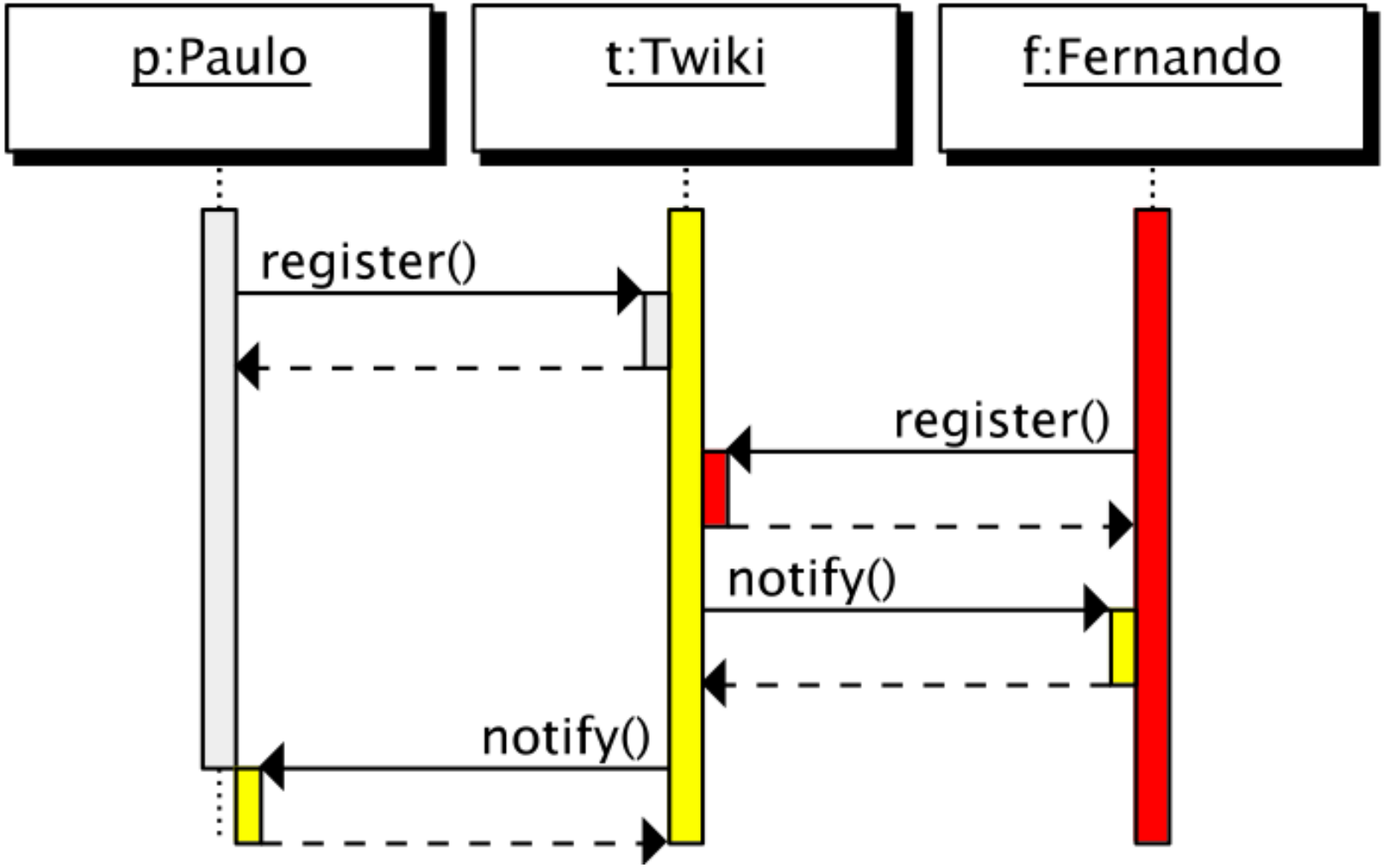
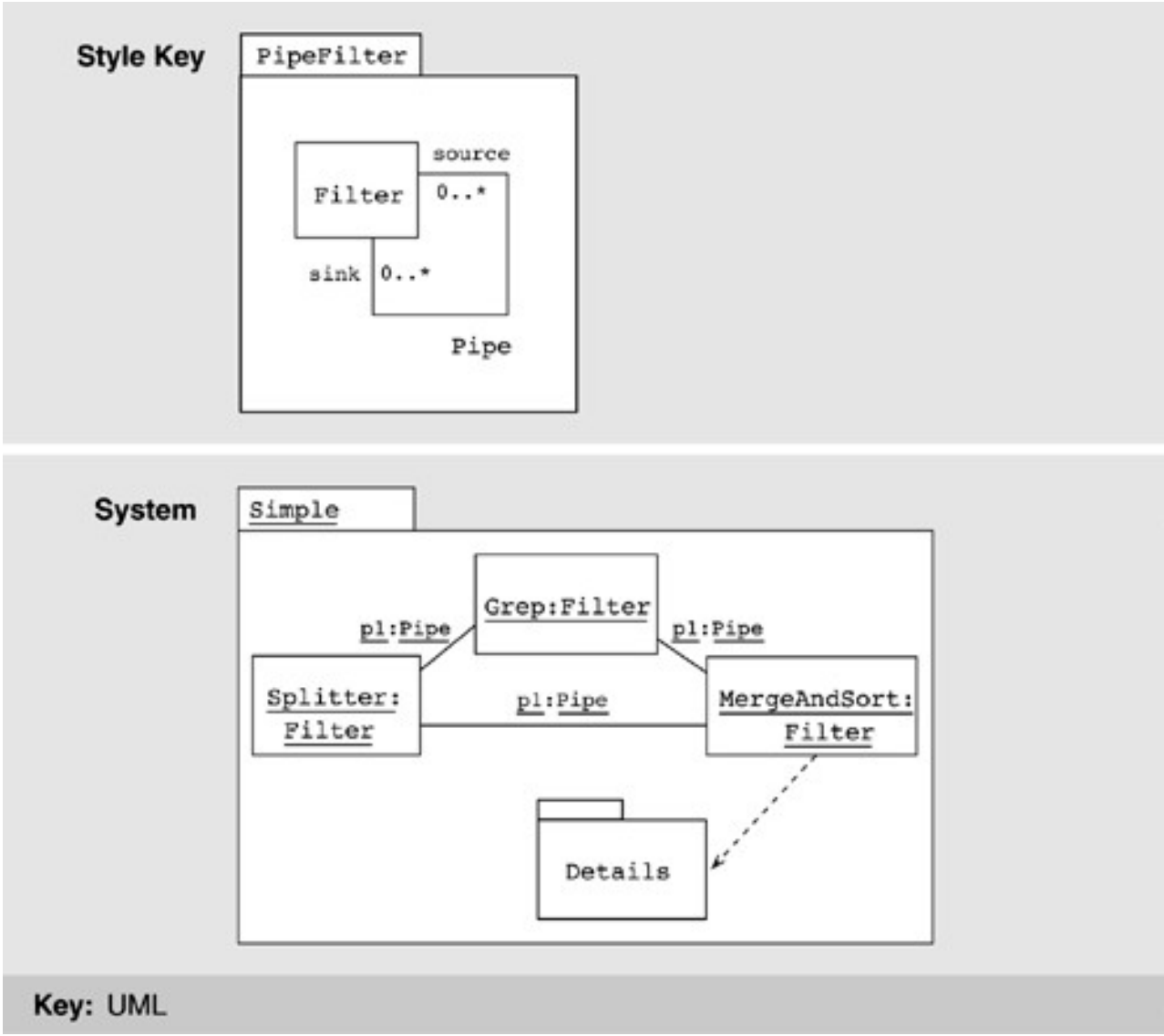
A Linguagem ACME para Descrição de Arquiteturas de Software

