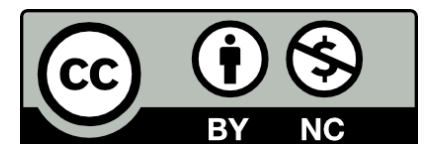


Inconsistency Detection in Distributed Model Driven Software Engineering Environments

The single-view inconsistency detection method



Presented by Alix Mougenot



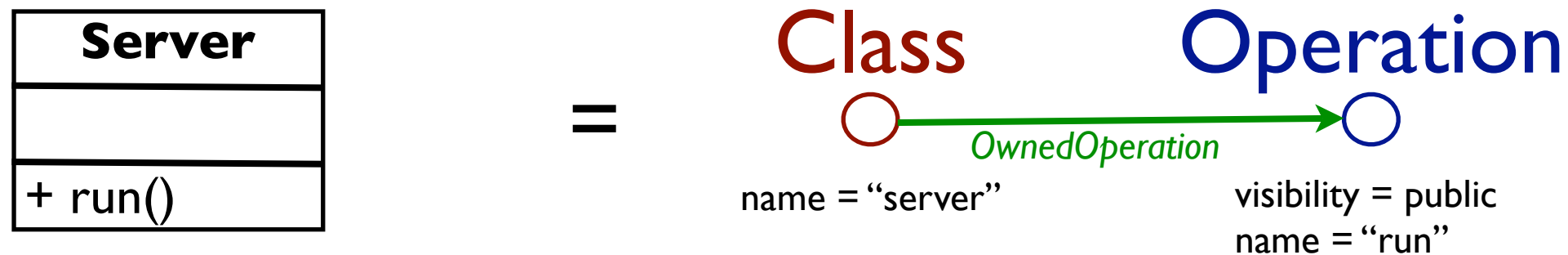
RoadMap

- Context definition
- Problems regarding global inconsistency detection
- Single-view inconsistency detection

Context

Inconsistency detection within Models:

- **Model**: Attributed Graph of model elements

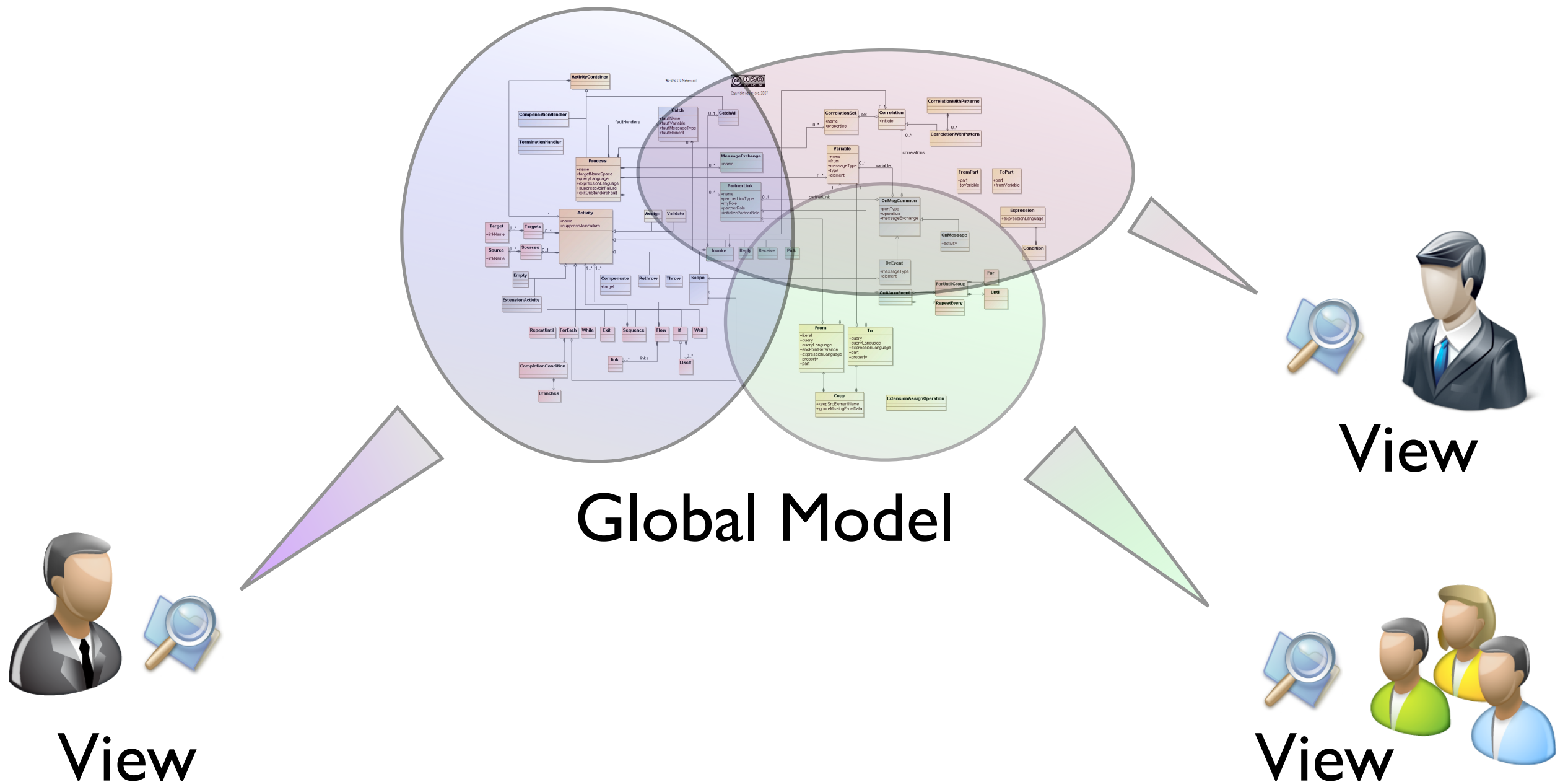


- **Inconsistency**: Set of model elements that matches an inconsistency rule.
Couple (model elements, inconsistency rule)

Context

Multiview model editing

- **View:** Model that is a sub-model of the **Global Model**



Problem

Today: Inconsistencies are detected on the global model.

Problem 1: That does not respect the separation of concerns principle.

Problem 2: The Global Model may not be available.

