cardinality is [1..n] where n is the receptacle cardinality. Table 3 describes facet and receptacle stereotypes. Table 4 defines the associated tagged values.

Table 3 - Facet and receptable UML representation

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
ProvidesDef	CORBAProvides	Association	N/A
UsesDef	CORBAUses	Association	N/A

Table 4 - Facet and receptacle tagged value definition

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAProvides	None	N/A	N/A	N/A
CORBAUses	multiple	Boolean	“no”	“yes”, “no”

Constraints

- It’s an association between a «CORBAComponent» and a «CORBAInterface»:

self.connection→exists(participant.isStereotyped("CORBAComponent")) and self.connection→exists (participant.isStereo-

typed("CORBAInterface"))

- The «CORBAComponent» side is a composition:

  self.connection→exists(participant.isStereotyped("CORBAComponent") and aggregation = #composite)

- The «CORBAInterface» role name must be defined.

  self.connection->exists (participant.isStereotyped("CORBAInterface") and name <> "")

CCMFacet additional constraints

- It's an association stereotyped «CORBAProvides»:

  self.isStereotyped("CORBAProvides")

- The "CORBAInterface" side cardianality must be 1:

  self.connection→exists(participant.isStereotyped("CORBAInterface") and multiplicity.min=1 and multiplicity.max=1)

CCMReceptacle additional constraints

- It's an association stereotyped «CORBAUses»:

  self.isStereotyped("CORBAUses")

- The «CORBAInterface» side cardinality must be 1 for simple receptacle.

  self.connection→exists(participant.isStereotyped("CORBAInterface") and multiplicity.min=1 and multiplicity.max=1)

- The «CCMReceptacle» side cardinality must greater than one for multiple receptacles.

  self.connection→exists(participant.isStereotyped("CORBAInterface") and multiplicity.min=1 and multiplicity.max>1)

Example

The Facets and Receptacles model is the UML counterpart of the following IDL3 declaration:

```
interface I1 { };
interface I2 { };
interface I3 { };
component C1 {
    provides I1 facet1;
uses    I2 receptacle1;
    uses multiple I3 receptacle2;
};
```

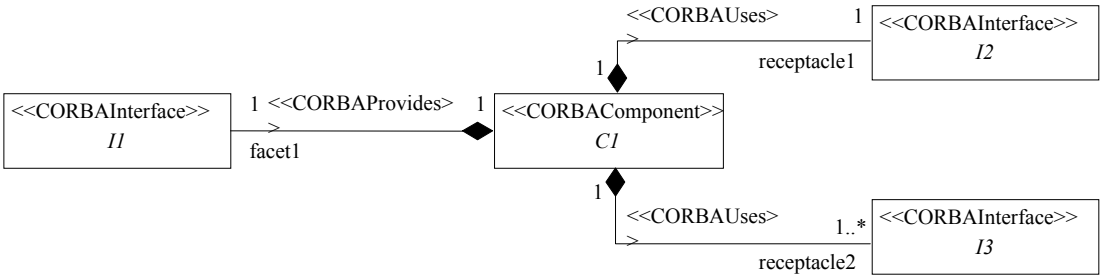

Figure 5 - Facets and Receptacles

Events

Event types are defined using a « CORBAEvent » stereotyped class. The “CORBAEvent” stereotype is a specialization of the « CORBAValue » stereotype. It inherits from all “CORBAValue” constraints. Event source is defined either by a “CORBAEmits” stereotyped composition association or by a “CORBAPublishes” association between a “CORBAComponent” and a “CORBAEvent”. The event source name is defined using the ”CORBAEvent” side association role name.

Event sinks are defined the same way using a “CORBAConsumes” stereotyped composition association.

Table 5 describes event ports stereotyped. Table 6 defines the associated tagged values.

Table 5 - UML representation of CCM event ports

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
EventDef	CORBAEvent	Class	ValueDef
EventPortDef	CORBAEventPort	Association	N/A
EmitsDef	CORBAEmits	Association	EventPortDef
PublishesDef	CORBAPublishes	Association	EventPortDef
ConsumesDef	CORBAConsumes	Association	EventPortDef

Table 6 - Tagged value definition for CCM event ports

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAEvent	None	N/A	N/A	N/A
CORBAEventPort	None	N/A	N/A	N/A

Table 6 - Tagged value definition for CCM event ports

CORBAEmits	None	N/A	N/A	N/A
CORBAPublishes	None	N/A	N/A	N/A
CORBAConsumes	None	N/A	N/A	N/A

Constraints

- It's a binary association.

  self.connection→size=2

- It's an association between a «CORBAComponent» and a «CORBAEvent».

  self.connection→exists(participant.isStereotyped("CORBAComponent")) and self.connection->exists (participant.isStereotyped("CORBAEvent"))

- The « CORBAComponent » side is a composition.

  self.connection→exists(participant.isStereotyped("CORBAComponent") and aggregation = #composite)

- The «CORBAEvent» side role name must be defined.

  self.connection→exists (participant.isStereotyped("CORBAEvent") and name <> "")

- The «CORBAEvent» side cardinality must be 1.

  self.connection→exists(participant.isStereotyped("CORBAEvent") and multiplicity.min=1 and multiplicity.max=1)

«CORBAEmits» constraints

- The «CORBAEmits» side cardinality must be 1.

  self.connection→exists(participant.isStereotyped("CORBAEmits") and multiplicity.min=1 and multiplicity.max=1)

«CORBAPublishes» constraints

- The «CORBAPublishes» side cardinality can be 1 or more.

  self.connection→exists(participant.isStereotyped("CORBAPublishes") and multiplicity.max>1)

Example

The Event sink and event source model is the UML counterpart of the following IDL3 declaration:

```
eventtype E1 { };
eventtype E2 { };
eventtype E3 { };
component C1 {
   emits E1 source1;
   publishes E2 source2;
   consumes E3 sink1;
};
```

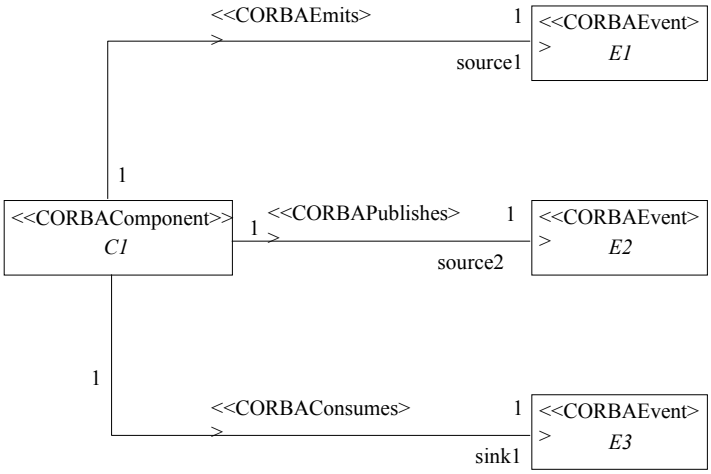

Figure 6 - Event sink and event source

Component Home

A Component home is described using a “CORBAHome” stereotyped class. This stereotype specializes the “CORBAInterface” stereotype. A component home must be associated to a component type. This relationship is made explicit using a “CORBAManages” stereotyped association between a “CORBAHome” and a “CORBAComponent”.