Fernando Castor e Paulo Borba

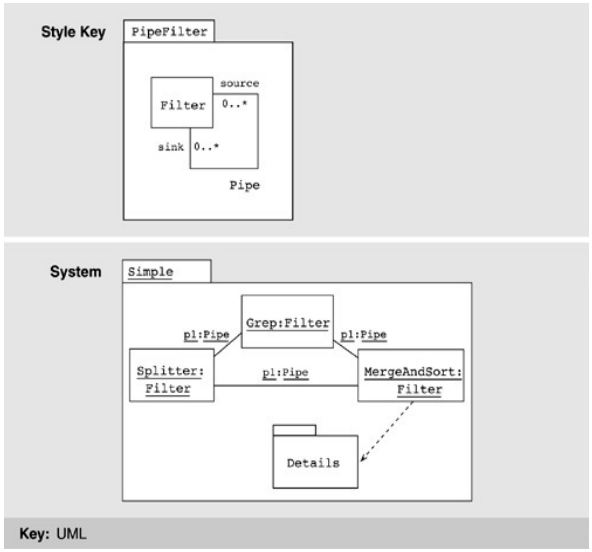


Quais os problemas desta visão arquitetural?

(discutidos na aula anterior)


Style Key
PipeFilter
source
Filter
0..*
sink
0..*
Pipe
System
Simple
Grep:Filter
p1:Pipe
p1:Pipe
Splitter:
Filter
p1:Pipe
MergeAndSort:
Filter
Details
Key: UML

E os destas?



- Estilo definido informalmente
- Nenhuma checagem de aderência
  - Não evita instanciações inválidas
  - Depende da ferramenta
- E se fosse necessário compor vários estilos?
- Portas e conectores ``artificiais''

Problema

- Notações como a UML apresentam diversas limitações para especificar arquiteturas
  - Apesar de terem inúmeras virtudes

Linguagens para Descrição de Arquitetura (ADLs)

- Surgiram para remediar esses problemas
- Auge na década de 90
- Foco na estrutura
  - Enriquecida com informação adicional

Alguns Exemplos de ADLs

- **Wright** → protocolos de interação entre comps.
- **Darwin** → reconfiguração dinâmica
- **Rapide** → prototipação de sistemas orientados a eventos e simulação
- **ACME** → especificação de estilos, checagem de aderência e extensibilidade

Esta aula foca na ADL

ACME

→ Núcleo simples

→ Amparada por ferramentas

→ Usada na academia

→ Talvez na prática também!

