Tipuri de vederi şi vederi

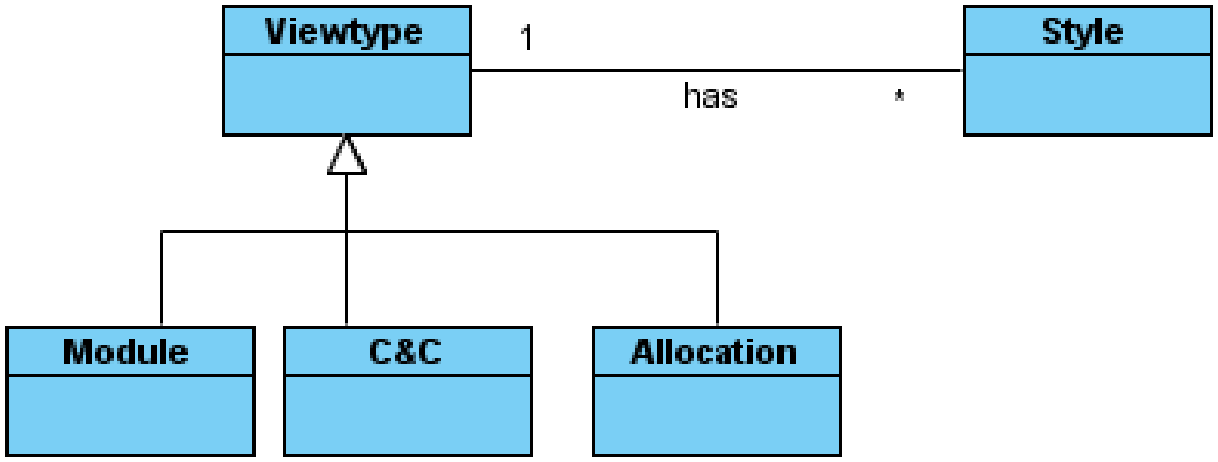
Arhitectul software consideră sistemul din trei perspective:

Perspectiva statică – modul în care sistemul este structurat ca set de ***unităţi de implementare (cod)***.
Vederile reprezintă ***modulele*** sistemului.
Tipul de vedere este ***Module Viewtype***.

Perspectiva dinamică – modul în care sistemul este structurat ca set de ***unităţi de execuţie*** (elemente ce au ***comportament şi interacţiuni la momentul execuţiei***).
Vederile reprezintă ***componentele*** sistemului ***şi conectorii*** dintre acestea.
Tipul de vedere este ***C&C Viewtype***.

Perspectiva alocării – modul în care sistemul se relaţionează cu ***structurile non-software*** din contextul său.
Vederile reprezintă modul de ***alocare*** a structurilor software pe structurile non-software ale sistemului.
Tipul de vedere este ***Allocation Viewtype***.

Concluzie



- Sistemele – combinaţii de stiluri

Ex. Tiers-repository-events

- Este necesară înţelegerea stilurilor pure, ce constituie blocurile constructive
- Regulă : nu amestecaţi stiluri din tipuri diferite de vederi
  - Crează confuzie (Ex: layers şi tiers)

Limbajul Acme şi instrumentul AcmeStudio

Perspectiva dinamică

- Descompunerea (generală) sistemului în componente ce interacţionează la execuţie
- Utilizare abstractizări globale pentru conectare componente
- Analiza proprietăţilor emergente ale sistemului
  - performanţă, rată de transfer, întârzieri, ...
  - fiabilitate, securitate, toleranţă la defecte, modificabilitate dinamică, ...

Perspectiva dinamică

- Vedere C&C include:
  - ***Componente***: definesc locaţiile de realizare a calculelor
    - Exemple: filtre, baze de date, obiecte, tipuri de date abstracte.
  - ***Conectori***: definesc interacţiunile dintre componente
    - Exemple: apel de procedură, conductă (pipe), anunţare eveniment.