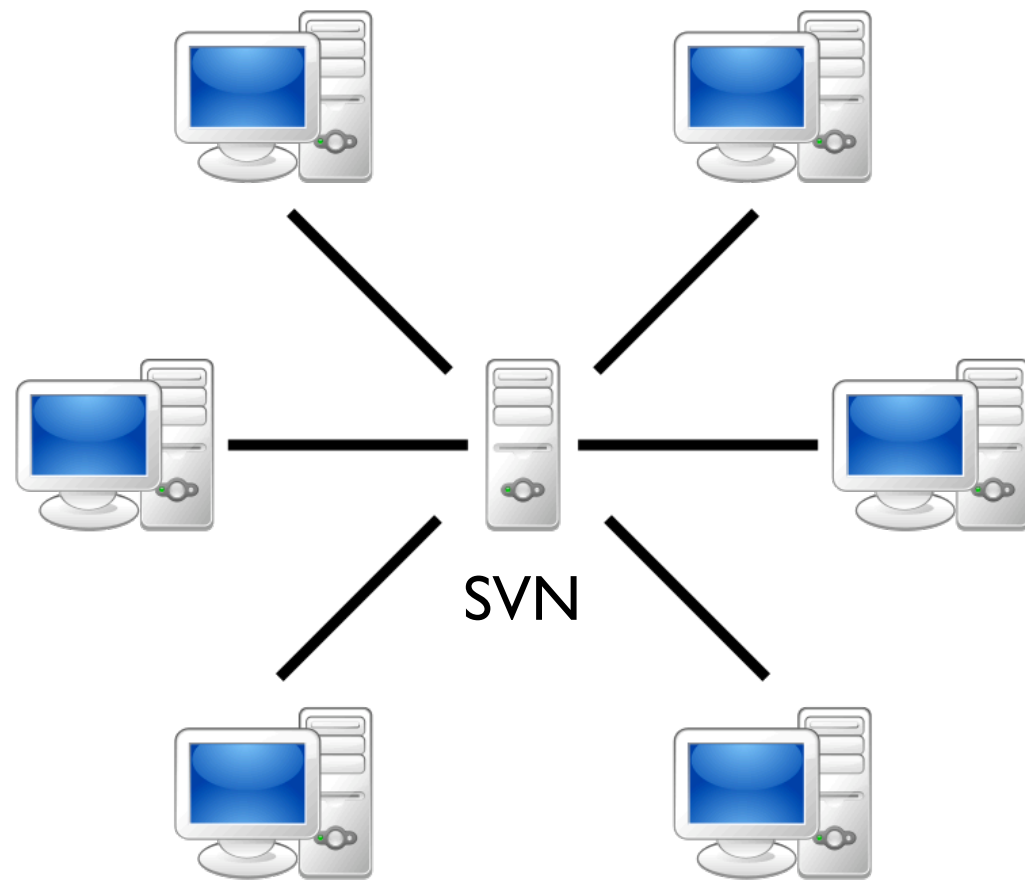
Problem I

Respecting the separation of concerns

- Global inconsistency detection can be confusing:
Why would a user from a particular view be interested in inconsistencies he/she can't resolve?
- Global inconsistency detection can be wasteful:
Global consistency is harder to compute than one view plus it's overlapping elements.

Problem 2

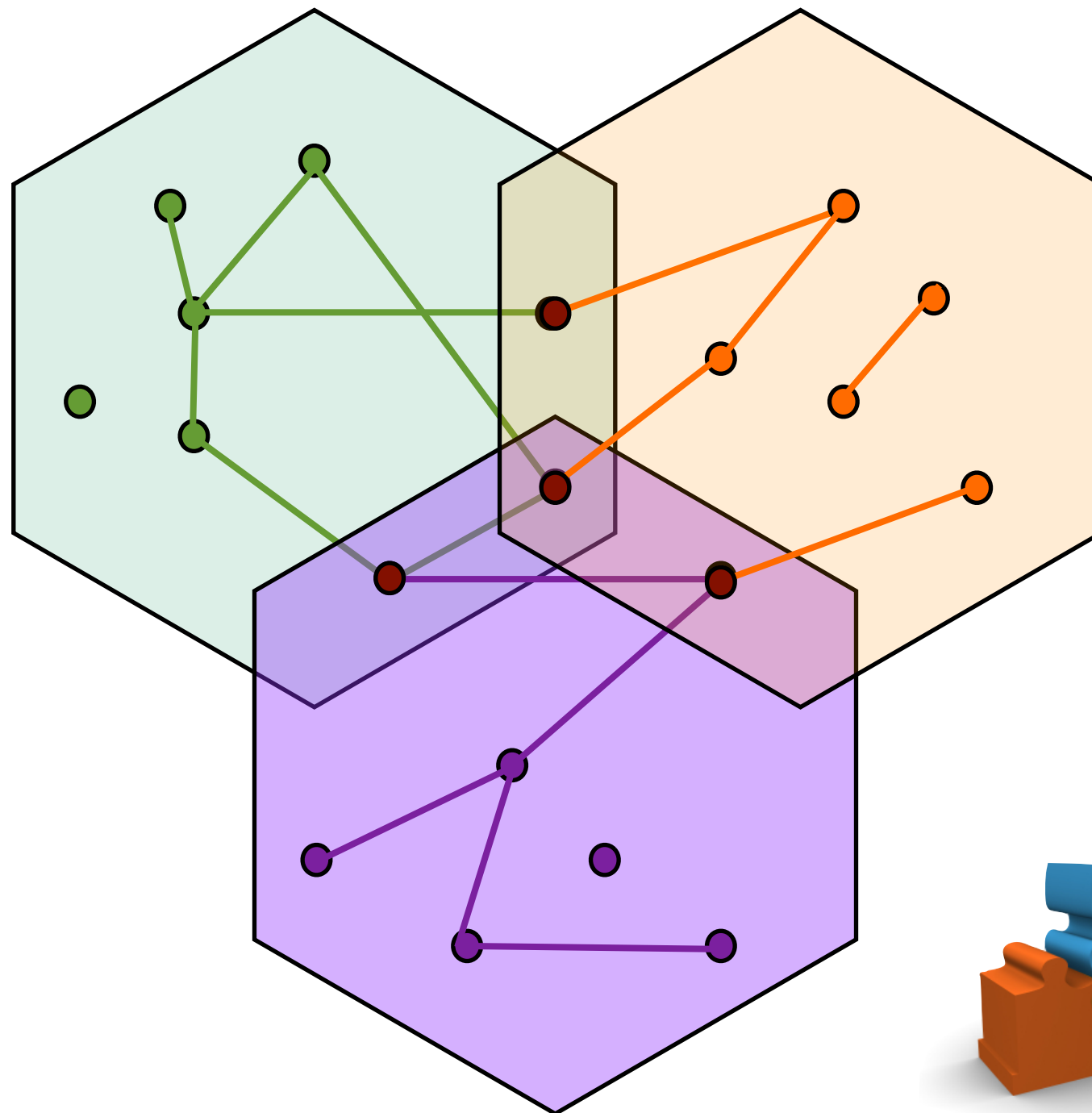
Distributed Views: Clarification



Versioning systems are used for **Collaborative editing** not so much for **Distributed editing**.