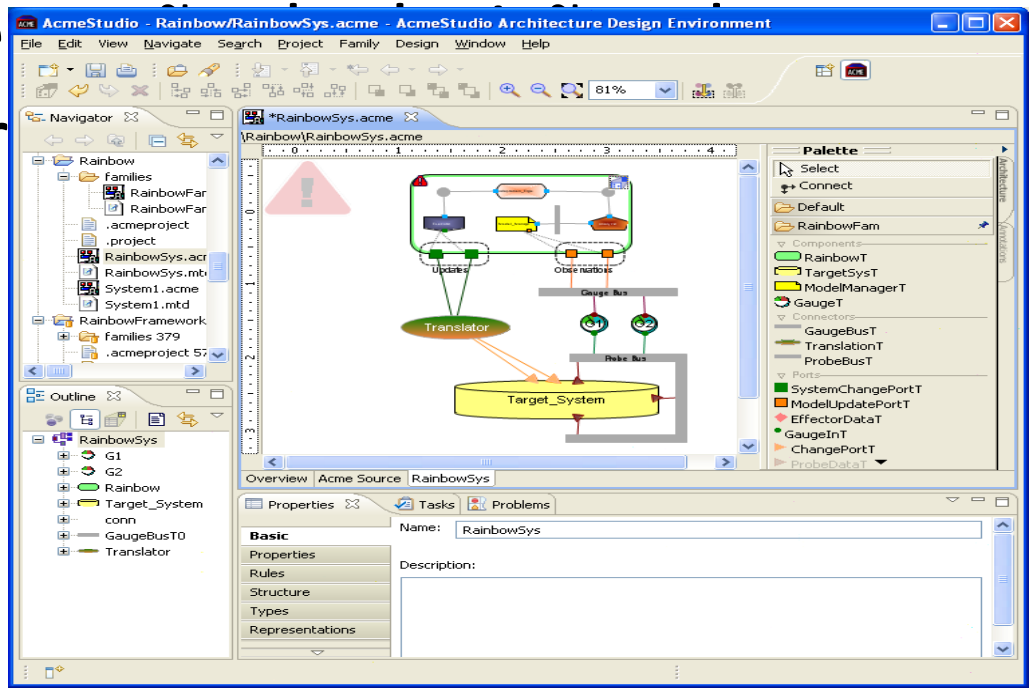
Por que falar sobre ADLS?

(e notações formais, como CSP)

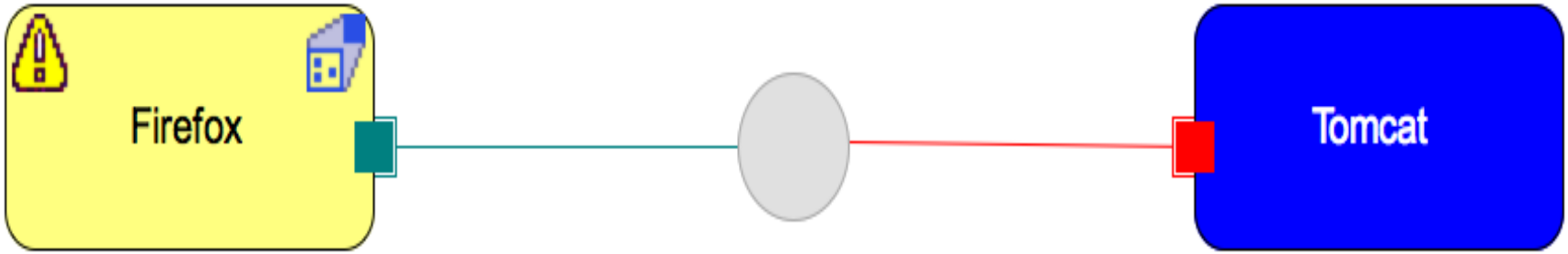
- Complementam descrições em UML
- Mais próximas do ``**ideal**'' de modelagem
  - Vale ressaltar: o ideal é útil na prática!
- Ainda são pouco usadas, mas no futuro...
  - Podem influenciar novas notações
    - Ex. Componentes em UML 2.0
- Existem alguns exemplos práticos: Koala, AADL
  - Ambas foram desenvolvidas na indústria