A “CORBAHome” can inherit from another “CORBAHome” (single inheritance) using a UML generalization. A “CORBAHome” can support several “CORBAInterface”. Each “CORBAInterface” generalization must be stereotyped ”CORBASupports”.

A “CORBAHome” can be associated with a primary key (necessary for persistent components). There is exactly one key instance for each (persistent component, home) instance couple. To enforce this constraint, the primary key is represented using a ”CORBAPrimaryKey” stereotyped AssociationClass.

A “CORBAHome” can own attributes and operations. The stereotype “CORBAHomeFactory” is used for the component factory operations. The stereotype “CORBAHomeFinder” is used for components finder operations. Table 7 describes the component home stereotypes. Table 8 defines the associated tagged values.

Table 7 - UML representation of CORBA home

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
HomeDef	CORBAHome	Class	InterfaceDef
PrimaryKeyDef	CORBAPrimaryKey	AssociationClass	ValueDef
FactoryDef	CORBAFactory	Operation	None
FinderDef	CORBAFinder	Operation	None

Table 7 - UML representation of CORBA home

HomeDef/ComponentDef and HomeImplDef/ ComponentImplDef relationships	CORBAManages	Association	None

Table 8 - Tagged value definition for a CORBA home

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAHome	None	N/A	N/A	N/A
CORBAPrimaryKey	None	N/A	N/A	N/A
CORBAFactory	None	N/A	N/A	N/A
CORBAFinder	None	N/A	N/A	N/A
CORBAManages	None	N/A	N/A	N/A

Constraints

- There is exactly one « CORBAManages » association for each Home.

  self.connection→select(isStereotyped("CORBAManages"))→size = 1

- The «CORBAHome» side cardinality must be 1..1

  self.connection→exists(participant.isStereotyped("CORBAHome")) and multiplicity.min=1 and multiplicity.max=1)

- The «CORBAComponent» side cardinality must be "0..n"

  self.connection→exists(participant.isStereotyped("CORBAComponent")) and multiplicity.min=0 and multiplicity.max=n)

- A « CORBAHome » can inherit from one « CORBAHome » at most.

  self.generalization→select(parent.isStereotyped("CORBAHome"))→size=1

- If "CORBAHome" h_1 inherits from "CORBAHome" h_2 and h_2 manages "CORBAComponent" C_2 then h_1 must manage C_2 or any other component C_1 *that* inherits from C2.

  let h1=self and let h2=self.generalization→select(parent.isStereotyped("CORBAHome")) and h2→notEmpty implies let C2=h2.connection→
  select(participant.isStereotyped("CORBAComponent")) and let C1=
  h1.connection→select(participant.isStereotyped("CORBAComponent")) and (C1 = C2 or C1.allParents→includes(C2))

- If « CORBAHome » h1 inherits from h2, and « CORBAHome » h2 is associated with primary key k2 then h1 must be associated with k2 or with a primary key k1 that inherits from k2.

  let h1=self and let h2=self.generalization→select(parent.isStereotyped("CORBAHome")) and h2→notEmpty implies let k2=h2.connection
  →select(isStereotyped("CORBAManages")).LinkToClass.ClassPart and let k1=self.connection
  →select(isStereotyped("CORBAManages")).LinkToClass.ClassPart and (k1 = k2 or k1.allParents->includes(k2))

- Each «CORBAHome» inheritance from a «CORBAInterface» must be stereotyped.

  self.generalization→forAll
  (g : Generalization \| g.parent.isStereotyped("CORBAInterface")
  implies g.isStereotyped("CORBASupports"))

«CORBAManages» constraints

- It's an association between a «CORBAHome» and a «CORBAComponent».

  self.connection→exists(participant.isStereotyped("CORBAHome")) and self.connection→exists(participant.isStereotyped("CORBAComponent"))

«CORBAPrimaryKey» constraints

- A primaryKey must own a public attribute.

  supplier.allAttributes→select (visibility=#public).notEmpty

«CORBAHomeFactory» constraints

- A « CORBAHomeFactory » operation has only input parameters.

  self. parameter→forAll(kind=#in)

- A «CORBAHomeFactory» can only be defined in a "CORBAHome".

  self.owner.isStereotyped("CORBAHome")

«CORBAHomeFinder» constraints

- A « CORBAHomeFinder » has only input parameters.

  self. parameter→forAll(kind=#in)

- A « CORBAHomeFinder » can only be defined in a "CORBAHome"

  self.owner.isStereotyped("CORBAHome")

Example

The following IDL3 example can be represented using the Component Home model.

```
abstract valuetype Key { };
component C1 { };
component C2 { };
home C1Home manages C1 primarykey Key {
   finder  findByName(in string name);
   factory create(in string name);
};
interface I1 { };
interface I2 { };
home myHome supports I1, I2 manages C2 { ... };
home C2Home : myHome manages C2 { };
```

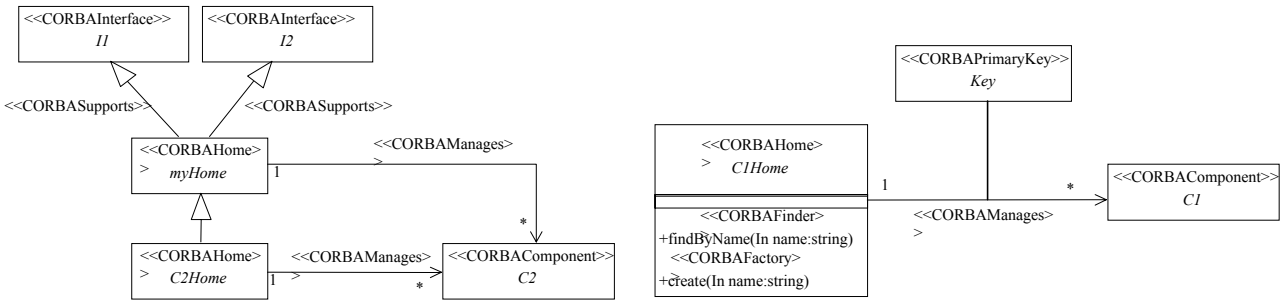

Figure 7 - Component Home

8.1.3 Metamodel to profile mapping

The mapping between the profile and the metamodel of the CCM is specified by giving the relation between the metamodel elements and the elements of the profile. It is shown in the following figures. The graphical relation "represents" means that the specific modelElement of the metamodel is represented by the associated construct(s) of the profile. For example, an instance of the metaclass ComponentDef is represented by a UML class stereotyped as CORBAComponent.