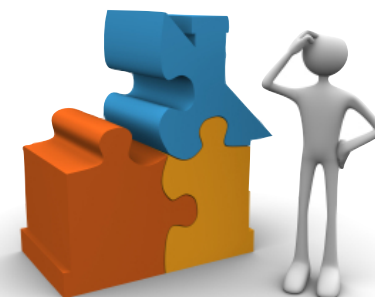
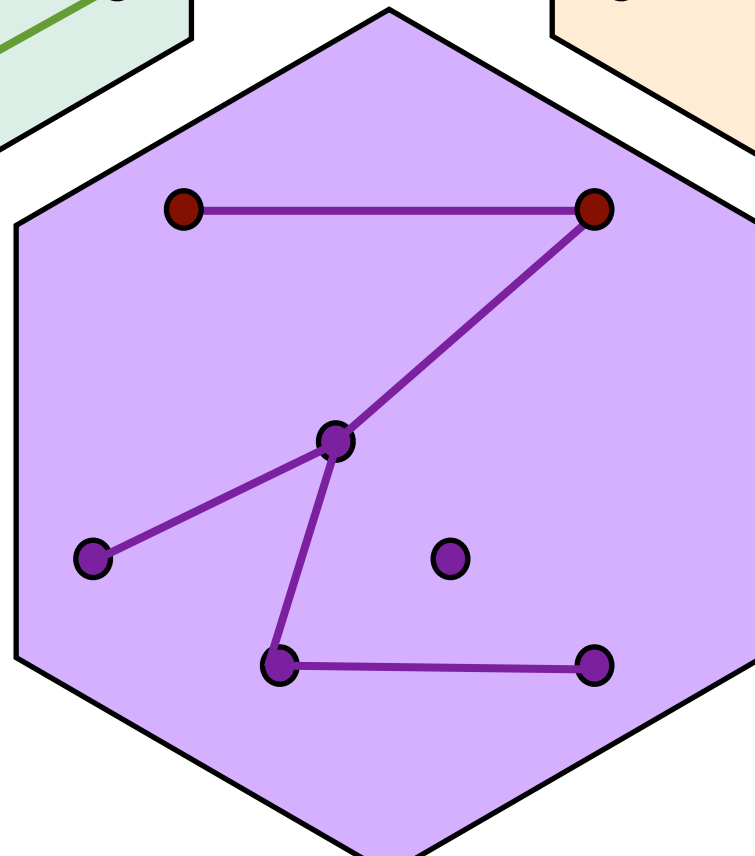
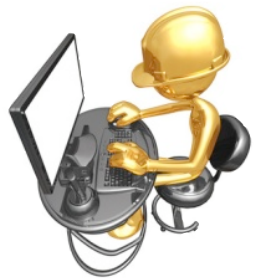
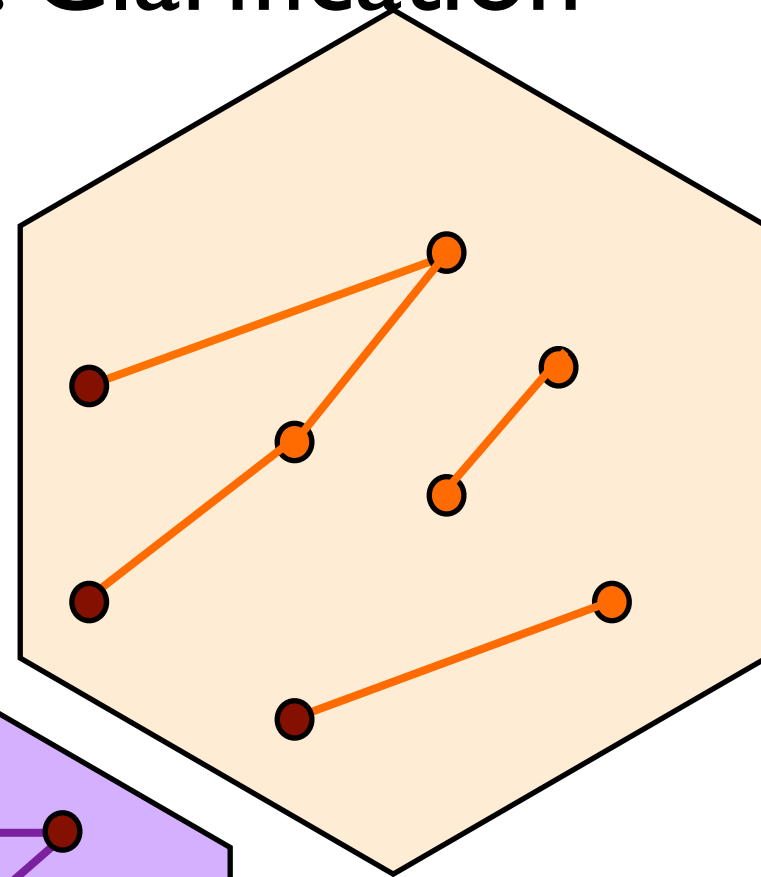
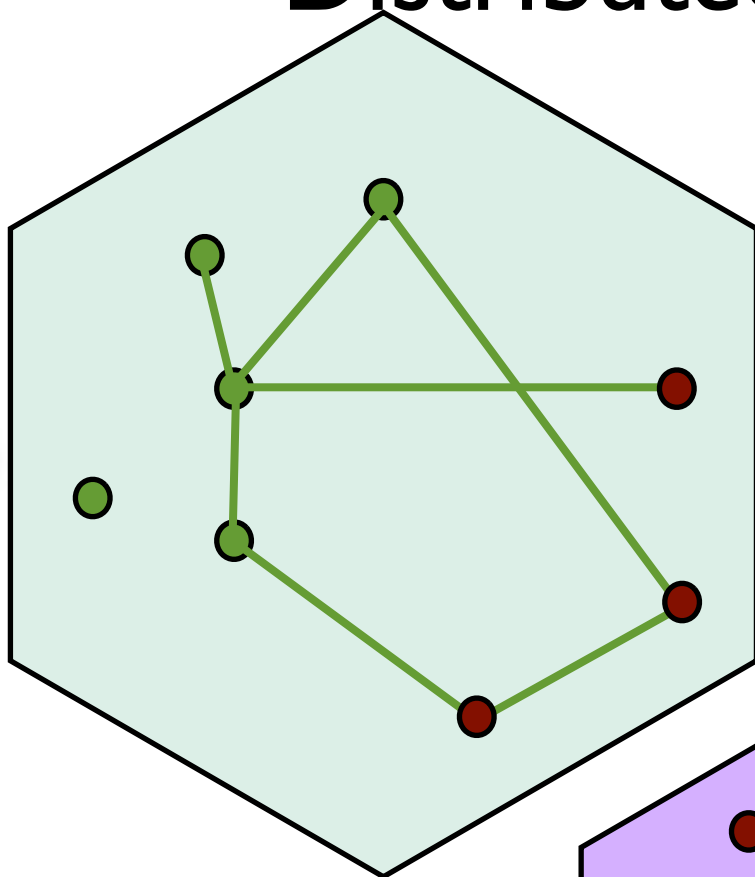
Problem 2

