An I-Module Generator for the infoAsset© Broker

Mapping CoCoMa-Assets to infoAsset Broker Assets

Bachelor Thesis of Gerald Mofor

2004 Hans-Werner Sehring

Requirement

Task:
Mapping assets according to the Asset Definition Language to asset objects for the infoAsset-Broker (bAssets).

Solution:
A generator for the asset language model compiler.
Here: design of the mapping from assets to bAssets.

Creation of Asset Implementation Classes: General Approach

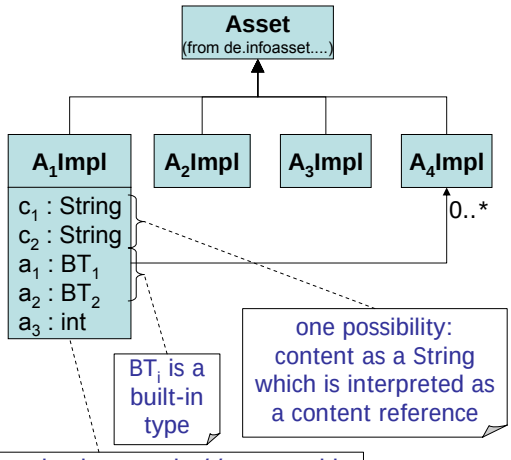
Asset definitions of the form [content | concept] are mapped to classes for infoAsset Broker assets (bAssets):

- The mapping of content objects is discussed on corresponding slides.
- Characteristics are mapped to bAsset attributes.
- Relationships become either attributes (one-to-one) or `Relationships` (many-to-many).
- Constraints have to be checked in the bodies of setter methods.

Creation of Asset Implementation Classes: Content, Characteristics, and Relationships

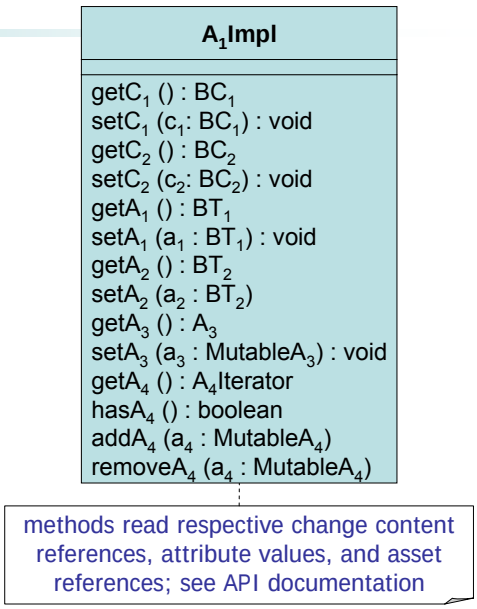
Example class A_1:

```
class A1 refines A2
[
  c1 : C1
  c2 : C2
  ...
|
  characteristic a1 : T1
  characteristic a2 : T2
  relationship a3 : A3
  relationship a4 : A4*
  ...
  constraint b
    onviolation s
]
```



Creation of Asset Implementation Classes: Access Methods

```
class A1 refines A2
[
  c1 : C1
  c2 : C2
  ...
|
  characteristic a1 : T1
  characteristic a2 : T2
  relationship a3 : A3
  relationship a4 : A4*
  ...
  constraint b
    onviolation s
]
```

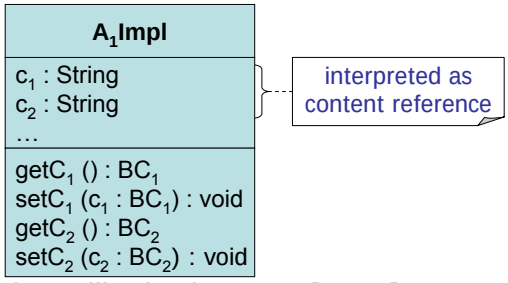


Creation of Asset Implementation Classes: Content Objects

Alternatives:

- The content compartment is mapped to a set of string-valued attributes which are interpreted as content references [RM]:

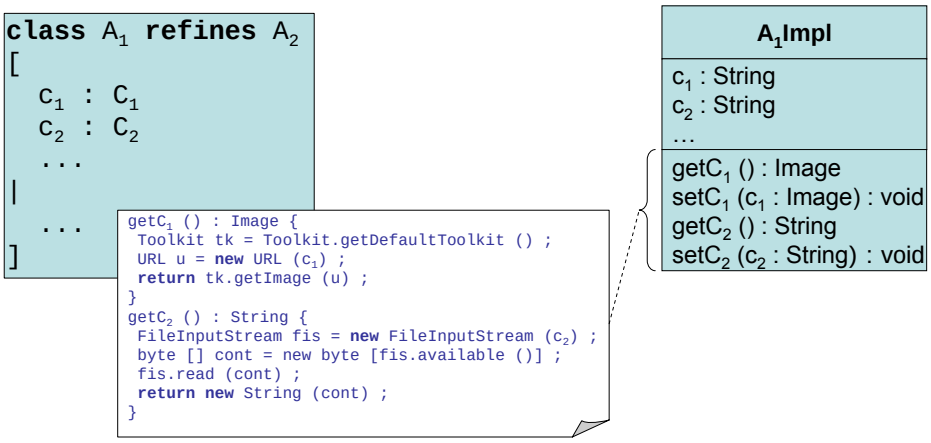
```
class A1 refines A2
[
  c1 : C1
  c2 : C2
  ...
|
  ...
]
```



- `DocumentLocation` objects like in the WEL [HWS]
- BLOBs and corresponding handlers etc. [PH]
- extension of the Broker class `Document` [US]

Creation of Asset Implementation Classes: Content Access Methods

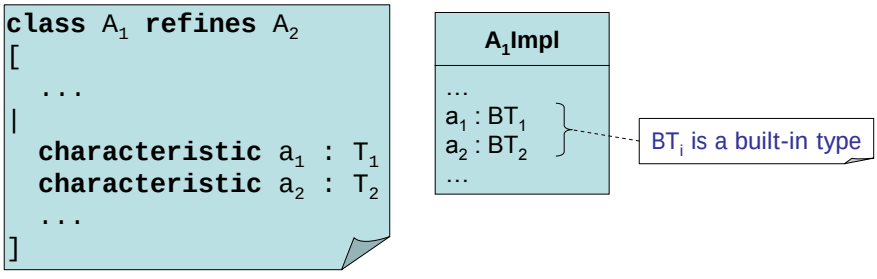
Example for the first alternative (content references encoded as string): Let C_1 be `java.awt.Image` for pictures accessible under a URL and C_2 be `String` for full text stored in files:



Creation of Asset Implementation Classes: Characteristics

As object types all those primitive types are allowed which are offered by the infoAsset Broker: `int`, `Date`, `String`, `ByteStream`, `CharacterStream` and `Timestamp`.

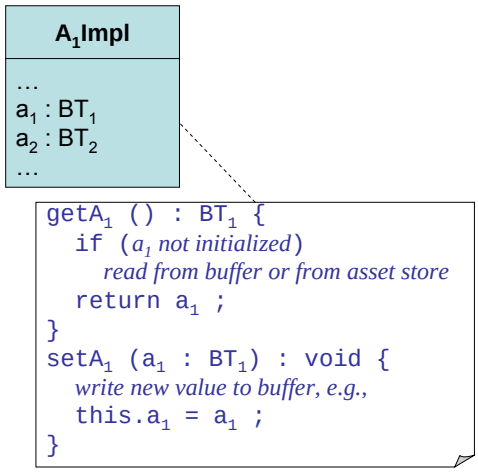
For other types: either introduction of attribute classes and adding the corresponding access methods or mapping to available types.



Creation of Asset Implementation Classes: Characteristics Access Methods

Access methods operate on in-memory copies of the assets' data.

```
class A1 refines A2
[
  ...
|
  characteristic a1 : T1
  characteristic a2 : T2
  ...
]
```

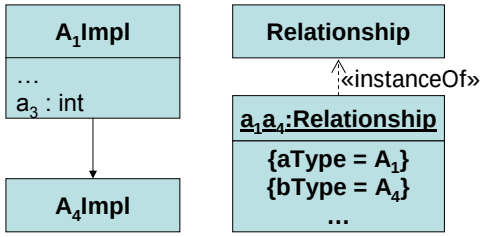


Creation of Asset Implementation Classes: Relationships

Asset relationships are mapped to:

- IDs as foreign keys in the case of one-to-one relationships, and
- relationship bAssets in the case of many-to-many relationships.

