ECONET CCMM

PA

16/04/2008

I. Starting point

Collected documents available on the Econet Wiki or SVN repository.

I.1. Prague Workshop Report

- SOFA metamodel (parts) p. 31, variant p. 52
- Mapping concepts (component elements, annotations) p. 37
- Model comparison p. 38-39
- Common metamodel p. 39
- Annotations (p. 40-41, p. 47-48

I.2. Concrete Metamodels

- SOFA metamodel
- Kmelia metamodel

I.3. Abstract Metamodels

- CMM metamodel (PA – november 2007)
- CMM metamodel (PH from another project – march 2008)
- CCMM_1.0_ecore (VP – march 2008)

I.4. Normative or specific model

This is a inspiration source for finding the core element and relations and to name them. For example we could define special profiles.

- Ecore from EMF project
- OMG UML 2.1 (UML 2.0, UML 1.5)

II. Model V1.0

This is a draft version.

From the above document and models I tried to get a synthesis model.

II.1. Modelling concepts and organisation

I added some basic and core concepts (elements, types…) that we find in most of abstract and concrete models.

Therefore I organised the metamodel in three layers :

- Basic layer : common concepts that overlap components (to be connected with usual core metamodels (UML, EMF).
- Common Component layer (an abstraction of what we find in general component models)
- Specific Component layer (for concrete models)

Many WFR will apply to concrete model layers especially to restrict the element combinations.

I also tried to present the diagrams in a layered presentation.

II.2. Conflicting concepts

In order to solve them (except noun conflicts), I propose to draw a specialisation hierarchy.

a. Interface

Can be a (restricted) Classifier, a NamedElement (Sofa, KADL) or simply an Element (Kmelia).

Can be separate between Provided/Required or not.

I made a specialisation hierarchy..

In other approaches we have also ports.

b. Operation

As a behavioural feature denoting some functional computation with or without dynamic features.

Can be simply an Operation (Sofa, KADL) or a complex Entity (Kmelia)

I took NamedElement.

c. Protocol

Can be simply associated to a component (Sofa, KADL), an interface or a service (Kmelia)

d. Service

Can be simply an Operation (Sofa, KADL) or a complex Entity (Kmelia)

I took NamedElement.

e. Constraints/Predicate/Properties

Can be used to writie assertions, classify concepts…

I put them in a special package.

f. Pre/post conditions

Set in operations as optional features.

g. Architecture/Assembly – Connectors-Bindings

I defined a Architecture type that denotes patterns of assembling.

Connectors are simply bindings. The question is about what we bind : this can be interfaces or services. A CCMM should accept boths. I tried to make it more abstract using EndPoints and specialised endpoints. An endpoint has a target which is either an interface or an operation (service).

II.3. Modelling issues

This is a short summary of discussion points.