Distributed Views: Clarification



Problem 2

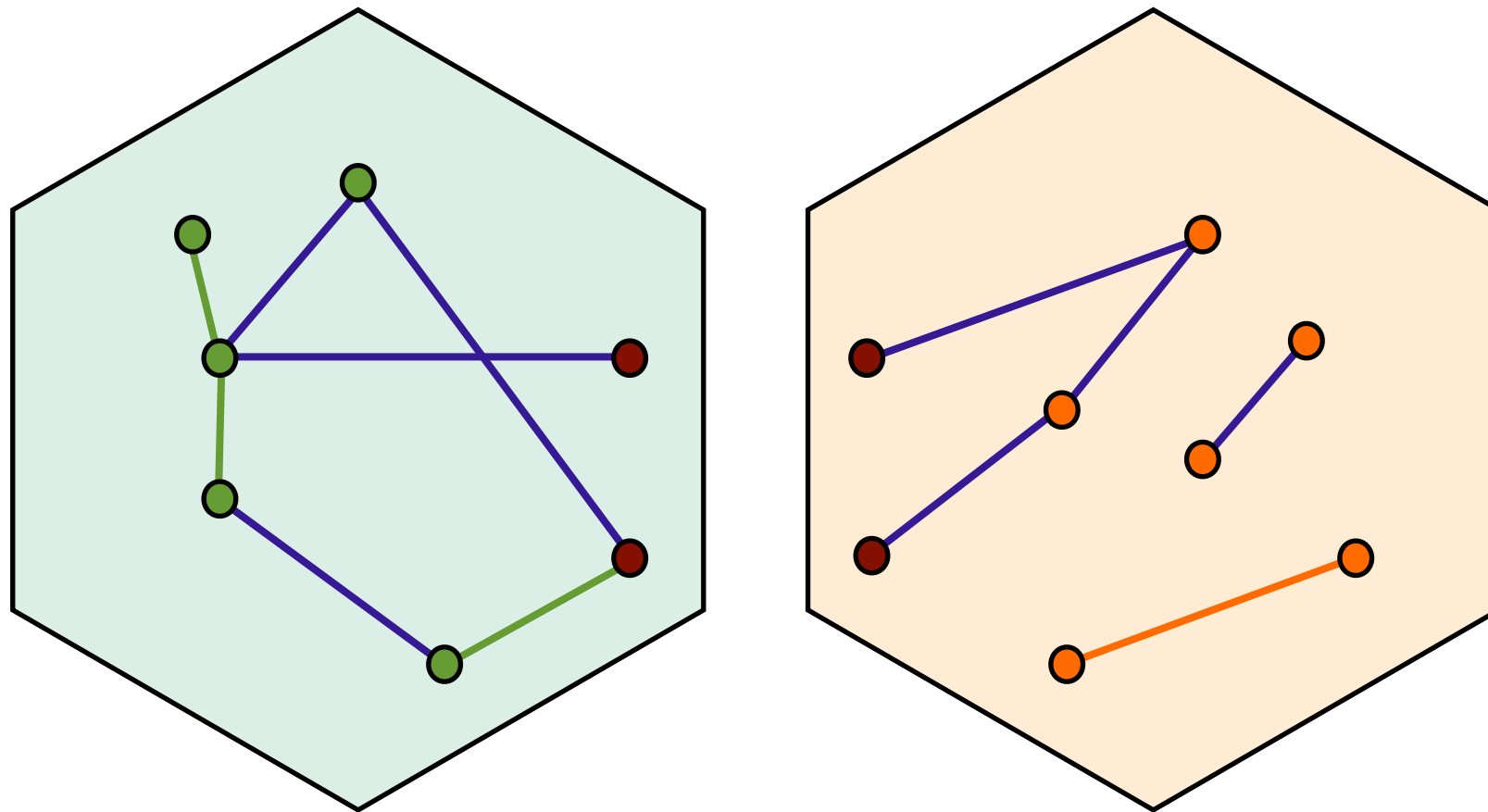
Distributed Views: Clarification



Problem 2

Distributed Views: Running example

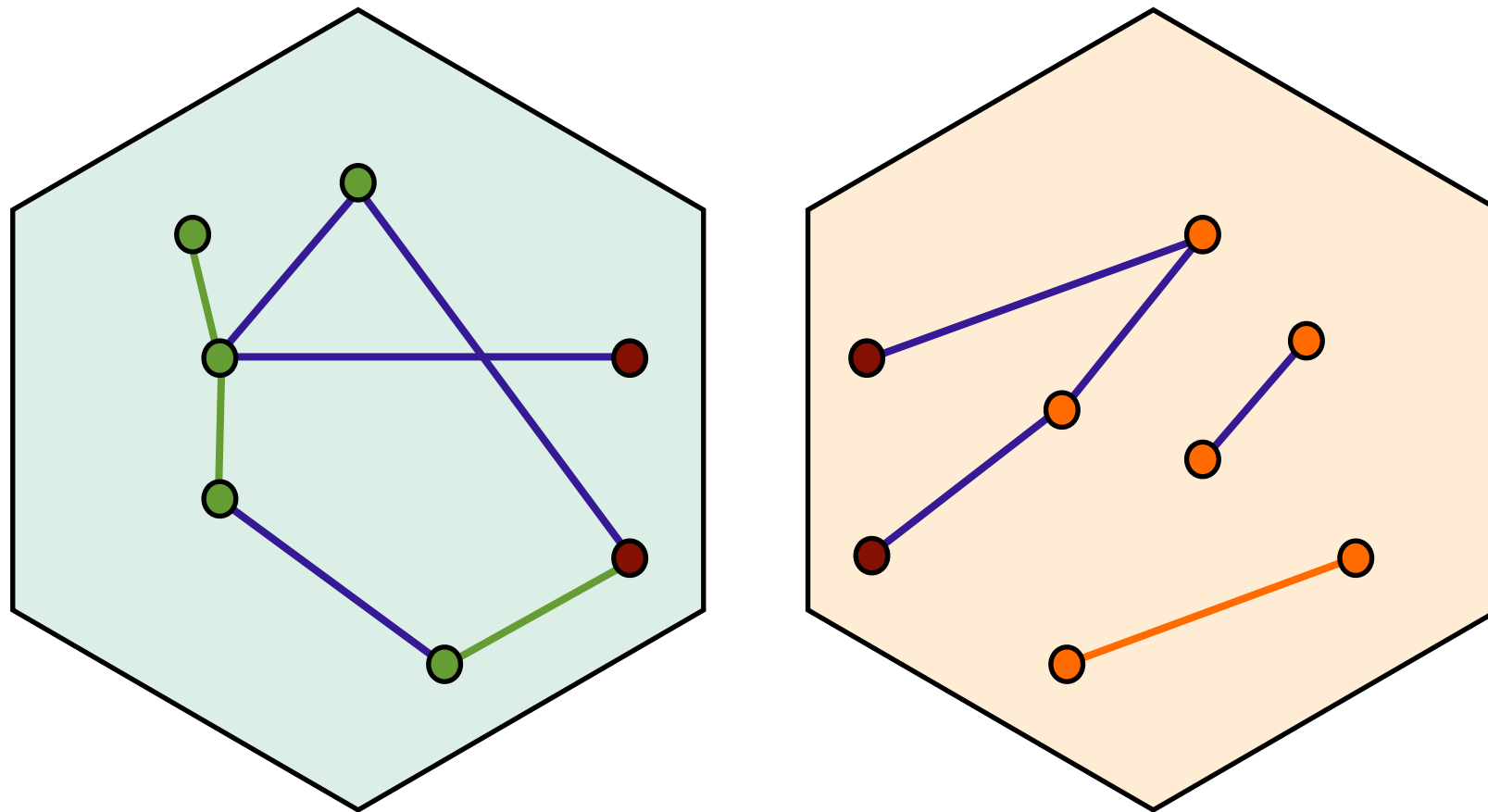
Inconsistency rule: Cycles for the relation **owned element** are forbidden.