- Stil arhitectural C&C - defineşte o familie de arhitecturi prin:
  - Tipuri de componente şi conectori (vocabular)
  - Constrângeri topologice
  - Constrângeri semantice

Descriere arhitecturi

Exemple de limbaje (ADLs)

- Rapide: evenimente cu simulare şi animaţie
- UniCon: accent pe eterogenitate şi compilare
- Wright: spacificaţii formale pentru conectori
- Aesop/Acme: orientate pe stiluri (style-specific)
- Darwin: arhitecturi orientate pe servicii
- SADL: rafinare arhitecturală
- Meta-H: descrieri arhitecturale specifice unui domeniu (avionics)
- C-2: stil arhitectural ce utilizează invocare implicită

Descriere arhitecturi

Acme – limbaj reprezentativ

- Încearcă să ofere o abordare deschisă (open-ended) pentru reprezentarea arhitecturii

- "XML pentru arhitectură"
  - Descriere structură + permite adnotări
  - Poate fi consumat selectiv de diferite instrumente

Descriere arhitecturi

Acme – ENTITĂŢILE DE PRIM RANG

Componentă – element computaţional

Port – punct de interfaţă pentru componentă

Conector – interacţiune între componente

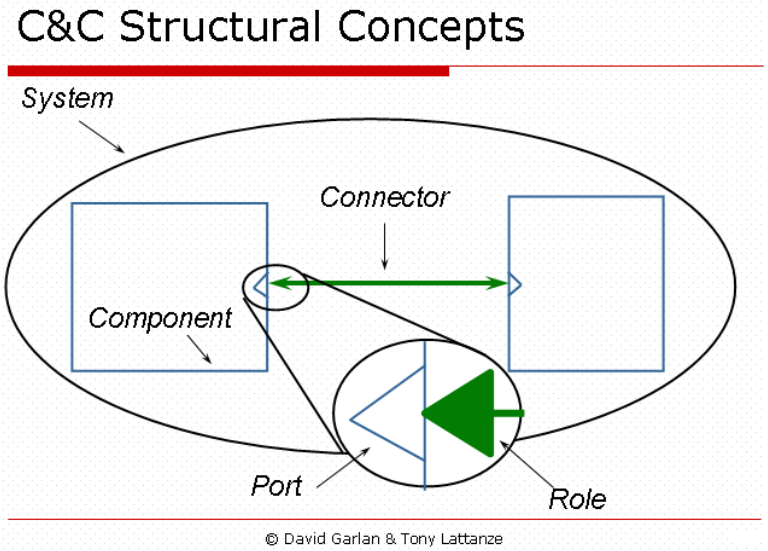
```
System simple-cs = {

  Component client = { port call-rpc; };
  Component server = { port rpc-request; };

  Connector rpc = {
     role client-side;
     role server-side;
  };

  Attachments = {
     client.call-rpc to rpc.client-side;
     server.rpc-request to rpc.server-side;
  }
}
```

Exemplu – sistem client-server



Rol – punct de interfaţă pentru conector

Sistem – graf de componente şi conectori

Descriere arhitecturi

Acme – PROPRIETĂŢI

Descriu caracteristici ***nestructurale*** ale elementelor

- Detalii de *interfaţă* (ex. servicii solicitate, servicii oferite)
- *Proprietăţi locale* ale componentelor sau conectorilor (ex. viteze, capacităţi, întârzieri)
- *Atribute de calitate* (proprietăţi emergente la nivelul sistemului) (ex. performanţă, fiabilitate, securitate)
- *Comportament* (ex. calcule executate de componente, protocoale pentru conectori)
- *Constrângeri* (ex. restricţii topologice, restricţii asupra proprietăţilor)

Descriere arhitecturi

Acme – PROPRIETĂŢI

Reprezentare proprietăţi – adnotări cu perechi atribut-valoare

Proprietăţile sunt ***tipate*** – categorii de tipuri:

- Built-in : int, boolean, string, etc
- Constructori : set, record, enumeration, sequence
- Definite de utilizator

```
System simple-cs = {
  …
  Component server = {
     port rpc-request = {
       Property sync-requests : boolean = true;
     };
    Property max-transactions-per-sec : int = 5;
    Property max-clients-supported : int = 100;
  };

  Connector rpc = { …
    Property protocol : string = "aix-rpc";
  };
…
};
```

Descriere arhitecturi

Acme – REPREZENTĂRI şi ABSTRACTION MAPS

Suport pentru descriere ierarhică a nivelelor de abstractizare.