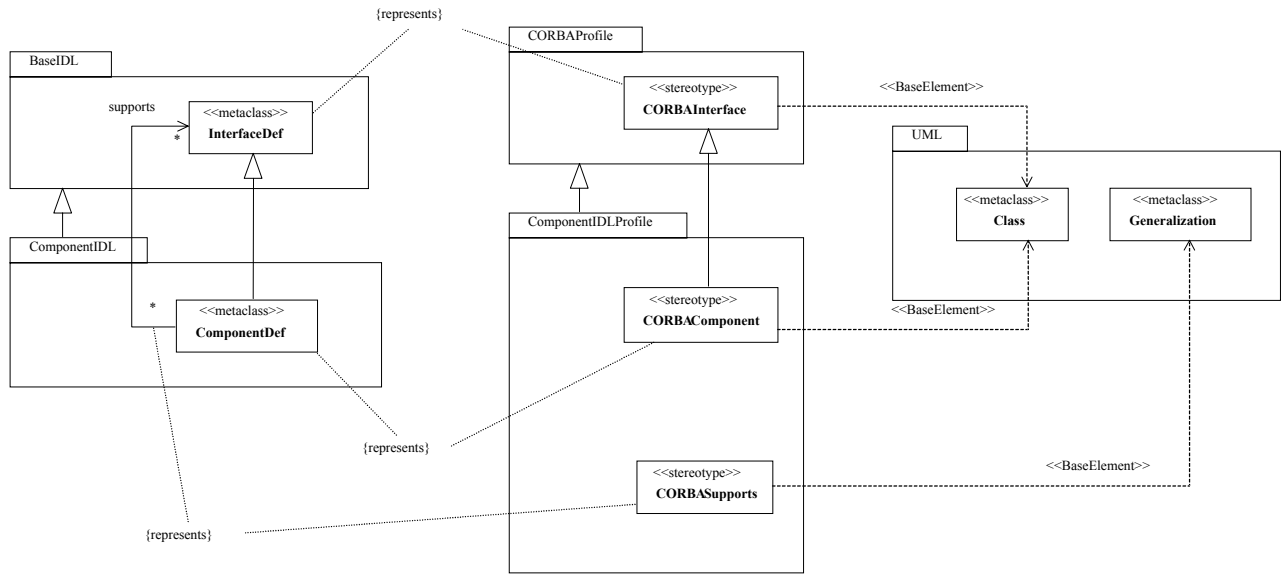

Figure 8 - Component mapping

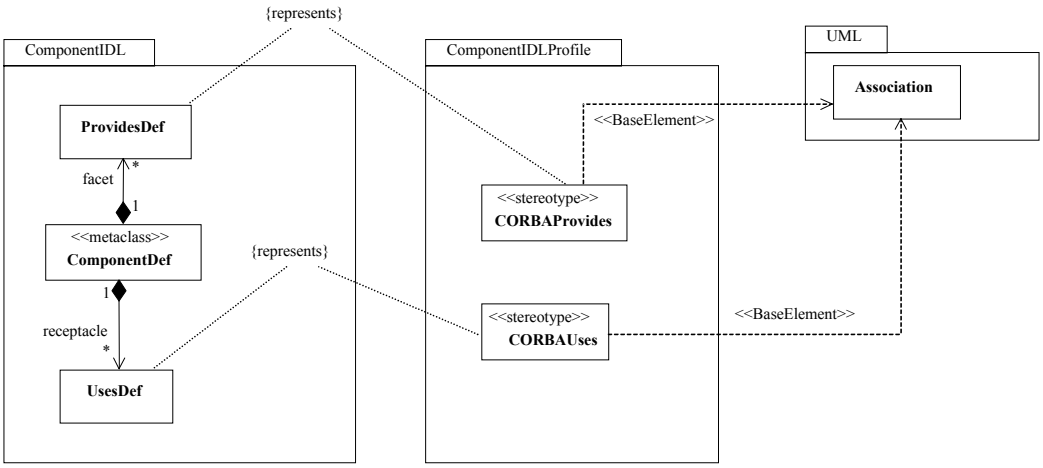

Figure 9 - Ports mapping

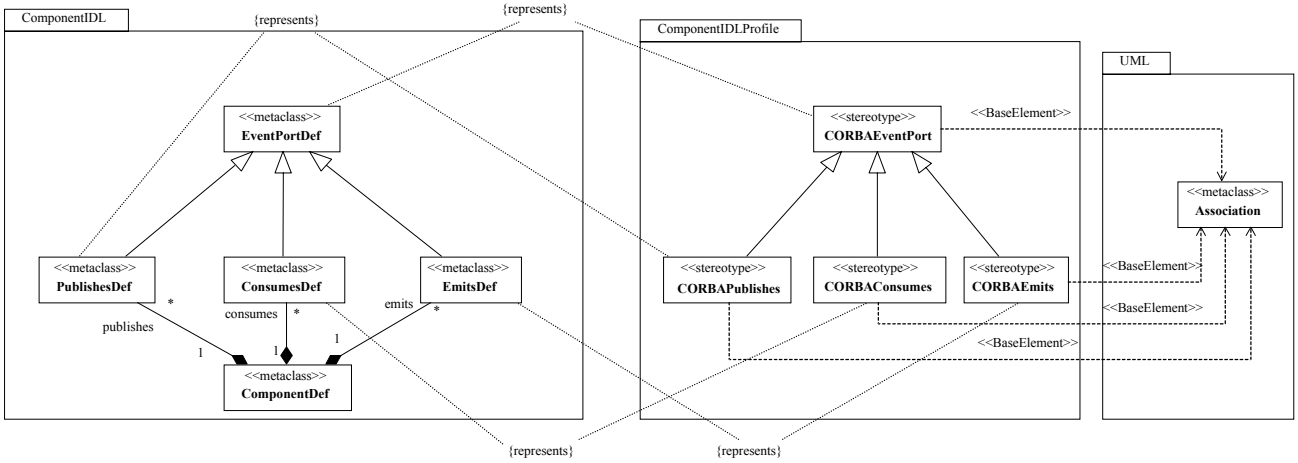

Figure 10 - Events mapping

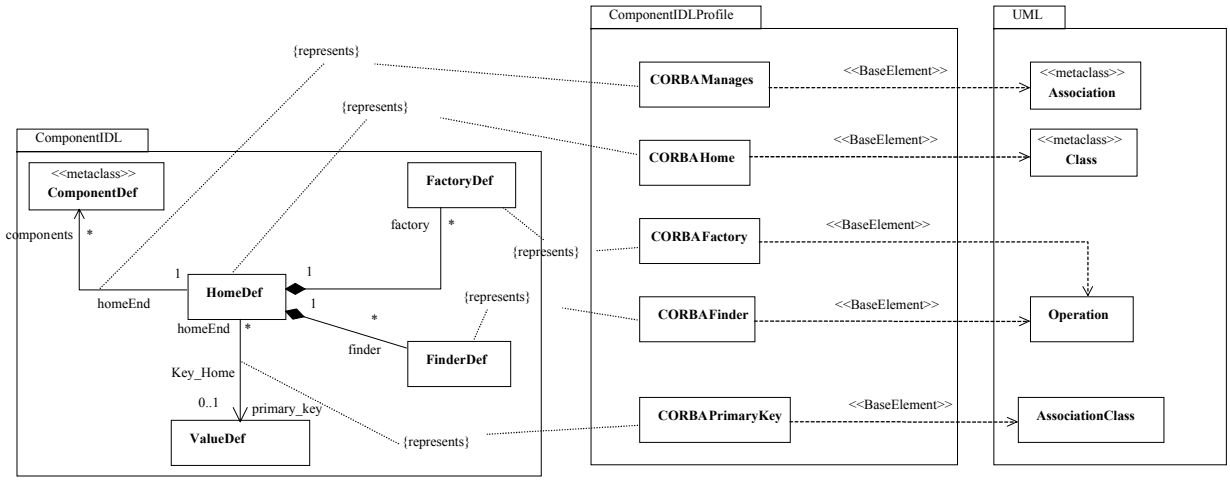

Figure 11 - Home mapping

8.2 UML Profile for CIF

8.2.1 CIF metamodel

The Figure 10 gives the structures of the CCM CIF concepts. A more detail description of these concepts can be found in the CCM OMG standard definition.

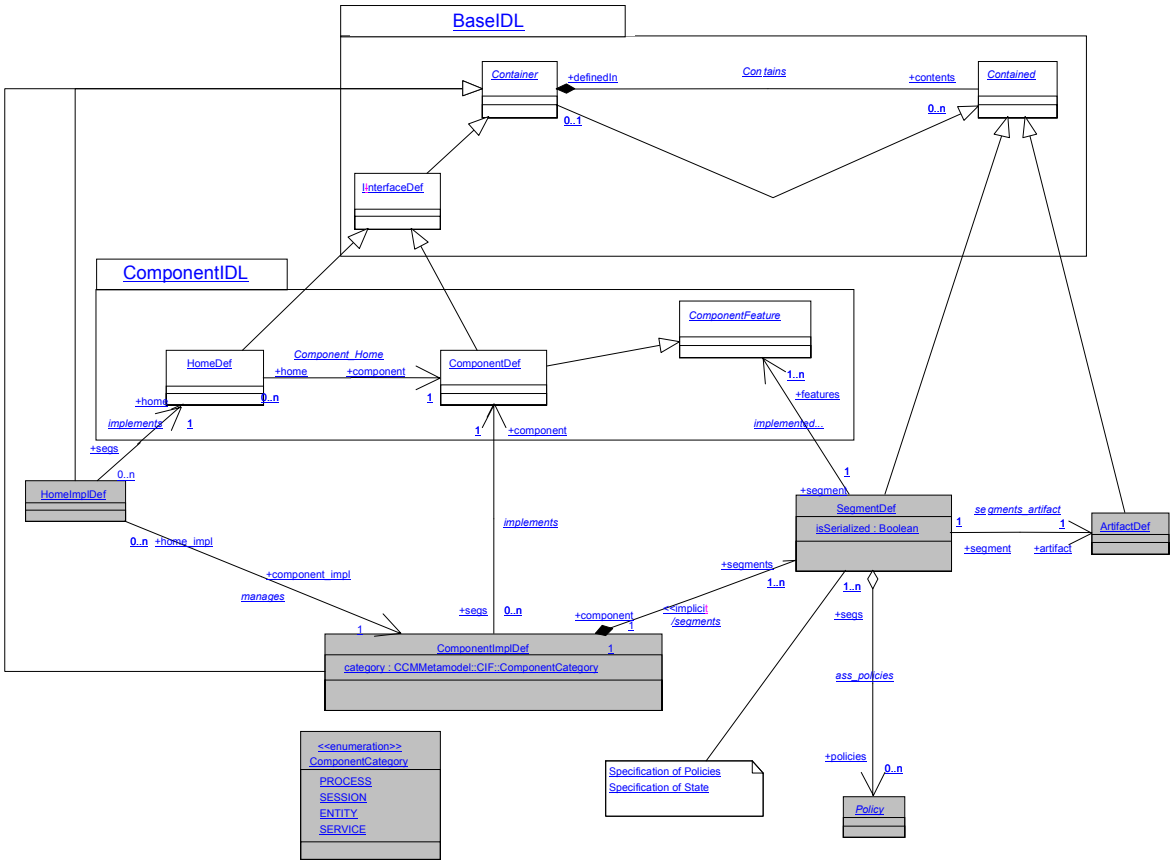

Figure 12 - CIF metamodel

The CIF metamodel defines additional metaclasses and associations to specify how a component has to be implemented.

The component implementation (ComponentImplDef) is used to model an implementation definition for a given component definition. It specifies an association to component definition to allow instances to point exactly to the component the instance is going to implement.

Segment type (SegmentDef) is used to model a segmented implementation structure for a component implementation. The behavior for each component feature (ComponentFeature) can be provided by a separate segment of the component implementation.

Segment type has in addition an association to an Artifact type (ArtifactDef) which is model of programming language constructs (e.g. classes) used to actually implement the behavior for component features.