Problem 2

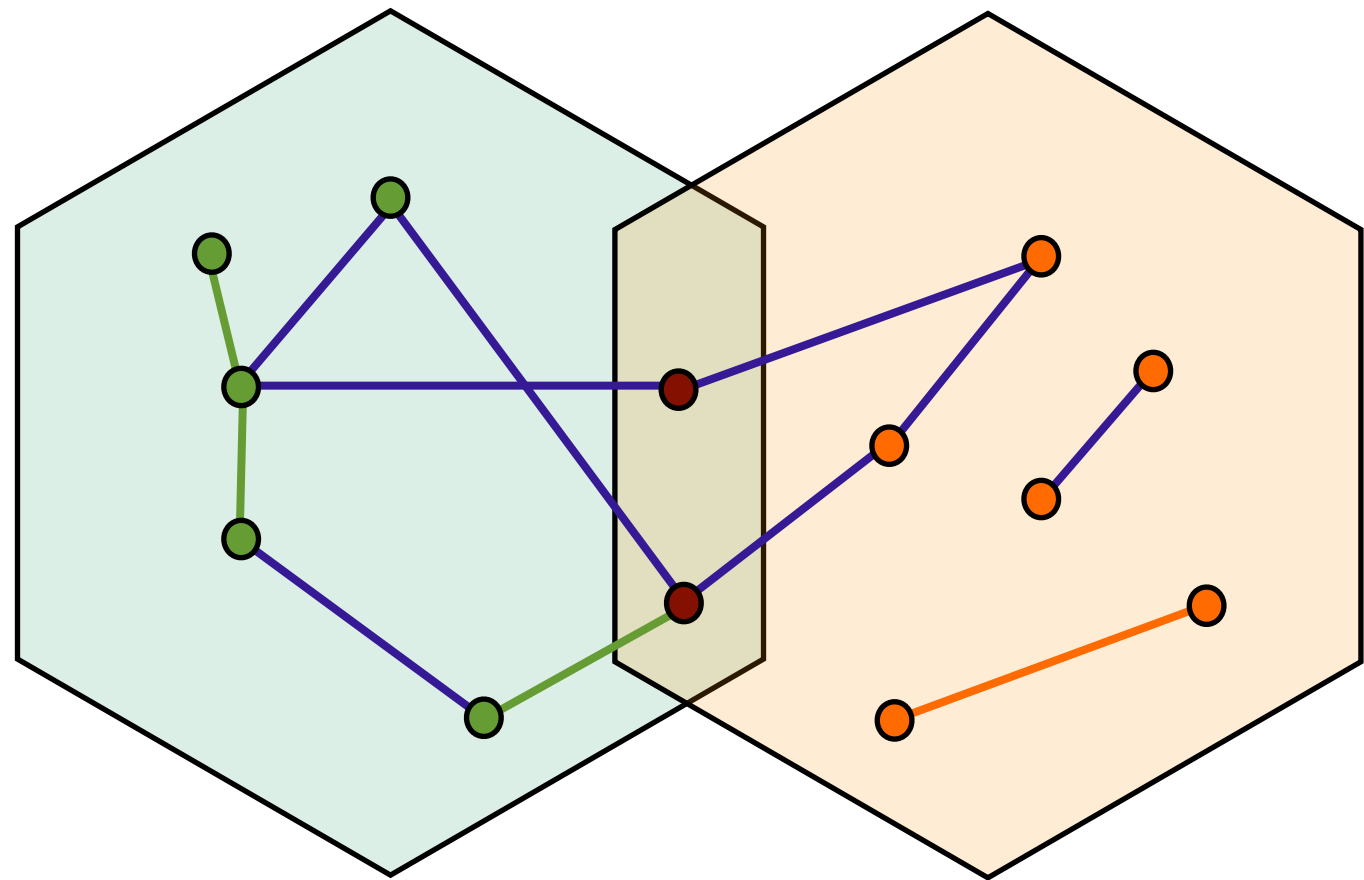
Distributed Views: Running example

Inconsistency rule: Cycles for the relation **owned element** are forbidden.



Proposal

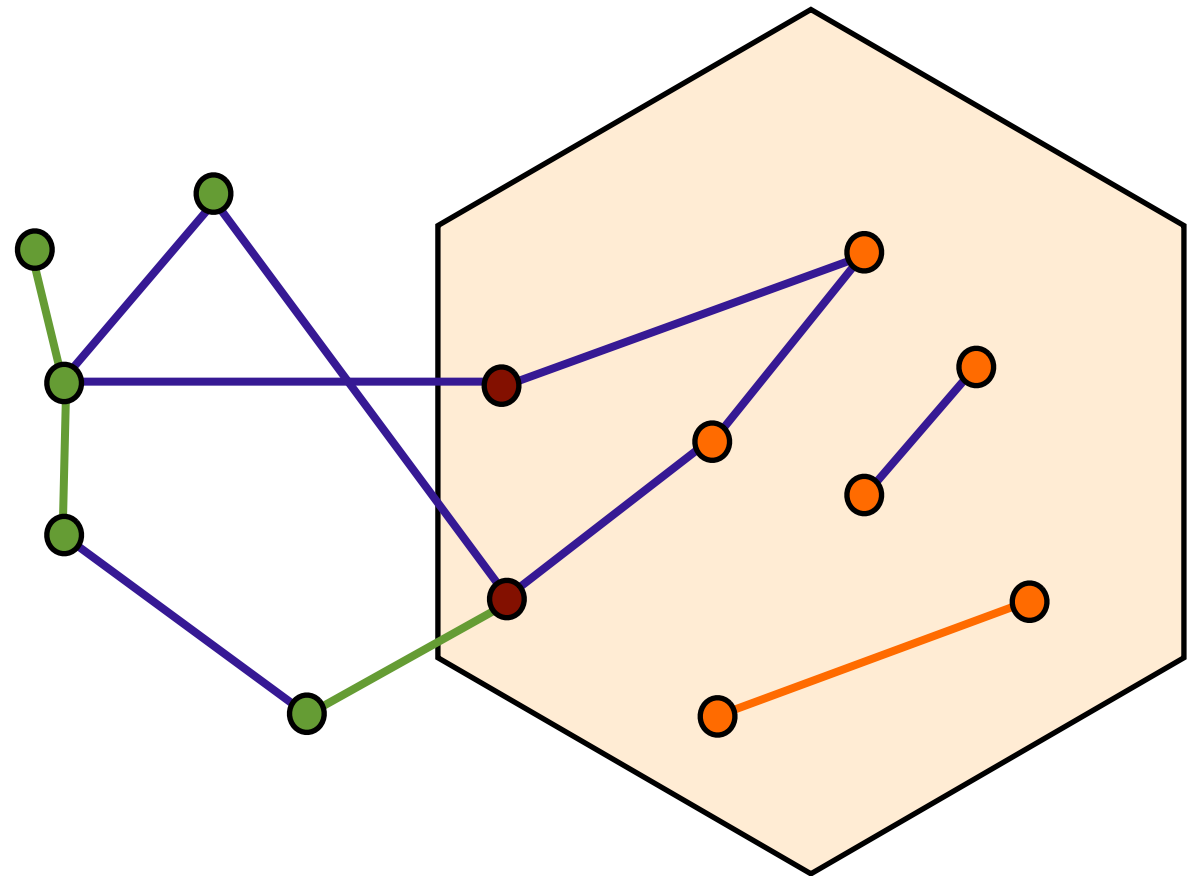
Single-View inconsistency detection



- Check fitted to one view
- Check that is fitted to one particular inconsistency rule
- Check for elements that are related to the considered view