- Reprezentarea sub-arhitecturilor.
- Descrieri de detaliu ale componentelor şi conectorilor
- Încapsularea detaliilor de nivel inferior

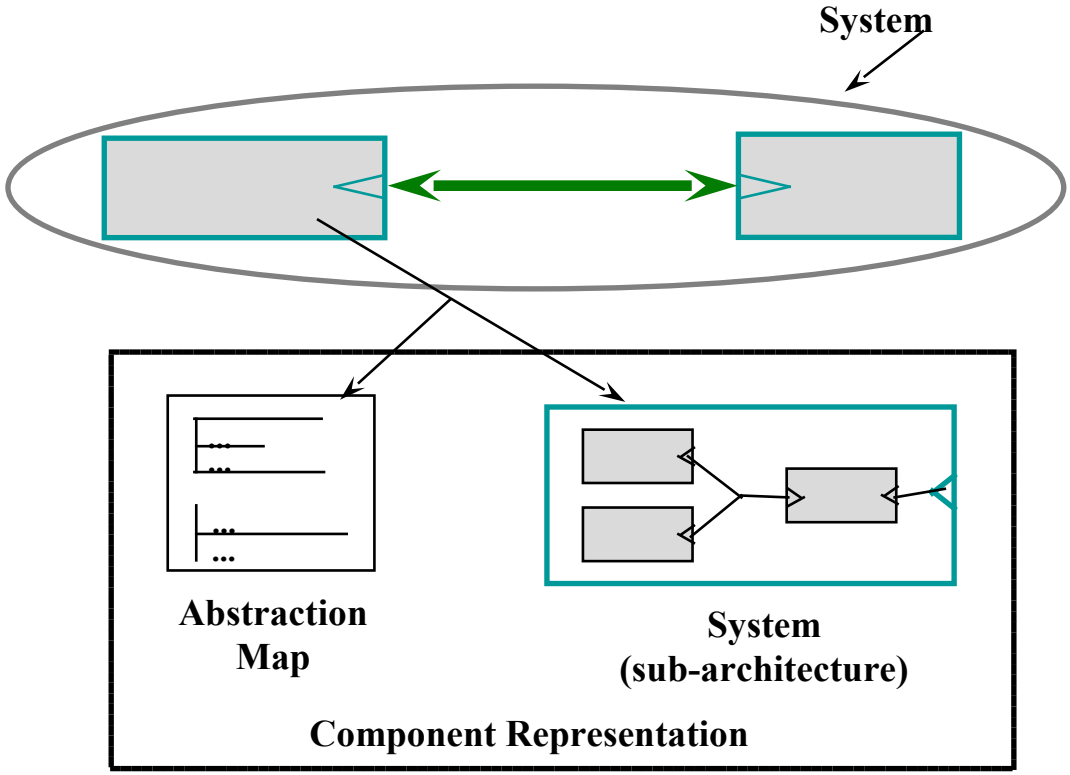
Descriere arhitecturi

Acme – REPREZENTĂRI şi ABSTRACTION MAPS

Unui element Acme i se pot asocia mai multe reprezentări.

Abstraction Map – specificare legături

- interiorul - exteriorul reprezentării
- port intern - port extern (la componente)



Descriere arhitecturi

Acme – DEFINIRE FAMILII (STILURI ARHITECTURALE)

Obiectiv : descrierea unui set de arhitecturi înrudite.

De ce?

- Reutilizarea stilurilor comune
- Suport pentru integritatea sistemului

Elemente constitutive:

- Set de ***tipuri*** (componente, conectori, ...)
  - Definirea vocabularului asociat stilului
  - Definirea structurii cerută fiecărui element
- Set de ***constrângeri***
  - Restricţii de utilizare corectă a tipurilor
- Set de ***vizualizări***
  - Modul de afişare a elementelor stilului
- Set de ***analize***
  - Ce se poate deduce despre o anumită arhitectură

Descriere arhitecturi

Acme – DEFINIRE FAMILII (STILURI ARHITECTURALE)

Definire ***tip element***

```
<categorie> Type <nume-tip> = <declaraţie-tip>
```

Exemplu

```
Component Type filterT = { Ports {in,out} };
```

Definire ***subtip element***

```
<categorie> Type <nume-tip> = <nume-tip>
                    extended with <declaraţie-tip>
```

Exemplu

```
Component Type unixFilterT = filterT
    extended with{ Port error;
     Property throughput : int; };
```

Descriere arhitecturi

Acme – DEFINIRE FAMILII (STILURI ARHITECTURALE)

Definire ***tip proprietate*** – se pot utiliza tipuri predefinite şi tipuri compuse

```
Property Type <nume-tip> = <declaraţie-tip>
```

Exemple

```
Property Type Wright-specT = string;

Property Type pointT = record [x:int; y:int];

Property Type messagesT = seq <string>;
```

Exemplu de utilizare tip de proprietate :

```
Property msgs : messagesT = <"start", "stop">;
```

Descriere arhitecturi

Acme – DEFINIRE FAMILII (STILURI ARHITECTURALE)

Definiţie ***familie*** (stil arhitectural) = colecţie de definiţii de tip.