1. Represent Java concepts (like JMI model)

   ⇒ NO

2. Represent model management

   ⇒ NOT YET

   ⇒ Only a package

3. Represent component instances

   ⇒ NOT YET

   ⇒ Only a package

4. Represent annotations

   ⇒ YES

Attributes

- YES

Special Package and Relations

- YES ? On going work

5. Represent non functional requirements

   ⇒ NO

6. Represent Ecore

   ⇒ NOT EXACTLY but inspired

   ⇒ Basically UML 2

II.4. Comments

This an on going work

⇒ Removed UML qualified associations

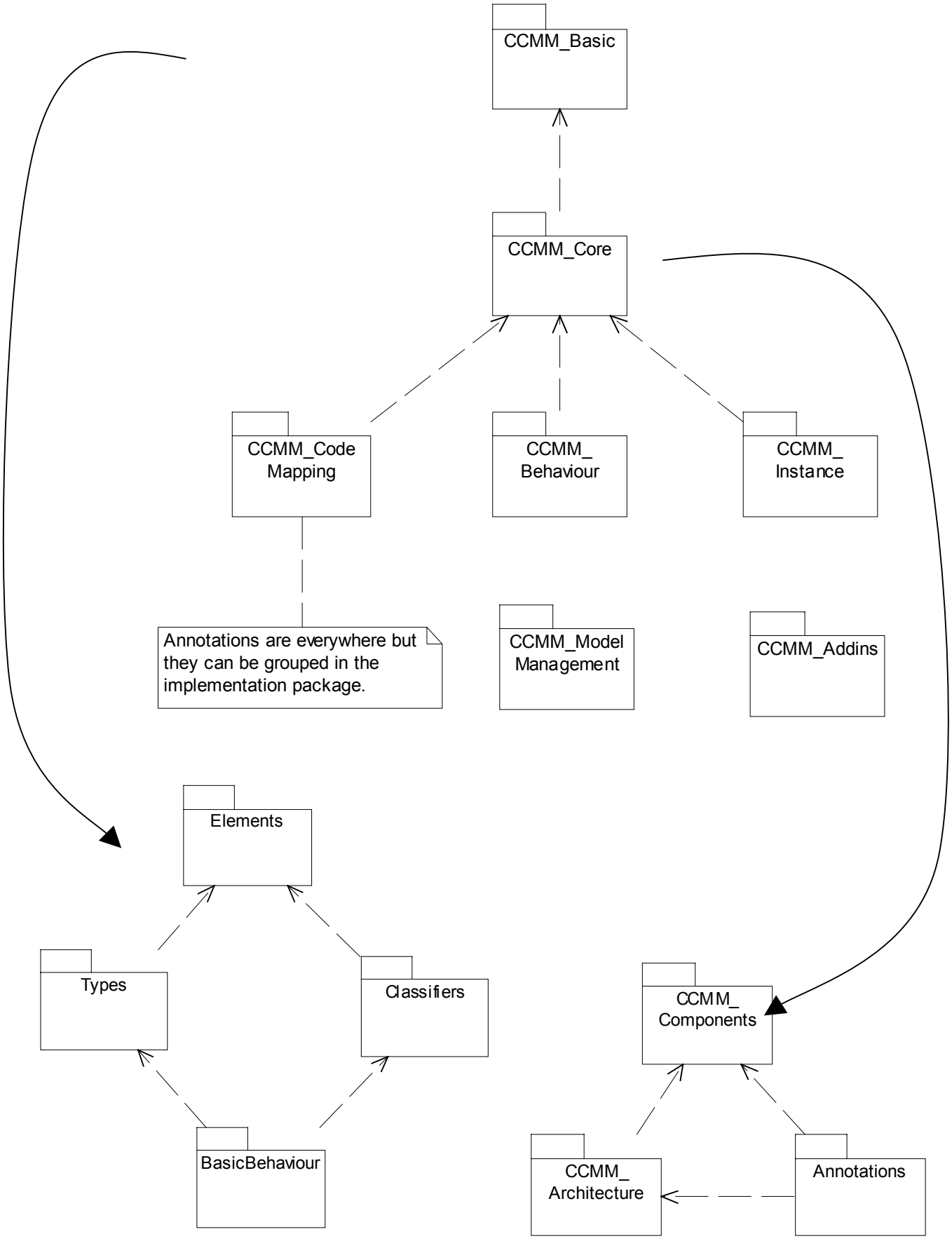
⇒ not finished

⇒ not validated by Econet group

II.5. Constraints

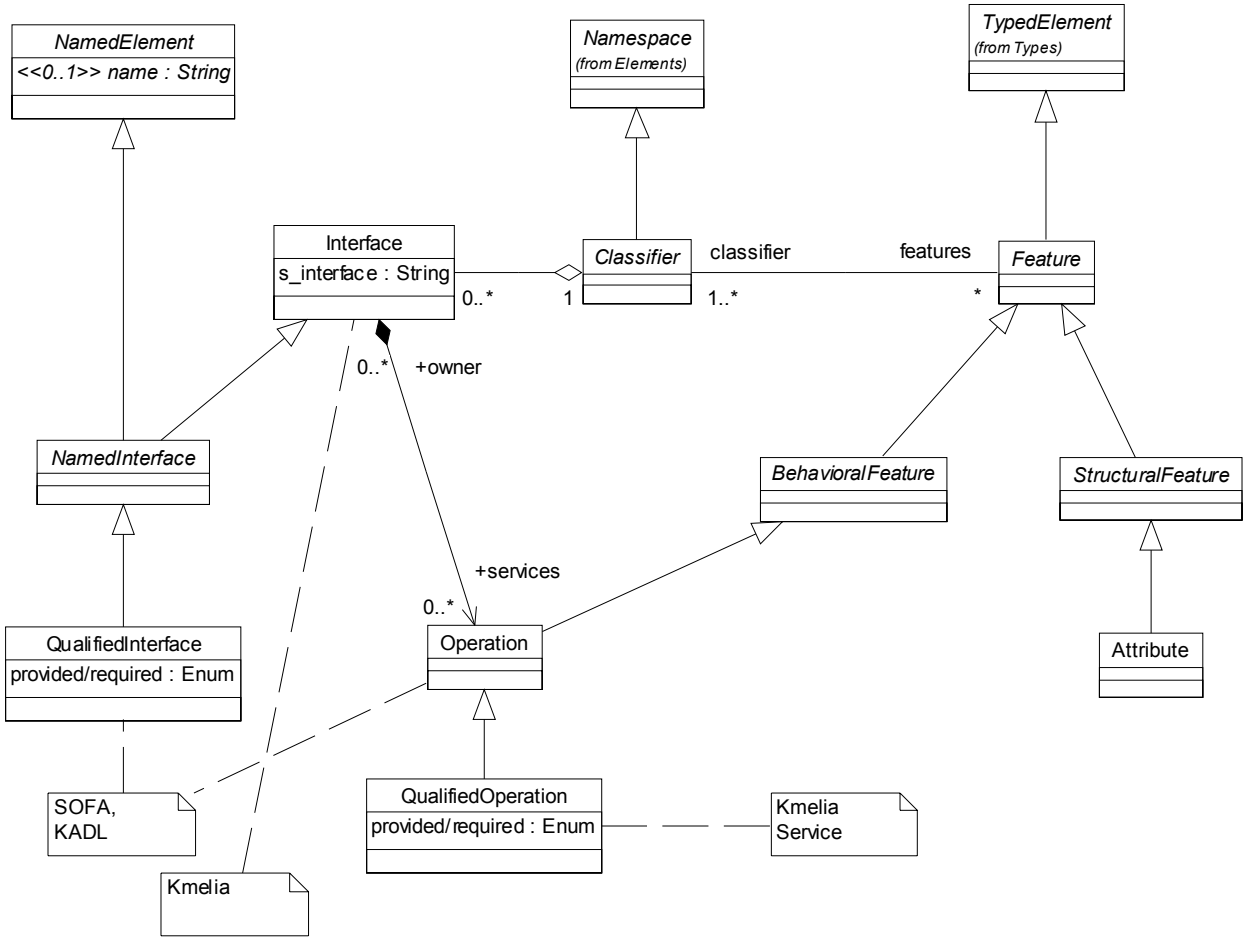
⇒ To be continued in the next section.

II.6. Overview



III. Constraints

This is a preliminary work that should be completed later once the model will be validated.