Proposal

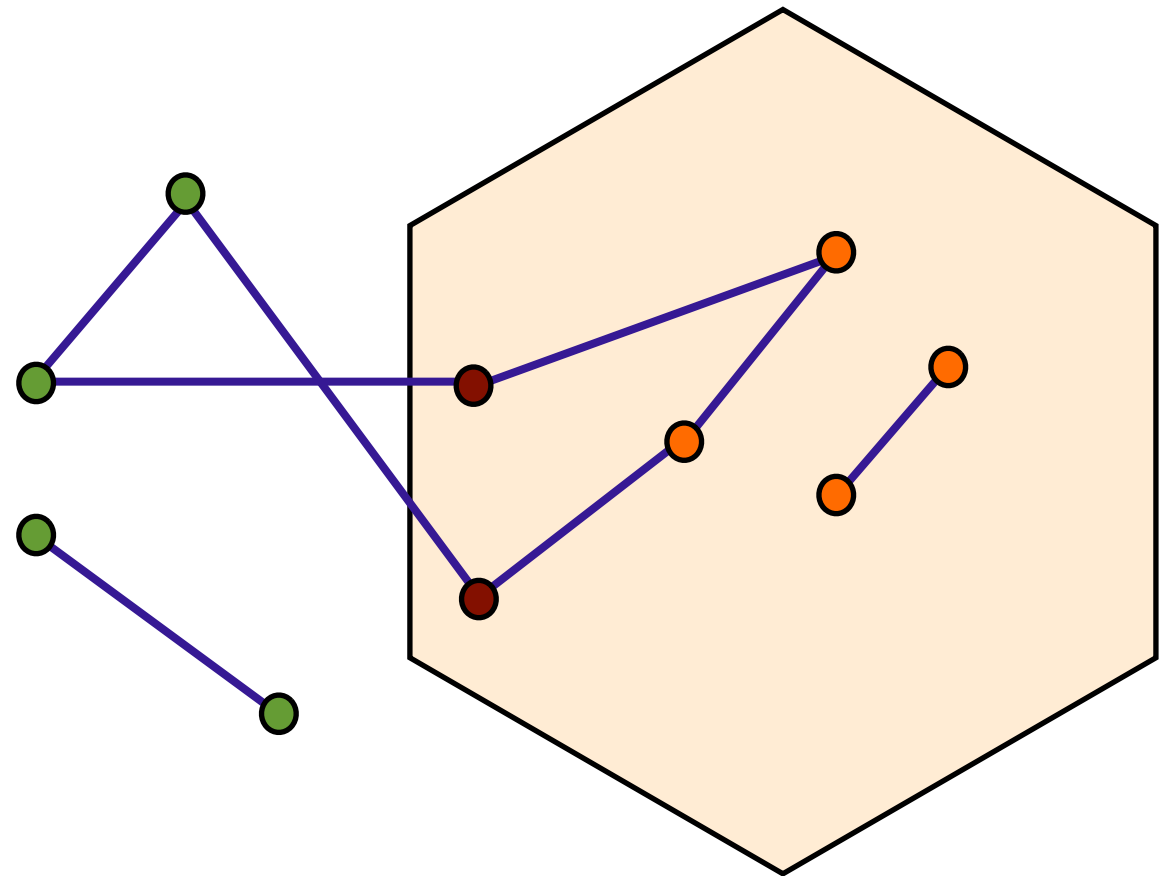
Single-View inconsistency detection



- Check fitted to one view
- Check that is fitted to one particular inconsistency rule
- Check for elements that are related to the considered view

Proposal

Single-View inconsistency detection

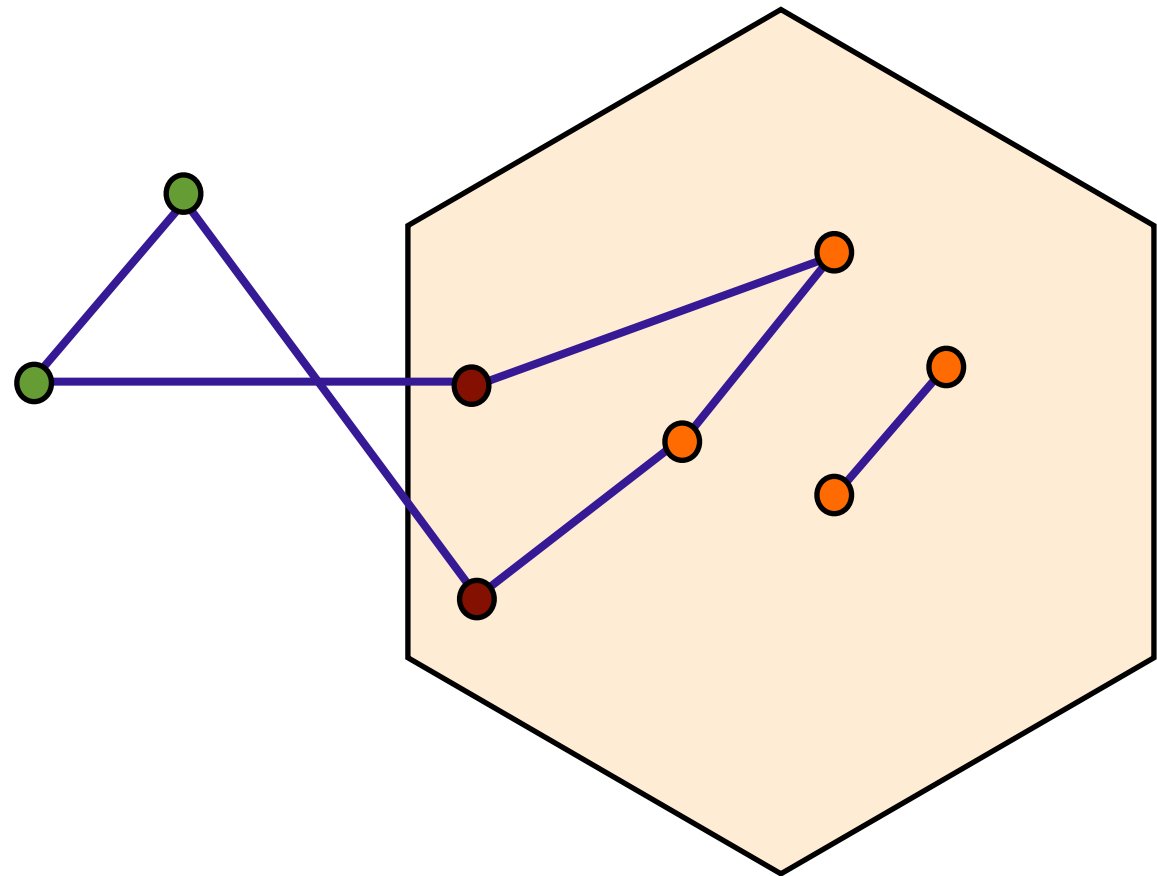


- Check fitted to one view
- Check that is fitted to one particular inconsistency rule
- Check for elements that are related to the considered view

Proposal

Single-View inconsistency detection

Single-View inconsistency detection



- Check fitted to one view
- Check that is fitted to one particular inconsistency rule
- Check for elements that are related to the considered view

How did we make it ?