ACME

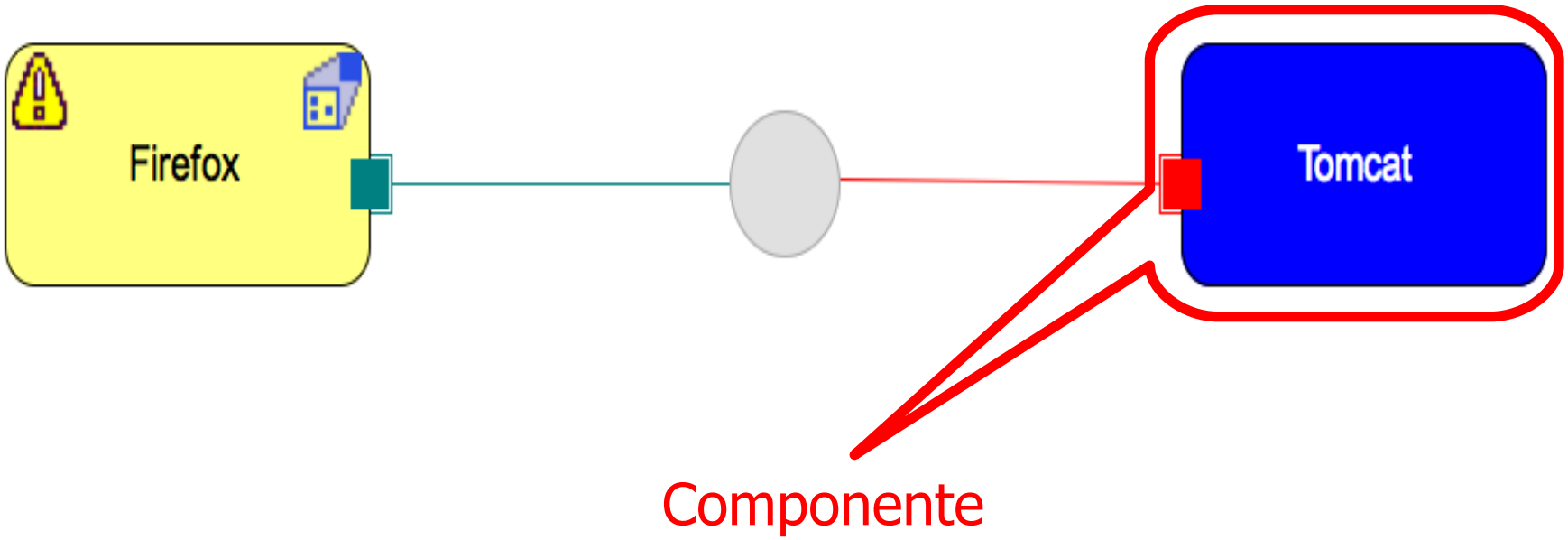
- Projetada como uma linguagem de intercâmbio
- Extensibilidade, através de *Propriedades*
- Foco na represe
  projeto estrutur
- AcmeStudio