Exemplu :

```
Family PipeFilterFam = {
    Component Type filterT = {…}
    Connector Type pipeT = {…}
    …
}
```

Exemplu de utilizare :

```
System pf : PipeFilterFam = {…}
```

Descriere arhitecturi

Acme – CONSTRÂNGERI

- Restricţii de combinare a elementelor
- Definite prin predicate
  - Valori : true/false
  - Variaţie a “First Order Predicate Logic”
    - Extinsă cu primitive pentru arhitectură
    - Modelată conform OCL (de la UML)
  - Se pot ataşa la orice instanţă în Acme, inclusiv la întregul sistem sau la familii.

Acme oferă un *cadru semantic deschis* ce permite punerea în corespondenţă a *aspectelor structurale* ale limbajului cu un *formalism logic* bazat pe relaţii şi constrângeri.

Descriere arhitecturi

Acme – CONSTRÂNGERI

Categorii principale :

- Invariant – trebuie satisfăcut pentru ca sistemul să fie legal.

Ex. Nu se acceptă bucle în graful C&C.

- Euristică – produce avertizări dacă nu e satisfăcută.

Ex. Nu se acceptă mai mult de 5 clienţi pentru un server.

Exemplu:

Cerinţă : Stabilirea unui invariant care să impună ca fiecare client să fie conectat la un server.

Soluţie : ataşarea următorului invariant la familia client-server

```
Invariant
   forall c : ClientT in self.components
                | exists s : ServerT in self.components
                | connected (c,s);
```

Descriere arhitecturi

Acme – DEFINIRE FAMILII (STILURI ARHITECTURALE)

Definiţie ***familie*** (stil arhitectural) = colecţie de definiţii de tip completată cu definiţii de constrângeri.

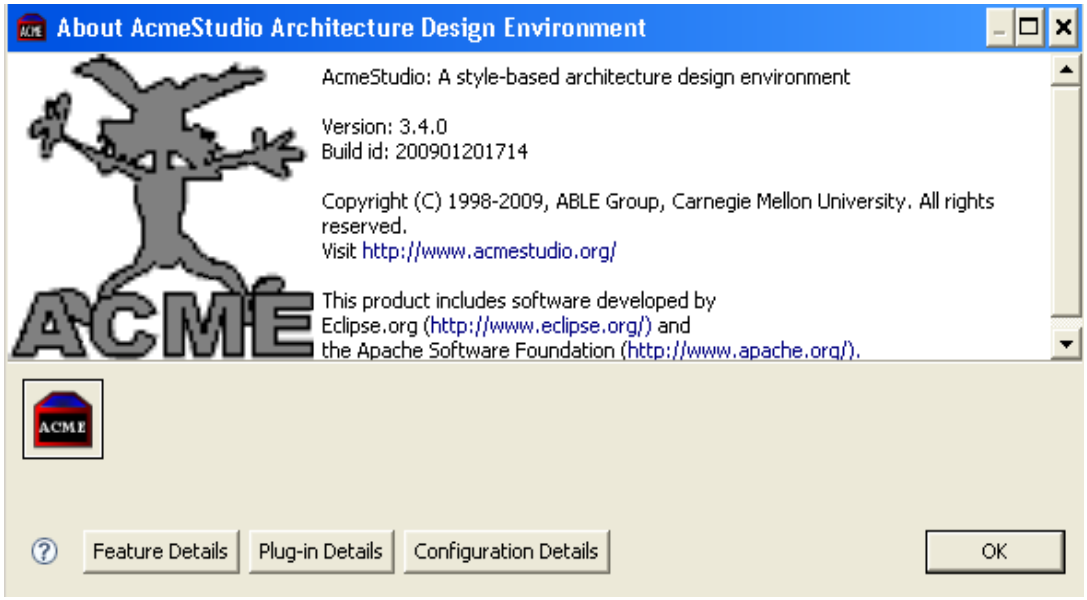
Rule noDanglingPorts is inherited from: PipesAndFiltersFam