We used **Praxis** to represent models, **DPraxis** for implementing the distributed views.

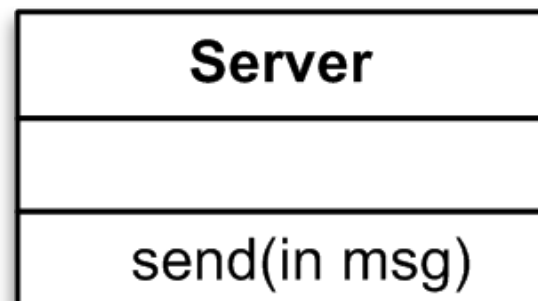
(1) We used the **Praxis Rules Impact Analysis** to filter the data that is specific to an inconsistency rule.

(2) We used the **DPraxis' routing tables** to filter and obtain the data that is related to the view.

Context:Praxis

Six unitary actions to represent models:

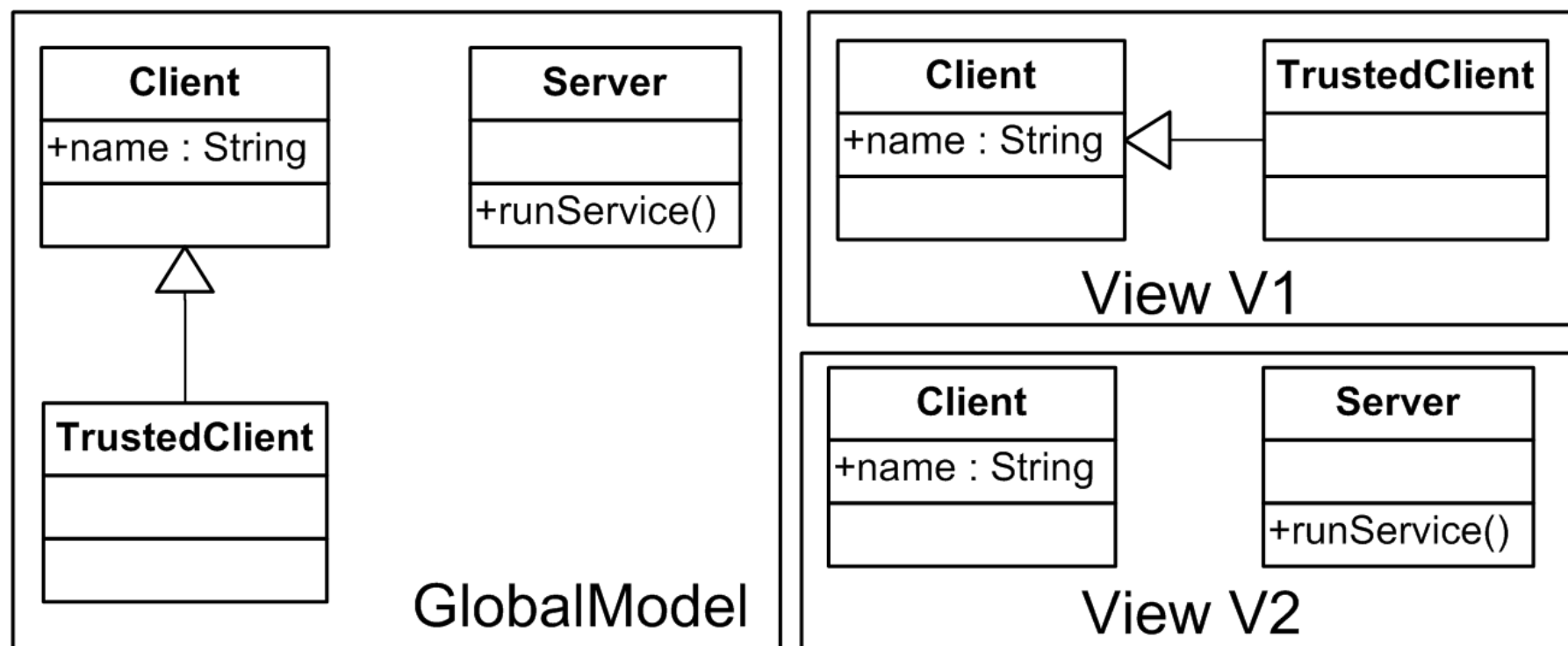
Create	Delete
AddProperty	RemProperty
AddReference	RemReference



- 1.Create(c1,class)
- 2.AddProperty(c1,name,Server)
- 3.Create(op, operation)
- 4.AddProperty(op, name, send)
- 5.AddReference(c1, ownedOperation,op)
- 6.AddReference(c1, ownedElement,op)
- 7.Create(p,parameter)
- 8.AddProperty(p,direction, in)
- 9.AddProperty(p,name, message)
- 10.AddReference(op,ownedParameter,p)
- 11.AddReference(op,ownedElement,p)
- 12.RemProperty(p,name,message)
- 13.AddProperty(p,name, msg)