Segment definitions modeled as instances of the Segment type may contain a set of policies (Policy), which have to be applied to realizations of the segment in the implementation code. These policies include for example activation policies for the artifact associated to a segment. The complete set of required policies is not defined yet, so the metamodel is flexible in this case. In the CCM Profile Policy concept is not for interest and is not considered further.

8.2.2 Profile definition

Component implementation

A component implementation is defined using a UML class with the stereotype “CORBAComponentImpl”.

Table 9 - UML representation of a CORBAComponent implementation

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
ComponentImplDef	CORBAComponentImpl	Class	N/A
ComponentImplDef/ ComponentDef and HomeDef/HomeImplDef relationships	CORBAImplements	Association	N/A

Table 10 - Tagged value definition for a CORBAComponent implementation

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAComponentImpl	category	ComponentCategory	“session”	“entity”, “session”, “process”, “service”
	containerType	String	not required	“Transient”, “Persistent”
	containerImplType	String	not required	“Stateless“, “Conversational“, “Durable“
	servanLifetimePolicy	String	not required	“Method”, “Transaction”, “Component”, “Container“
	transactionPolicy	String	not required	“NOT_SUPPORTED”, “REQUIRED”, “SUPPORTS”, “REQUIRES_NEW”,“MANDATORY”, “NEVER”
	securityPolicy	String	not required	User specific
	eventPolicy	String	not required	“Normal”, “Default”, “Transaction”
	persistenceMechanism	String	not required	“CORBA”, “User”

<<CORBAComponentImpl>> contraints

- There is an association between <<CORBAComponentImpl>> and <<CORBAComponent>>.