Rule	Description

```
forall c : Component in self.COMPONENTS |
  forall p : Port in c.PORTS |
    attachedOrBound(p)
```

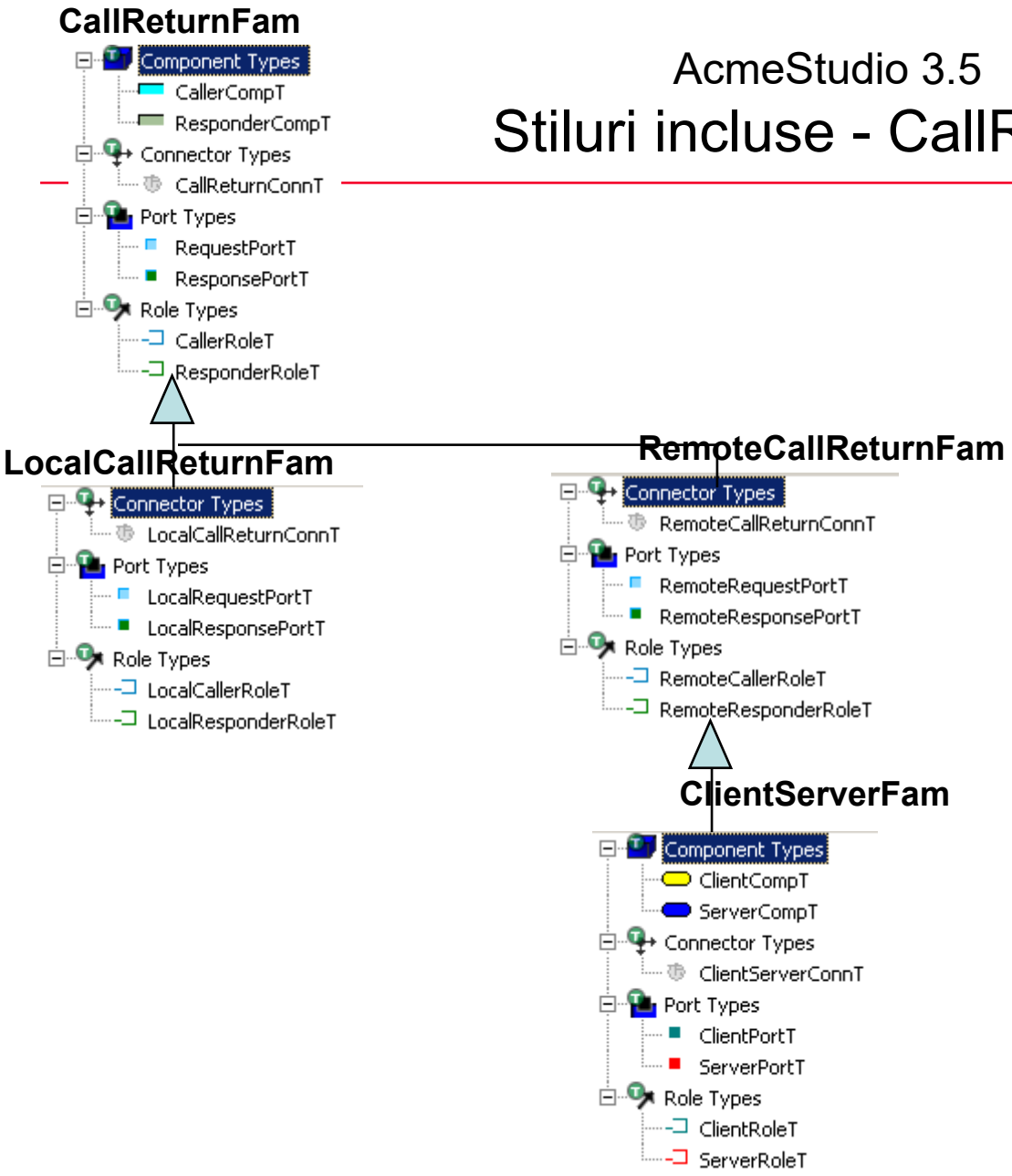
Descriere arhitecturi

AcmeStudio – instrument software construit peste Eclipse.