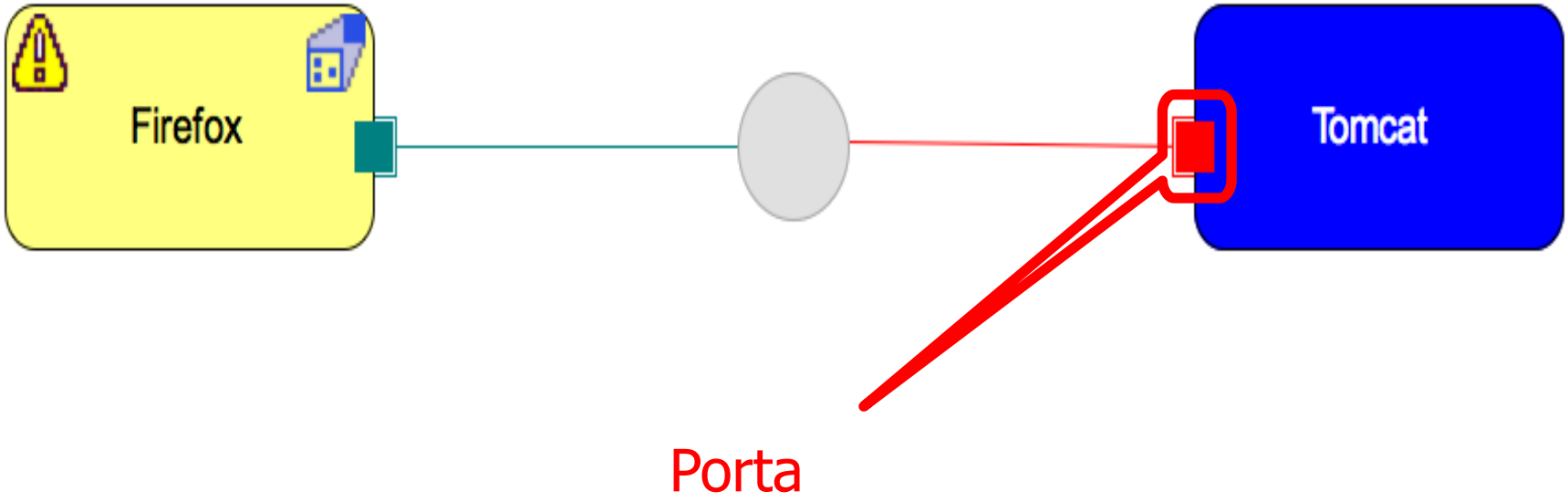
Elementos de ACME



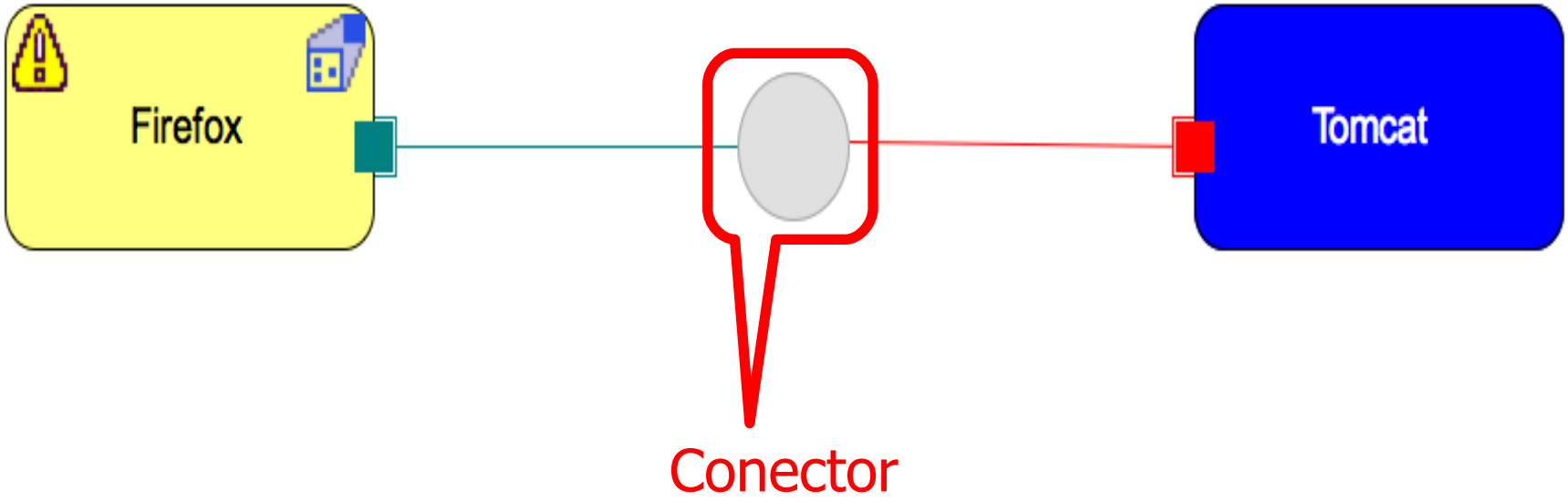
Elementos de ACME



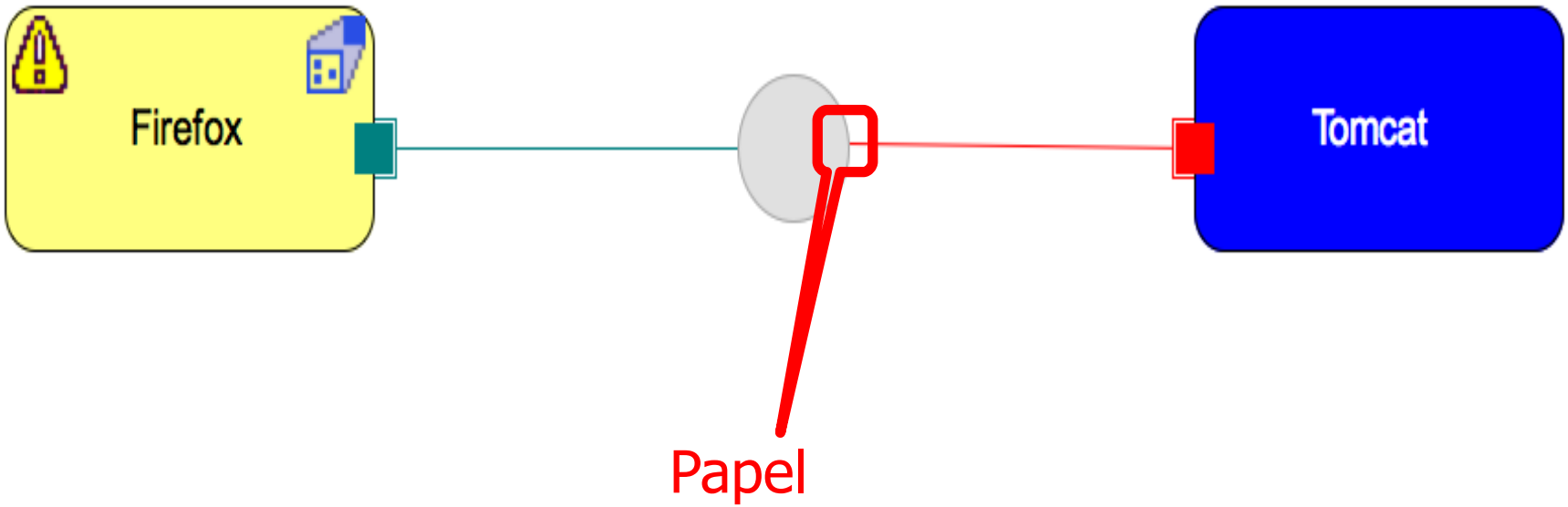
Elementos de ACME



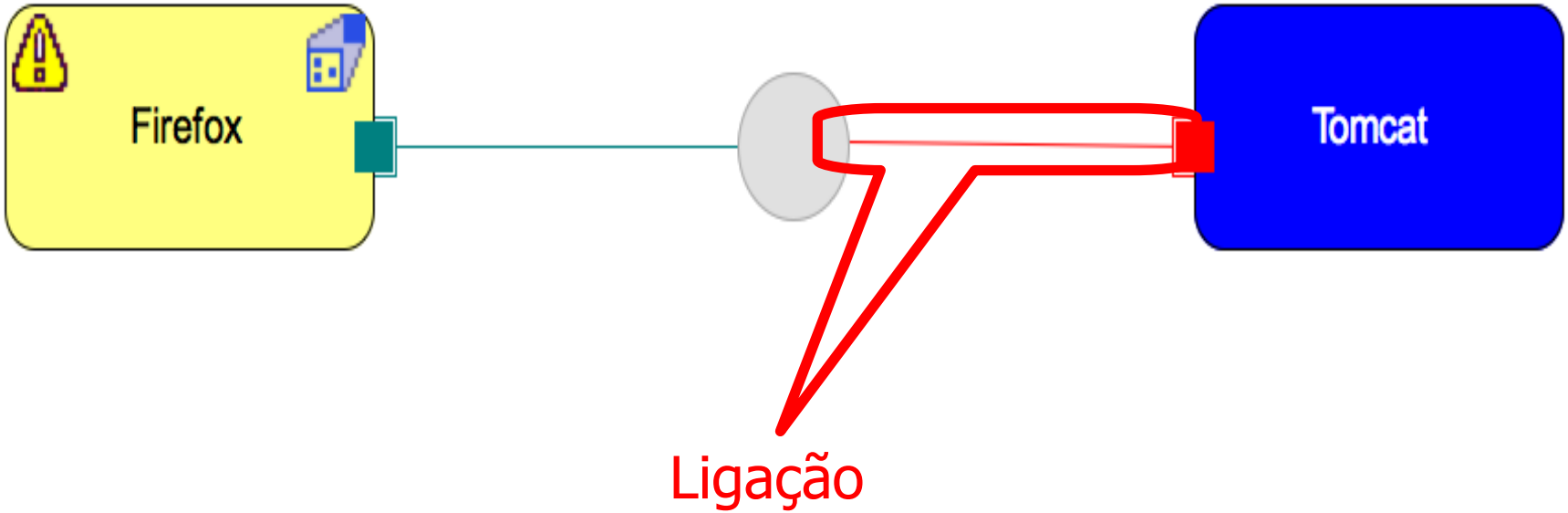
Elementos de ACME



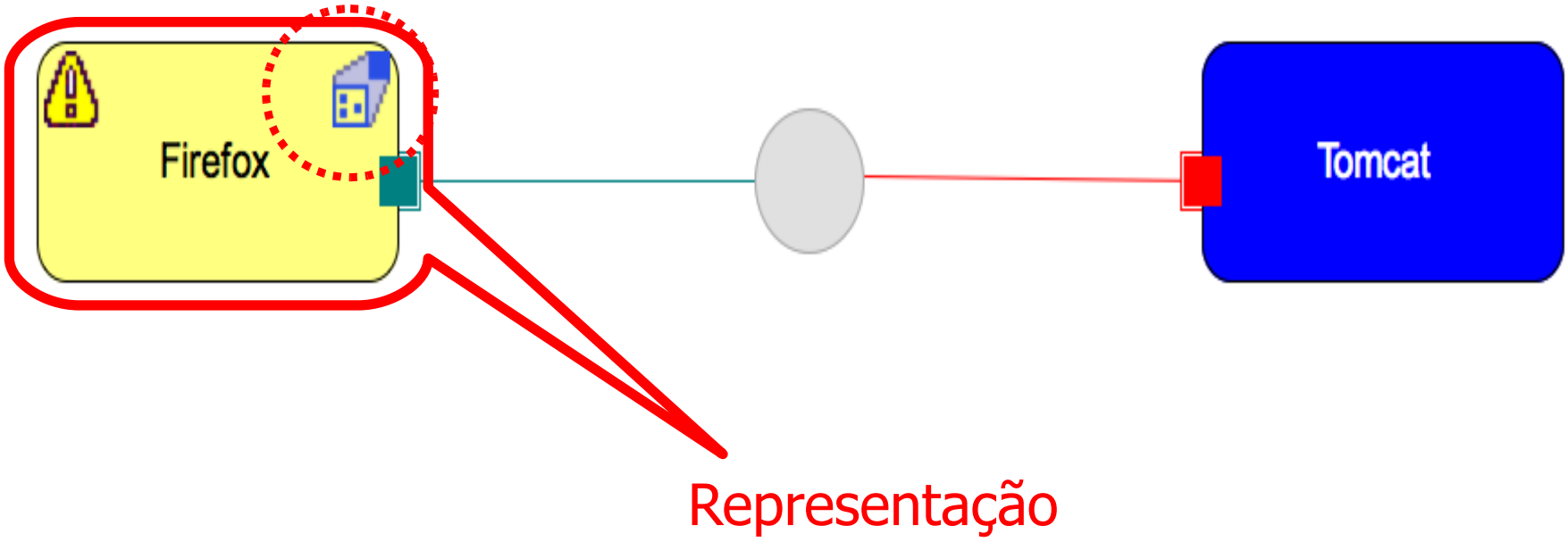
Elementos de ACME