  self.connection→ exists(participant.isStereotyped("CORBAComponentImpl")) and self.connection→exists(participant.isStereotyped("COR-

BAComponent"))

- The only classes that are allowed to be contained by a <<CORBAComponentImpl>> are classes with the stereotype <<CORBASegment>>.

  self.connection→
  exists(participant.isStereotyped("CORBAComponentImpl") and aggregation = #composite and aggregation.participant.isStereotyped("CORBASegment"))

<<CORBAImplements>> contraints

- A <<CORBAComponentImpl>> always has exactly one <<CORBAComponent>> associated while each <<CORBAComponent>> might be implemented by different types of <<CORBAComponentImpl>>.

  self.connection→
  exists(participant.isStereotyped("CORBAComponentImpl") and multiplicity.min=1 and max=*)
  self.connection→exists(participant.isStereotyped("CORBAComponent") and multiplicity.min=1 and max=1)

- Each << CORBAHomeImpl >> in a model implements exactly one <<CORBAHome>>.

  self.connection→exists(participant.isStereotyped("CORBAHomeImpl") and multiplicity.min=1 and max=1)
  self.connection→exists(participant.isStereotyped("CORBAHome") and multiplicity.min=1 and max=1)

<<CORBAManages>> constraints

- It's an association between a «CORBAHomeImpl» and a «CORBAComponentImpl».

  self.connection→
  exists(participant.isStereotyped("CORBAHomeImpl")) and self.connection→
  exists(participant.isStereotyped("CORBAComponentImpl"))

- Each << CORBAHomeImpl >> manages exactly one <<CORBAComponentImpl>>, this relation is modeled by the association <<CORBAManages>>.

  self.connection→
  exists(participant.isStereotyped("CORBAComponentImpl") and multiplicity.min=1 and max=1)

Home implementation

A home implementation of a component is defined using a UML class with the stereotype "CORBAHomeImpl".

Table 11 - UML representation of a CCM home implementation

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
HomeImplDef	CORBAHomeImpl	Class	InterfaceDef

Table 12 - Tagged value definition for a CORBAComponent implementation

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBAHomeImpl	None	N/A	N/A	N/A

<<CORBAHomeImpl>> contraints

- For each instance x of <<CORBAHomeImpl>> the instance of <<CORBAComponent>>, which is associated to the

instance of <<CORBAHome>> associated to x is the same instance as the instance of <<CORBAComponent>> associated to the instance of <<CORBAComponentImpl>>, which is associated to x.

self.home.component = self.component_impl.component

- The life cycle category of the <<CORBAComponentImpl>> must be "***entity"*** or "***process"*** if the component implementation is segmented.

self.segments>1 implies (self.category=ENTITY or self.category=PROCESS)

Example

The following IDL3 example describes a representation of the minimal form as a composition (without Managed Storage), which specifies a unit of component implementation.

```
component ExmplCom {};
home ExmplHome manages ExmplCom {};
composition session ExmplComImpl {
    home executor ExmplHomeImpl{
implements    ExmplHome;
manages    ExmplComSessionImpl;
};
};
```

Using the UML Profile for CCM the described composition above can be represented with the following UML model:

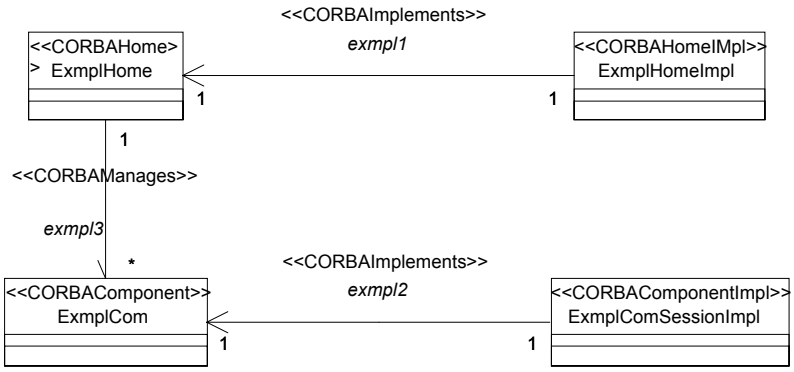

Figure 13 - CIDL composition: unit of a component implementation in CCM

Segment and artifact

A segment of a component implementation is defined using a UML class with the stereotype "CORBASegment".

An artifact of a component implementation is defined using a UML class with the stereotype "CORBAArtifact".

Table 13 - UML representation of a CCM segment and artifact

Metamodel element	Stereotype name	UML Base Class	Parent (from Metamodel)
SegmentDef	CORBASegment	Class	Contained
ArtifactDef	CORBAArtifact	Class	Contained

Table 14 - Tagged value definition for a CCM segment

Stereotype name	Tagged value	Tagged value type	Default value	Tagged value Definition
CORBASegment	isSerialized	Boolean	“no”	“yes”, “no”
	features	List <<String>>	N/A	N/A

<<CORBASegment>> contraints

- <<CORBASegment>> classes are always contained in <<CORBAComponentImpl>>.

  self.definedIn.oclIsTypeOf(ComponentImplDef)

<<CORBAArtifact>> contraints

- The only allowed *Container* for *ArtifactDef* is *ModuleDef*.

  self.definedIn.oclIsTypeOf(ModuleDef)

Example

The following IDL3 example extends the previous example to illustrate segmented executors (component implementation). A segmented executor `ExmplComEntityImpl` is a set of physically distinct artifacts `ExmplFacet1` and `ExmplFacet2`.

```
component ExmplCom {};
home ExmplHome manages ExmplCom {};
composition entity ExmplComImpl {
    home executor ExmplHomeImpl{
implements    ExmplHome;
manages    ExmplComEntityImpl{
  segment ExmplSeg1{
provides (ExmplFacet1); };
  segment ExmplSeg2{
provides (ExmplFacet1); };
};
  };
};
```

Using the UML Profile for CCM the described composition above can be represented with the following UML model:

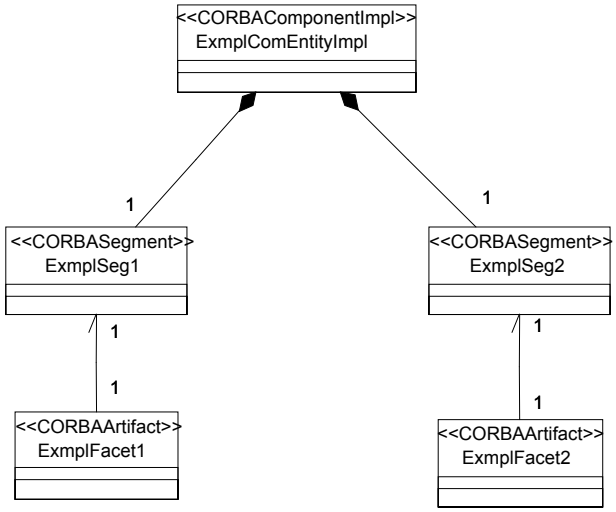

Figure 14 - Segments and artifacts

8.2.3 Metamodel to profile mapping

The mapping between the profile and the metamodel of the CCM is specified by giving the relation between the metamodel elements and the elements of the profile. It is shown in the following figure. The graphical relation "represents" means that the specific modelelement of the metamodel is represented by the associated construct(s) of the profile. For example, an instance of the metaclass ComponentImplDef would be represented by a UML class stereotyped as CORBAComponentImpl.