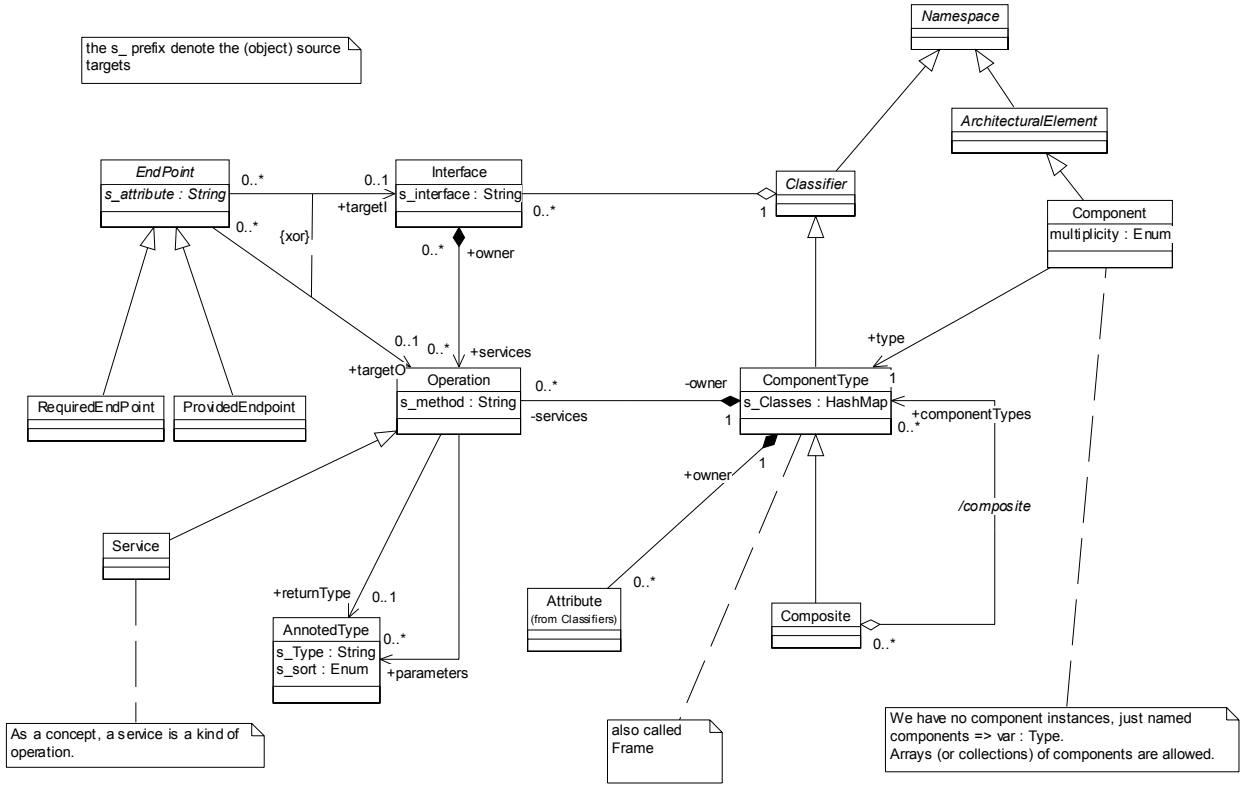


C1. In an interface, the provided/required qualification must be consistent.

```
Context Interface
inv qualif-consistency:
self.isTypeOf(QualifiedInterface) implies
    self.services->forAll(s |
        if(s.isTypeOf(QualifiedOperation))
        then s.provide/required = self.s.provide/required
        endif   )
```

C2. For each interface of a classifier there must have a provided/required qualification.

```
Context Classifier
inv qualif-exists:
self.interface->forAll(i | i.isTypeOf(QualifiedInterface) or
    i.services->forAll(s | s.isTypeOf(QualifiedOperation)))
```



C3. The operation of an interface, are those of its component type.

```
Context ComponentType
inv interf-op:
self.services->includesAll(self.interface.services)
```

C4. Recursive component composition is not possible at the type level. We could work only at an instance level (C5) providing a more flexible constraint.