AcmeStudio 3.5

Stiluri incluse - CallReturn

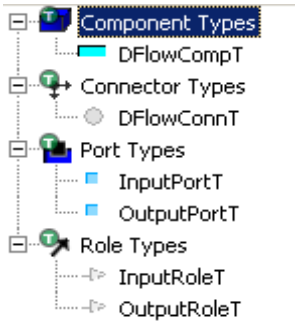


AcmeStudio 3.5

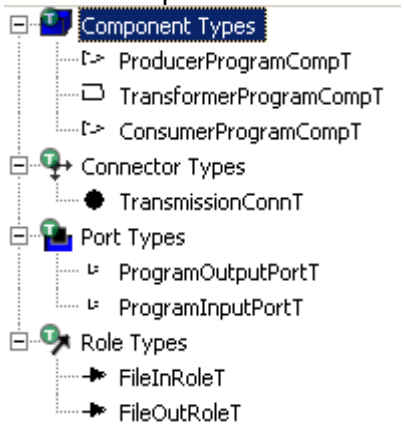
Stiluri incluse - DataFlow

```
import families/DataflowFam.acme;
family PipeAndFilterFam extends DataflowFam with {
    port type FilterInputPortT extends InputPortT with {
            rule dataOutRoles = invariant forall r in
self.ATTACHEDROLES | declaresType(r, DataOutRoleT);};
    port type FilterOutputPortT extends OutputPortT with {
            rule dataInRoles = invariant forall r in self.ATTACHEDROLES
| declaresType(r, DataInRoleT);};
    component type SourceCompT extends DFlowCompT with {
       rule atLeast1portOfTypeFilterOutputPortT = invariant
size(/self/PORTS:!FilterOutputPortT) >= 1;};
    component type SinkCompT extends DFlowCompT with {
       rule atLeast1portOfTypeFilterInputPortT = ...
    component type FilterCompT extends DFlowCompT with {
       rule atLeast1portOfTypeFilterInputPortT = ...
       rule atLeast1portOfTypeFilterOutputPortT = ...
    role type DataInRoleT extends InputRoleT with {
        rule dataInMustBeAttachedToFilterOutput = ...
        rule exactly1attachment = invariant size(self.ATTACHEDPORTS) ==
1 or (size(self.ATTACHEDPORTS) == 0 and attachedOrBound(self));};
    role type DataOutRoleT extends OutputRoleT with {
        rule dataOutMustBeAttached = ...
        rule exactly1attachment = ...
    connector type PipeConnT = {
        rule exactly2roles = invariant size( self.ROLES) == 2;
        rule atLeast1RoleOfTypeDataInRoleT = ...
        rule atLeast1RoleOfTypeDataOutRoleT = ...
}
```