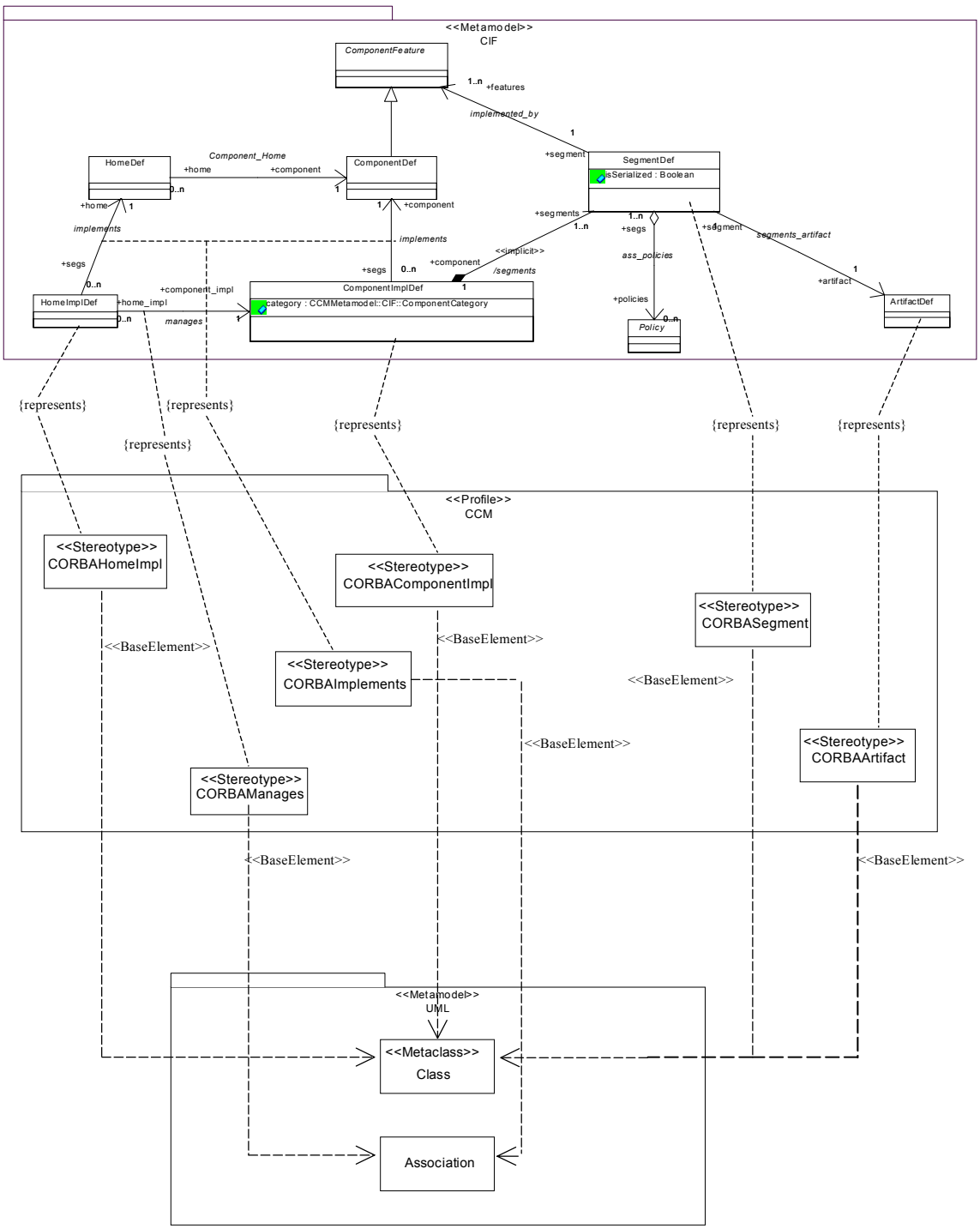

Figure 15 - CIF metamodel to CCM Profile and CCM Profile to UML mappings

9 Profile Illustration with the dinning philosopher

9.1 Example scenario description

The example scenario includes three different types of components:

1. Philosopher.
2. Fork.
3. Observer.

A configurable number of philosophers (active components) are sitting around a table. Philosophers perform actions: thinking, eating and sleeping. They do not need any resources in order to think or to sleep, but they need two forks in order to eat, one for the left hand and one for the right hand. Therefore, before starting to eat, a philosopher tries to get two forks.

An observer will be notified by all philosophers in case of an activity change (when a philosopher starts eating, starts thinking or starts sleeping). Furthermore, the critical state of getting hungry is notified to an observer as well.

9.2 Type definition

The example use the following IDL3 basic types and exceptions:

```
module Dinner {
exception InUse{};
Exception TooMuchPhilosopher{};

typedef string PhilosopherName;
typedef enum PhilosopherState {
    EATING,
    THINKING,
    HUNGRY,
    STARVING,
    DEAD
};
}
```

The Type and exception definition model gives the same information using the CORBA2 UML profile.

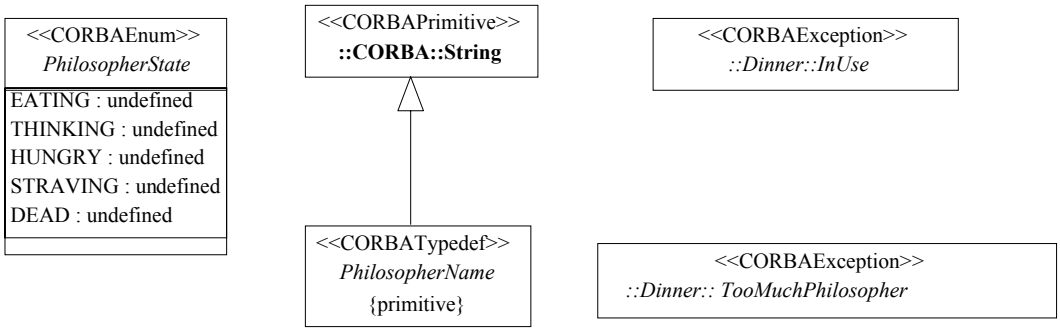

Figure 16 - Type and exception definition

9.3 Interface definition

The interfaces needed for port definitions are the following ILD3 definition:

```
Module Dinner {

interface Registration {
        PhilosopherName register()raises (TooMuchPhilosopher);
    };
interface Fork {
        void get() raises (InUse);
        void release();
    };
};
```

The Interfaces definition model gives the same information using the CORBA2 UML profile.