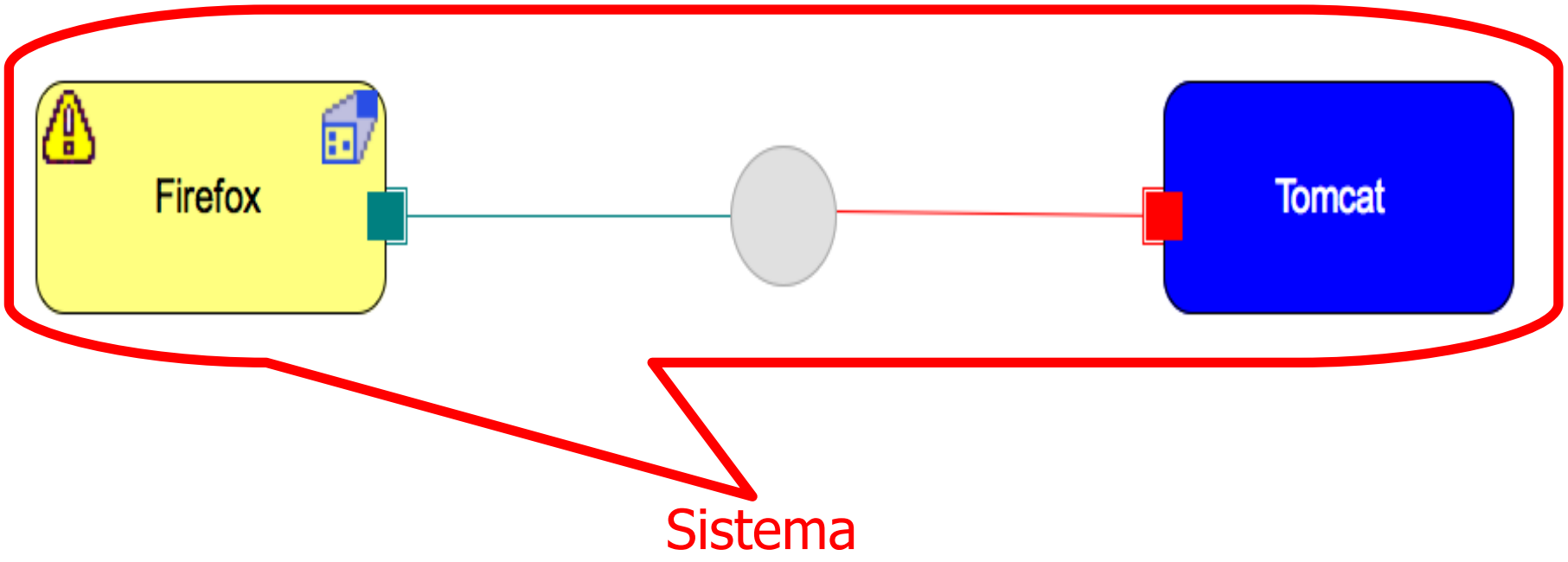
Elementos de ACME



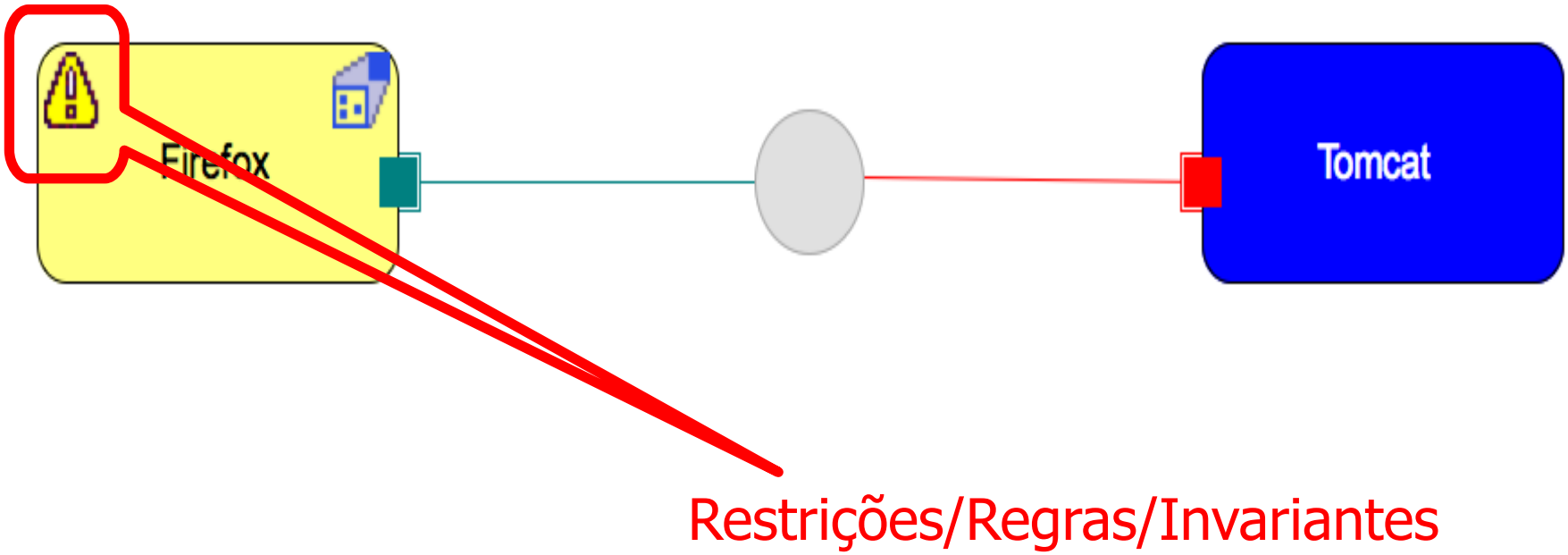
Elementos de ACME



Elementos de ACME



Elementos de ACME



Um Exemplo de Sistema em ACME

```
System exemplo = {
   Component Firefox = {  Port sendRequest; }
   Component Tomcat = {
      Port receiveRequest = { Property persistent : boolean; }
      Property multiThreaded : boolean;
      Property max-concurrent-requests : int;
   }
   Connector HTTP = { Roles = { clientSide, serverSide; } }
   Attachment Tomcat.receiveRequest to HTTP.serverSide;
   Attachment Firefox.sendRequest to HTTP.clientSide;
}
```