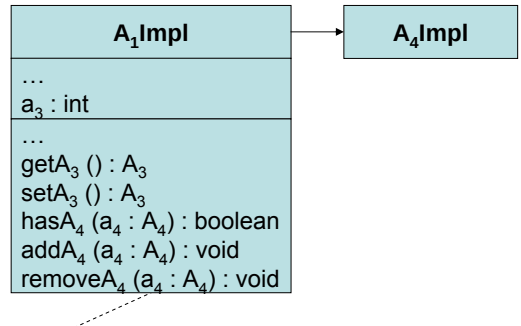
```
class A1 refines A2
[
  ...
|
  ...
  relationship a3 : A3
  relationship a4 : A4*
  ...
]
```



Creation of Asset Implementation Classes: Relationship Access Methods

Corresponding access method?

```
class A1 refines A2
[
  ...
|
  ...
  relationship a3 : A3
  relationship a4 : A4*
  ...
]
```




```
public A3Impl getA3 () {
  return (???)??? (a3) ;
}// getA3
public A4Impl getA4 () {
  return (???)service.getRelationshipsOfAandBOfType ("A1","A4","a4") ;
}// getA4
```


Creation of Asset Implementation Classes: Lifecycle Methods

The life cycle methods (see API documentation) change an asset object's state.

In the case of the I-module for the Broker they access the asset store:

- `store()`: write data to store and upload content
- `lock()`: lock asset and create local buffer
- `commit()`: write data to store, create a new version of the bAsset, and update content
- `abort()`: dispose local buffer
- `delete()`: delete data from store and delete content file

Creation of Asset Implementation Classes: Mapping Class Hierarchies

Example: how to map the following definitions?

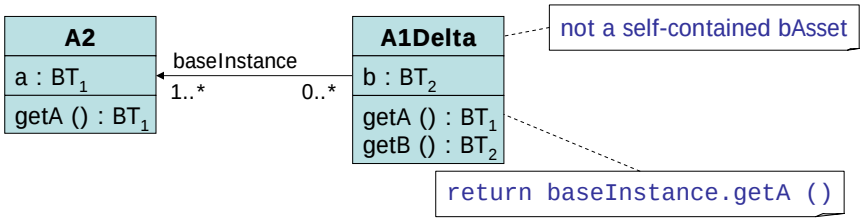
```
class A2 {
  concept characteristic a : T1
}
class A1 refines A2 {
  concept characteristic b : T2
}
```

Problem:
infoAsset Broker does not support subclassing of bAssets.

Creation of Asset Implementation Classes: Mapping Class Hierarchies (1)

Alternative 1:

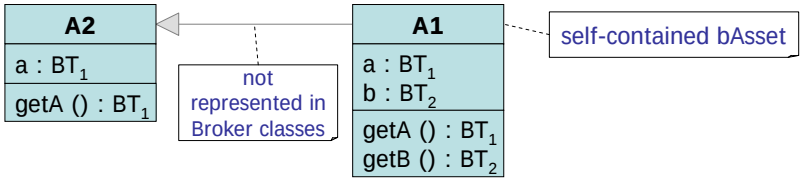
- associated classes
- subclass delegates to base class
- factory creates base class instances (transitive)
- possible with Broker queries?



Creation of Asset Implementation Classes: Mapping Class Hierarchies (2)

Alternative 2:

- full-fledged classes
- asset subclasses mapped to bAsset classes
- queries have to issue Broker queries for all subclasses (transitive)



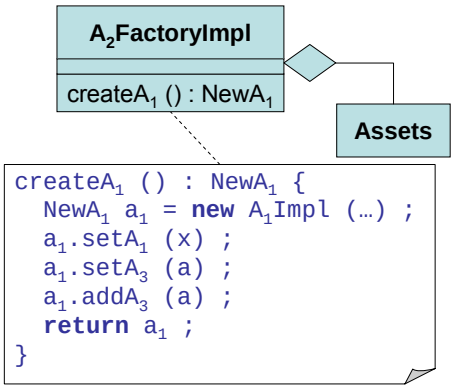
Factory Classes

Factory classes have to be implemented according to the API in order to be able to create bAssets.

According to the API, new instances are volatile.

Factory methods can raise a `VetoException`.

```
class A1 refines A2
[
  ...
|
  characteristic a1 : T1 := x
  characteristic a2 : T2
  relationship a3 : A3* := a
  relationship a4 : A4* := {a}
  ...
]
```



Query Classes

Query classes according to the API allow to query the set of bAssets.

how is this implemented in the Broker???

```
class A1 refines A2
[
  c1 : C1
  c2 : C2
  ...
|
  characteristic a1 : T1
  characteristic a2 : T2
  relationship a3 : A3
  relationship a4 : A4*
  ...
  constraint b
    onviolation s
]
```