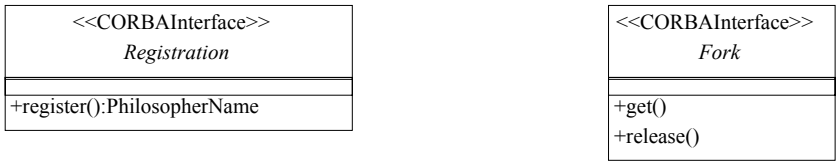

Figure 17 - Interfaces definition

9.4 Component definition

The IDL3 component definitions are the following:

```
module Dinner {
eventtype StatusInfo {
    public PhilosopherName  name;
    public PhilosopherState state;
```

```
    public long secondesSinceLastMeal;
    public boolean hasLeftFork;
    public boolean hasRightFork;
};

    component Philosopher {
        readonly attribute long metabolicRate;
        uses Fork leftFork;
        uses Fork rightFork;
        uses Registration registration;
        publishes StatusInfo statusInfo;
    };

    component Fork {
        provides Fork one_fork;
    };

    component Registrator supports Registration {
    };

    component Observer {
        consumes StatusInfo info;
    };
};
```

The Philosopher external view model, the Fork component external view model, the Registrator external view model, and Observer external view model give the same information using the CCM UML profile.

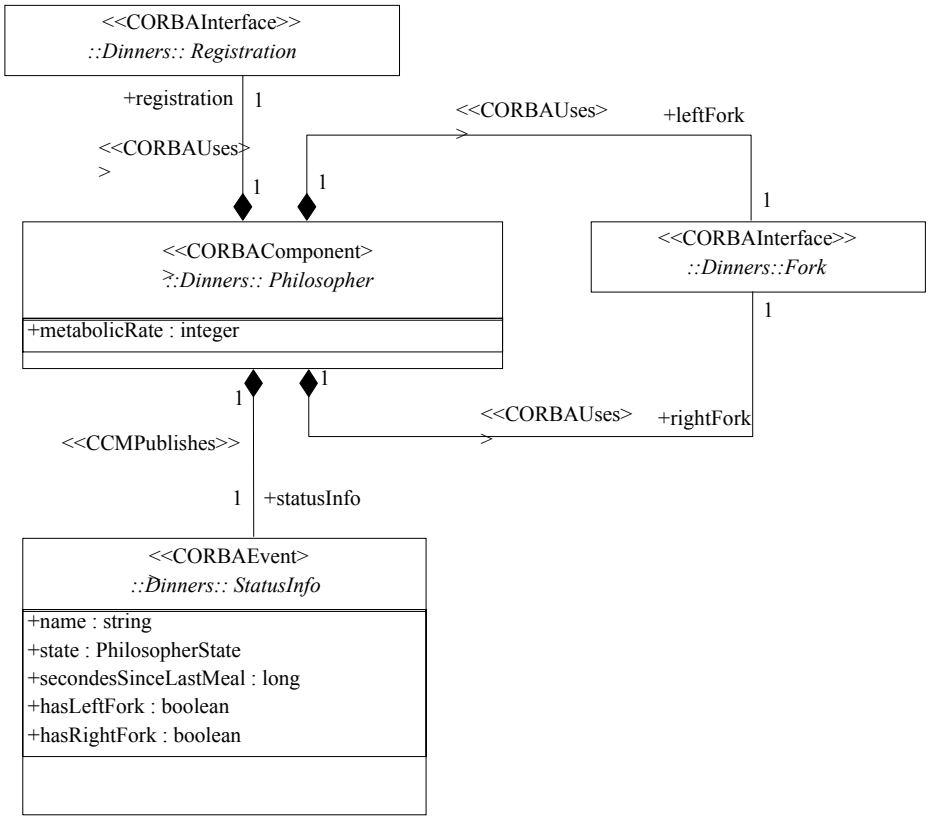

Figure 18 - Philosopher external view

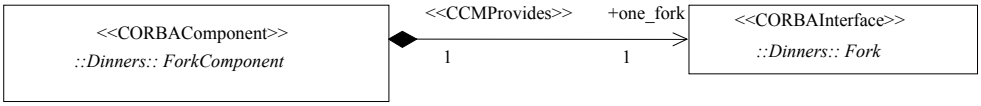

Figure 19 - Fork component external view

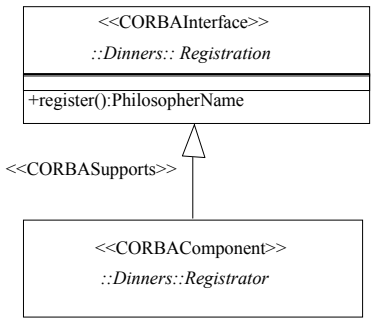

Figure 20 - Registrator external view

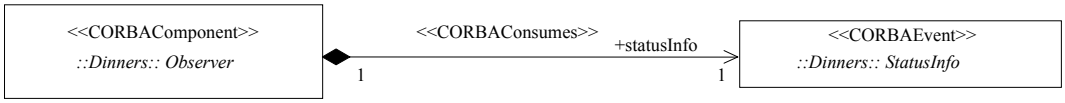

Figure 21 - Observer external view

9.5 Home definition

The IDL3 home definitions are the following:

```
module Dinner {
home RegistratorHome manages Registrator {};
home PhilosopherHome manages Philosopher {};
home ForkHome manages Fork {};
home ObserverHome manages Observer {};
};
```

The Registrator home, Philosopher home, Fork home and Observer home give the same information using the CCM UML profile.

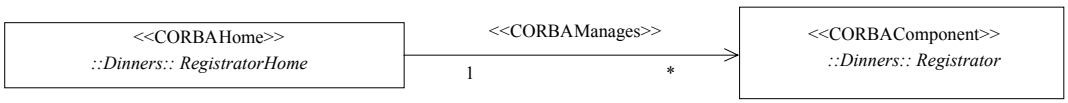

Figure 22 - Registrator home

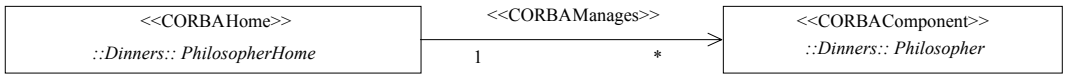

Figure 23 - Philosopher home

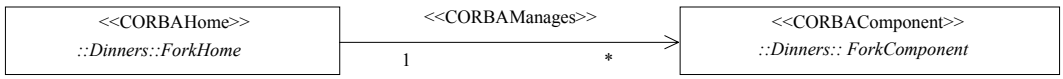

Figure 24 - Fork home

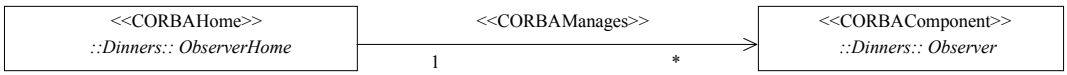

Figure 25 - Observer home

9.6 Component implementation definition

The IDL3 component implementation definition is a composition. For each component the following composition descriptions were defined:

```
module Dinner {

composition session PhilosopherImpl {
home executor PhilosopherHomeImpl {
implements PhilosopherHome;
manages PhilosopherSessionImpl;
};
};

composition entity ForkImpl {
home executor ForkHomeImpl {
implements ForkHome;
manages ForkEntityImpl {
segment Seg { provides the_fork; }
};
};
};

composition session ObserverImpl {
home executor ObserverHomeImpl {
```

```
implements ObserverHome;
manages ObserverSessionImpl;
};
};

composition session RegistratorImpl {
home executor RegistratorHomeImpl {
implements RegistratorHome;
manages RegistratorSessionImpl;
};

};
};
```