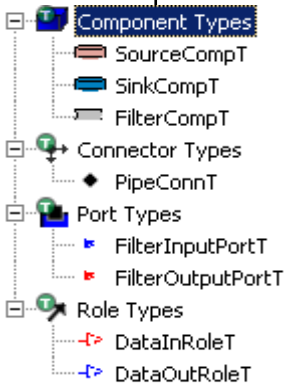
DataFlowFam



BatchSequentialFam

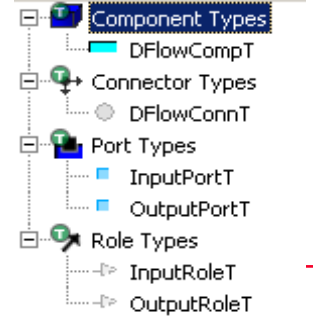


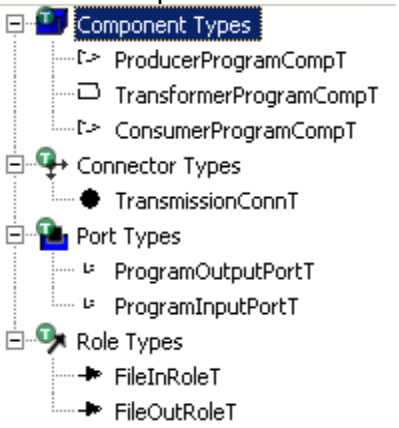
PipeAndFilterFam

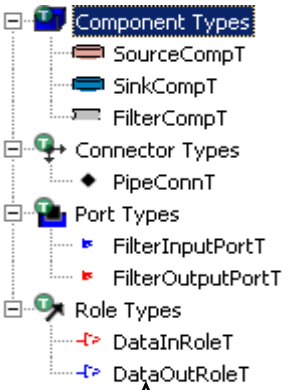


AcmeStudio 3.5

Exemplu : CourseRegistrationFam şi sistemul CourseRegistration








```
import families/CourseRegistrationFam.acme;
import $AS_GLOBAL_PATH/families/PipeAndFilter.acme;

System CourseRegistration : CourseRegistrationFam,
PipeAndFilterFam = new CourseRegistrationFam,
PipeAndFilterFam extended with {
    Component InputFile : DataSourceT = new DataSourceT
extended with {
        Port output : FilterOutputPortT = new
FilterOutputPortT extended with {}}
    Component SplitMSE : SplitFilterT = new SplitFilterT ...
    ...
    Connector PipeT0 : PipeT = new PipeT extended with {
        Role input : DataInRoleT = new DataInRoleT extended
with {}
        Role output : DataOutRoleT = new DataOutRoleT
extended with {}}
    Connector PipeT1
    ...
    Attachment InputFile.output to PipeT0.input;
    Attachment SplitMSE.input to PipeT0.output;
    Attachment SplitMSE.output to PipeT1.input;
    ...}
```

```
import $AS_GLOBAL_PATH/families/PipeAndFilter.acme;
Family CourseRegistrationFam extends PipeAndFilterFam with {
Component Type MergeFilterT extends FilterCompT with {
            Port input2 : FilterInputPortT = new FilterInputPortT extended with {
            Property protocol : string = "char input";}}
Component Type SplitFilterT extends FilterCompT with {
      Port output2 : FilterOutputPortT = new FilterOutputPortT extended with {
            Property protocol : string = "char output";}}
Component Type CourseFilterT extends FilterCompT with {}
Component Type DataSourceT extends SourceCompT with {}
Component Type DataSinkT extends SinkCompT with {}
Connector Type PipeT extends PipeConnT with {}}};
```

Sistemul CourseRegistration

```
import families/CourseRegistrationFam.acme;
import $AS_GLOBAL_PATH/families/PipeAndFilter.acme;
System CourseRegistration : CourseRegistrationFam, PipeAndFilterFam
= new CourseRegistrationFam, PipeAndFilterFam extended with {
    Component InputFile : DataSourceT = new DataSourceT extended with {
        Port output : FilterOutputPortT = new FilterOutputPortT...
    Component SplitMSE : SplitFilterT = new SplitFilterT extended with {
       Port input : FilterInputPortT = new FilterInputPortT ...
       Port output : FilterOutputPortT = new FilterOutputPortT ...}
    Component Course17651 : CourseFilterT = new CourseFilterT extended with {
       Port input ... Port output...}
    Component Course21701 : CourseFilterT = new CourseFilterT extended with {
        Port input... Port output...}
    Component OutputFile : DataSinkT = new DataSinkT extended with {
        Port input : FilterInputPortT = new FilterInputPortT ...}
    Component MergeFilter : MergeFilterT = new MergeFilterT extended with {
        Port input... Port output...}
    Connector PipeT0 : PipeT = new PipeT extended with {
        Role input : DataInRoleT = new DataInRoleT...
        Role output : DataOutRoleT = new DataOutRoleT...}
    Connector PipeT1 ... Connector PipeT2...  Connector PipeT3
    Connector PipeT4 ... Connector PipeT5

    Attachment InputFile.output to PipeT0.input;
    Attachment SplitMSE.input to PipeT0.output;
    Attachment SplitMSE.output to PipeT1.input;
    Attachment Course17651.input to PipeT1.output;
    Attachment Course21701.input to PipeT2.output;
    Attachment SplitMSE.output2 to PipeT2.input;
    Attachment Course17651.output to PipeT3.input;
    Attachment Course21701.output to PipeT4.input;
    Attachment OutputFile.input to PipeT5.output;
    Attachment MergeFilter.input to PipeT3.output;
    Attachment MergeFilter.input2 to PipeT4.output;
    Attachment MergeFilter.output to PipeT5.input;
}
```