Famílias

- Especificam estilos arquiteturais
- Sistemas instanciam 0 ou mais famílias
- Definem diversos tipos de elementos
  - E instâncias de alguns deles
  - Elementos podem ter vários tipos em uma arquitetura

Exemplo Revisitado

(instanciando a família Cliente-Servidor)

```
import $AS_GLOBAL_PATH/families/PubSubFam.acme;
import $AS_GLOBAL_PATH/families/ClientAndServerFam.acme;

System exemplo : ClientAndServerFam = new ClientAndServerFam
    extended with {
   Component Firefox : ClientT = new ClientT extended with {
      Representation Firefox_Rep = { … }
   }
   Component Tomcat : ServerT = new ServerT extended with {}
   Connector HTTP : CSConnT = new CSConnT extended with {}
   Attachment Tomcat.receiveRequest to HTTP.serverSide;
   Attachment Firefox.sendRequest to HTTP.clientSide;
   …
}
```

Família Cliente-Servidor (1/2)

```
Family ClientAndServerFam = {
   Port Type ServerPortT = {
      rule portHasAttachment = invariant size(self.ATTACHEDROLES) >= 1 OR
     (size(self.ATTACHEDROLES) == 0 AND attachedOrBound(self));
   }
   Role Type serverSideRoleT = {
      rule roleHasAttachment = invariant size(self.ATTACHEDPORTS) >= 1 OR
      (size(self.ATTACHEDPORTS) == 0 AND attachedOrBound(self));
   }
   Port Type ClientPortT = { ... };  Role Type clientSideRoleT = { ... }
   ...
```

Família Cliente-Servidor (2/2)

```
  ...
  Component Type ServerT = {
    Port receiveRequest : ServerPortT = new ServerPortT;
    Property multiThreaded : boolean;  Property max-concurrent-requests : int;
  }
  Component Type ClientT = { Port sendRequest : ClientPortT = new ClientPortT; }
  Connector Type CSConnT = {
    Role clientSide : clientSideRoleT = new clientSideRoleT extended with {}
    Role serverSide : serverSideRoleT = new serverSideRoleT extended with {}
    Property blocking : boolean;
    Property protocol : string;
    rule hasTwoRoles = invariant size(self.ROLES) == 2;
  }
}
```

Representações

- Decomposição hierárquica da arquitetura
  - Modelam a arquitetura interna dos elementos
  - Funciona em vários níveis
- Aplicam-se a componentes e conectores
- Favorecem a modelagem incremental

Representações

```
Component Firefox : ClientT = new ClientT extended with {
  Representation Firefox_Rep = {
    System Firefox_Rep : ClientAndServerFam, PubSubFam =
        new ClientAndServerFam, PubSubFam extended with {
      Component JS_Engine :ParticipantT = new ParticipantT extended with {
        Port pjs : p_announce = new p_announce extended with {}}
      Component Communication : ParticipantT, ClientT = new ParticipantT,
        ClientT extended with { … }
      Component GUI : ...
      Connector EventBusT0 : EventBusT = new EventBusT extended with {
        Role rp1 : r_publisher = new r_publisher extended with {}  … }
      Attachments { GUI.p to EventBusT0.rp1; JS_Engine.pjs to EventBusT0.rp3;
        Communication.pc to EventBusT0.rp2; }
    } Bindings { Communication.sendRequest to Firefox.sendRequest; }
  }
}
```

Propriedades

- Mecanismo básico de extensão
- Acrescentam semântica a sistemas e famílias
  - Para ACME, são valores não-interpretados
  - Ferramentas adicionais são responsáveis por processá-las
- Estrutura simplificada:

  `Property' <nome> ':' <tipo> [ `=' <valor>] `;'

Tipos de Propriedades

- Tipos primitivos:
  - **Int**, **Long**, **Float**, **Boolean**, **String**
- Coleções
  - **Set**, **Sequence**
- Dados estruturados
  - **Record**
- Novos tipos podem ser definidos

Regras

- Expressas em uma linguagem específica
  - Lógica de predicados de primeira ordem
- Aplicáveis em vários contextos
- Avaliadas automaticamente pelo AcmeStudio
- Operadores e ``API'' de funções pré-definidos
- Exemplo:

```
rule portHasAttachment = invariant size(self.ATTACHEDROLES) >= 1 OR
  (size(self.ATTACHEDROLES) == 0 AND attachedOrBound(self));
```

Algumas Funções

- Connected (comp1, comp2)
- Reachable (comp1, comp2)
- HasProperty (elt, propName)
- HasType (elt, typeName)
- SystemName.Connectors
- ConnectorName.Roles

Um Exemplo mais Complexo

```
rule onlyOneBus = heuristic forall comp :
   Component in self.COMPONENTS |
      forall p : p_announce in comp.PORTS |
         ( ! (exists r1 : Role in p.ATTACHEDROLES |
            exists r2 : Role in p.ATTACHEDROLES |  r1 != r2 AND
               (exists conn : Connector in self.CONNECTORS |
                  isSubset({r1, r2}, conn.ROLES))));
```

Obrigado!

Dúvidas e comentários:

castor@cin.ufpe.br

phmb@cin.ufpe.br