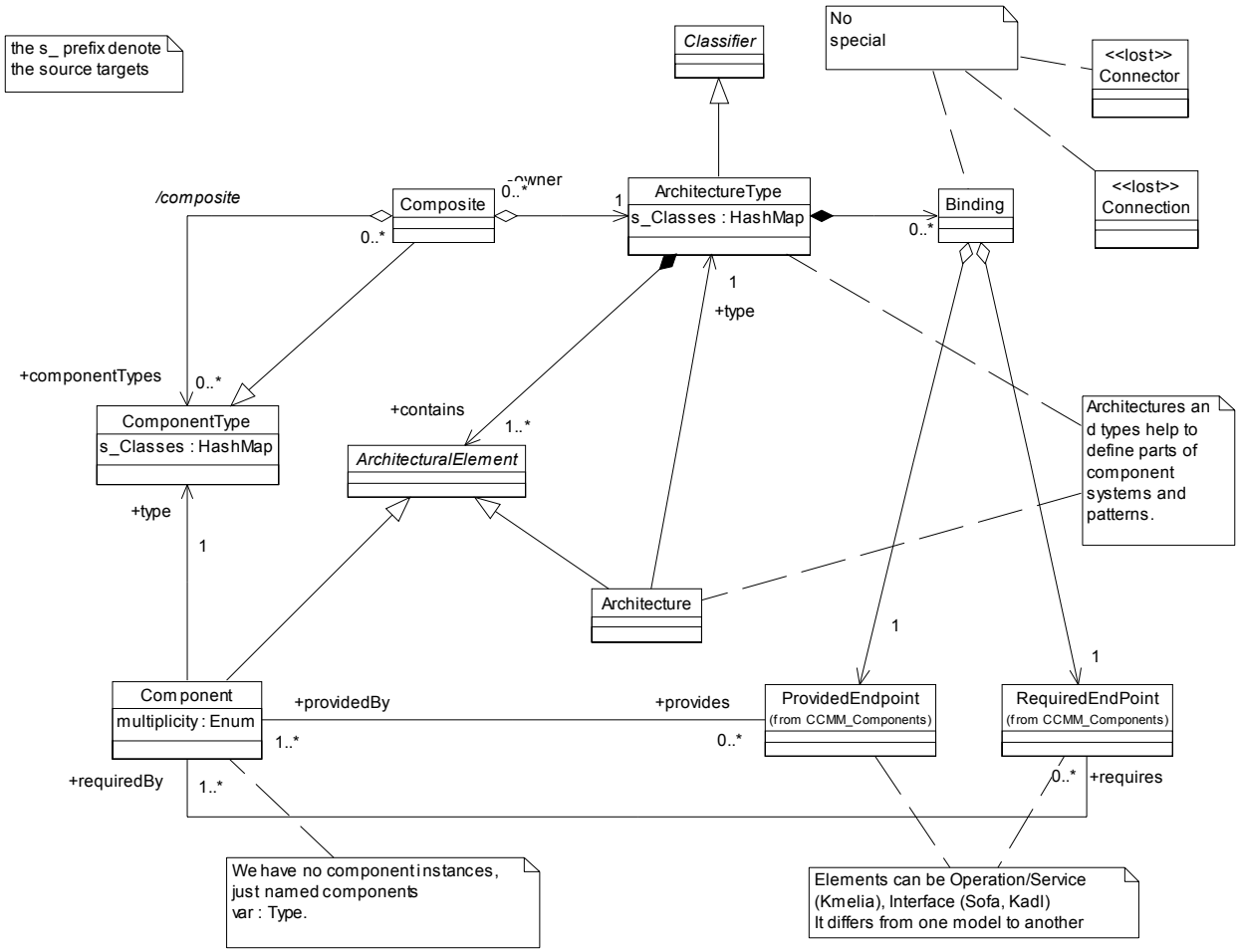
```
Context Composite::myComponents():Set(ComponentType)
pre: true
post:
    let mct = self.componentTypes in
    mct->iterate(ct ; result : Set(ComponentType) = mct |
        if ct.isTypeOf(Composite) then
            result->union(ct.myComponents())
        else
            result
        endif)
Context Composite
inv recursive:
    self.myComponents()->excludes(self)
```

C5. Adapt C4 to the component instance (true instance or simple variable association) level.

C6. Adapt C4 to architectures

C7. EndPoints are either interfaces or operations. (this can be specified differently in the metamodel).

```
Context EndPoint
inv exclusion:
    self.targetO->isEmpty xor self.targetI->isEmpty
inv consistency:
    let epi : Set(EndPoint) = EndPoint.allInstances in
    epi.targetO->isEmpty xor epi.targetI->isEmpty
```



C8. The component endpoints are consistent with their types.

```
Context Interface
inv comp-ep-consistency
let ct : Set(ComponentType) = self.provides.targetO.owner->union(
        self.provides.targetO.owner)  in
    ct->size() = 1 and ct->includes(self.type)
```

C9. etc.

IV. Toward Model V1.1

Next step will be model v1.1. Everyone is invited to propose (meta) model evolutions.