The implementations for the components: Fork, Philosopher, Registrator and Observer in form of compositions described above have following representations using the UML Profile for CCM:

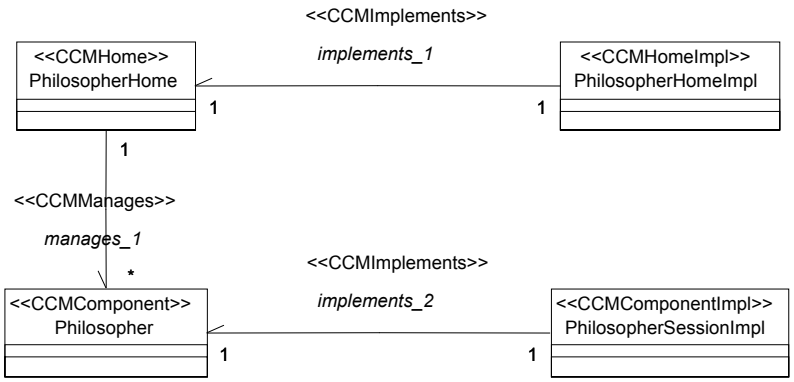

Figure 26 - The Composition model for the Philosopher component

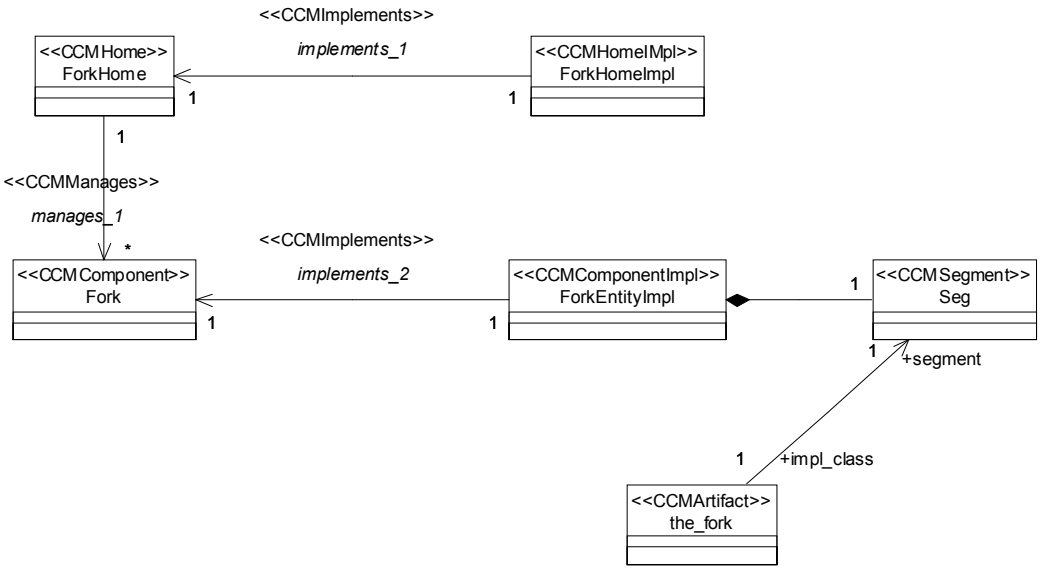

Figure 27 - The Composition model for the Fork component

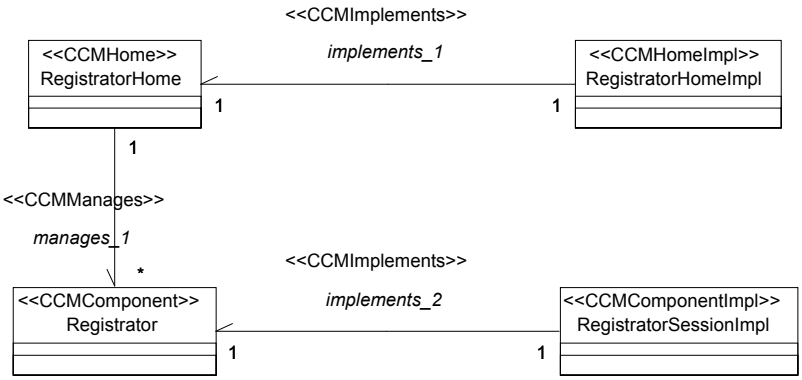

Figure 28 - The Composition model for the Registrator component

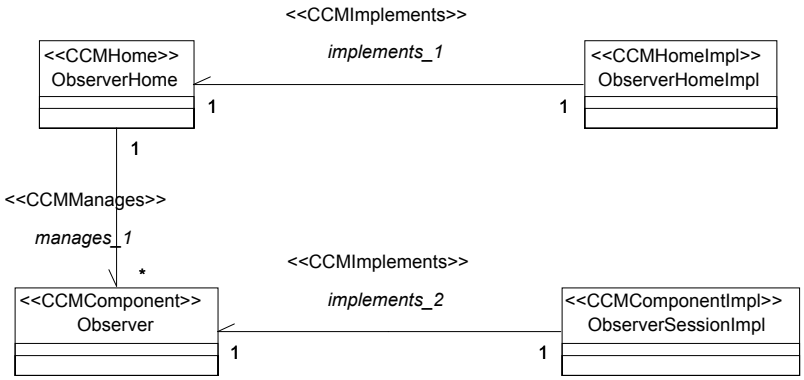

Figure 29 - The Composition model for the Observer component

A References

[1] Meta Object Facility (MOF) Specification, Version 1.4, OMG document ptc/2001-10-04

[2] MOF 2.0 to OMG IDL Mapping RFP, OMG document ad/2001-11-07

[3] MOF 2.0 Core RFP, OMG document: ad/2001-11-05

[4] CORBA Components Specification, OMG TC Document formal/02-06-65

[5] www.puml.org/mml

[6] Unified Modeling Language (UML) Specification, Version 1.5, OMG TC Document formal/03-03-01