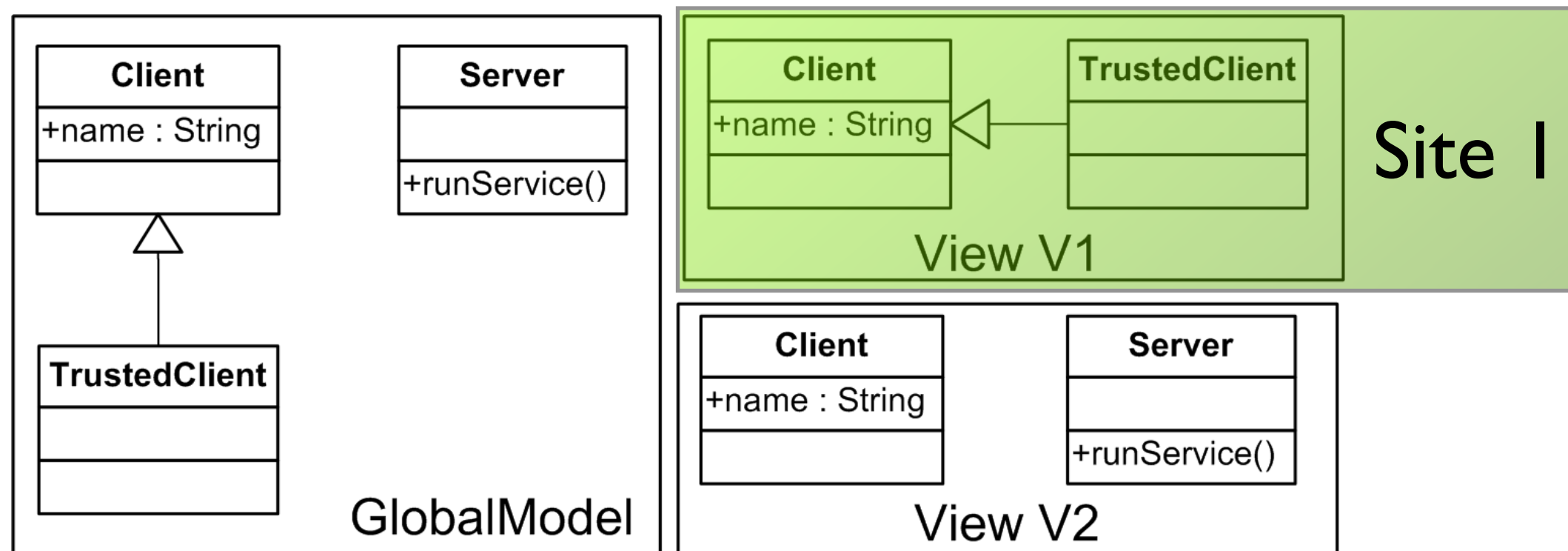
Context:DPraxis

We believe that P2P can tackle modeling scaling issues by providing only the needed information to each developer. DPraxis then shares the common subparts.



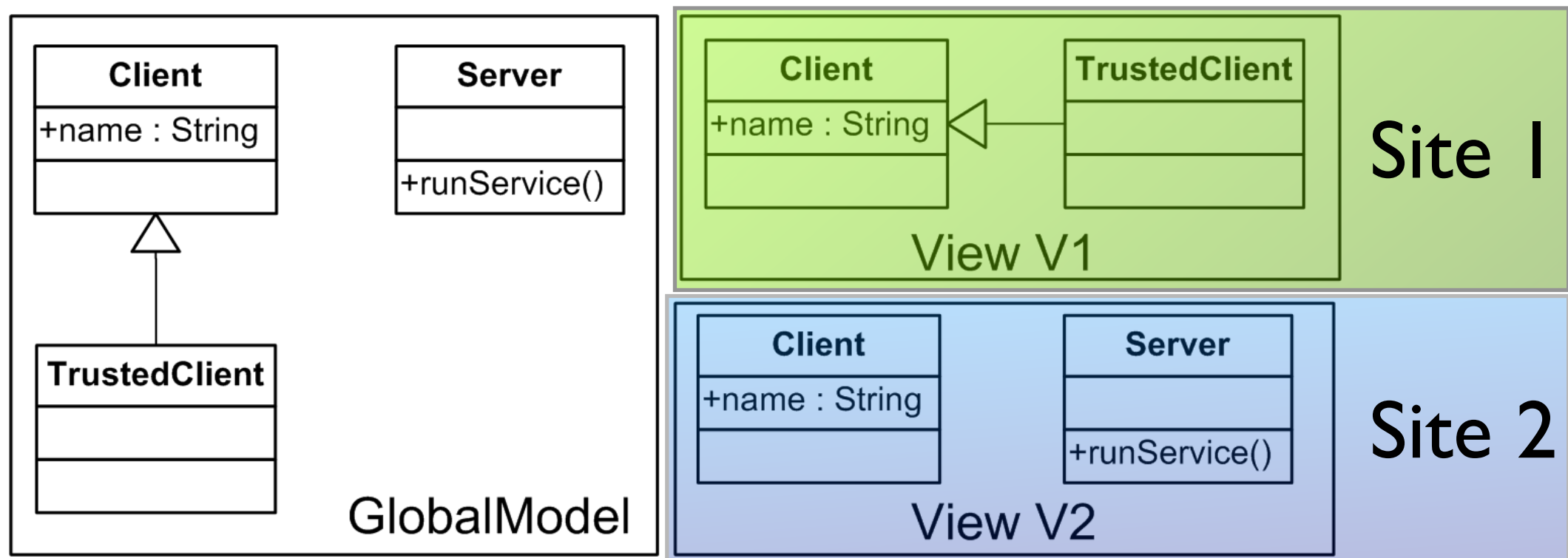
Context:DPraxis

We believe that P2P can tackle modeling scaling issues by providing only the needed information to each developer.
DPraxis then shares the common subparts.



Context:DPraxis

We believe that P2P can tackle modeling scaling issues by providing only the needed information to each developer.
DPraxis then shares the common subparts.

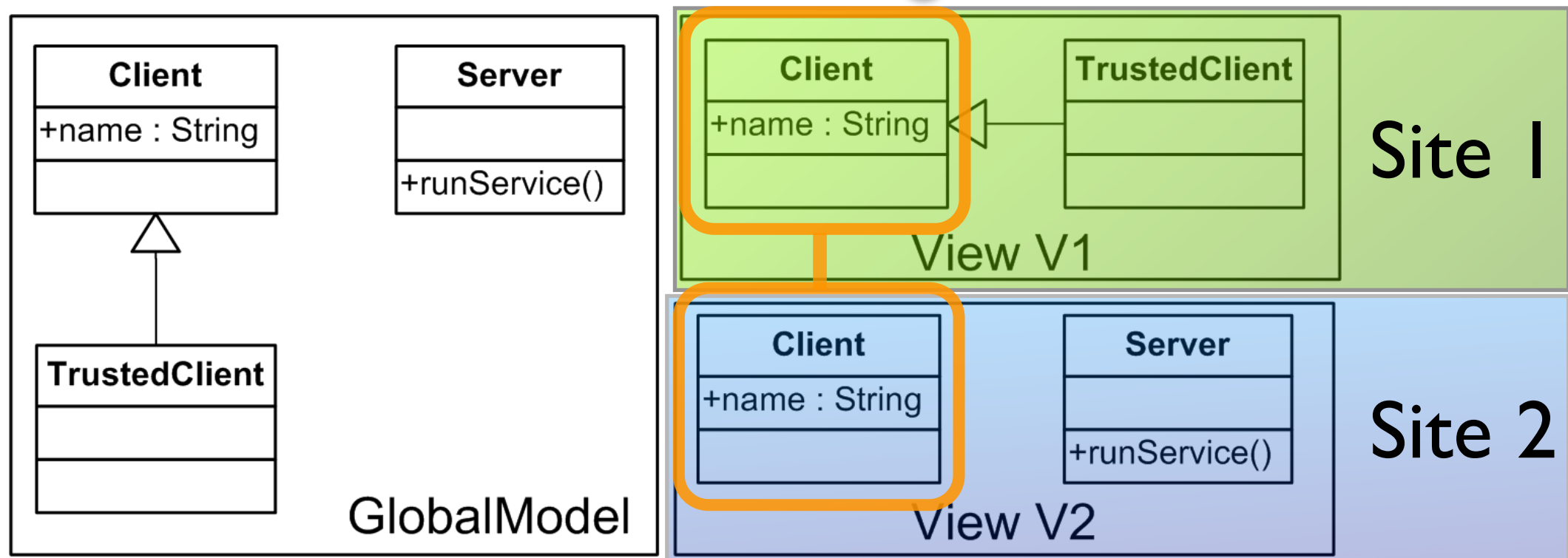


Context:DPraxis

We believe that P2P can tackle modeling scaling issues by providing only the needed information to each developer. DPraxis then shares the common subparts.

Common SubParts

Sharing

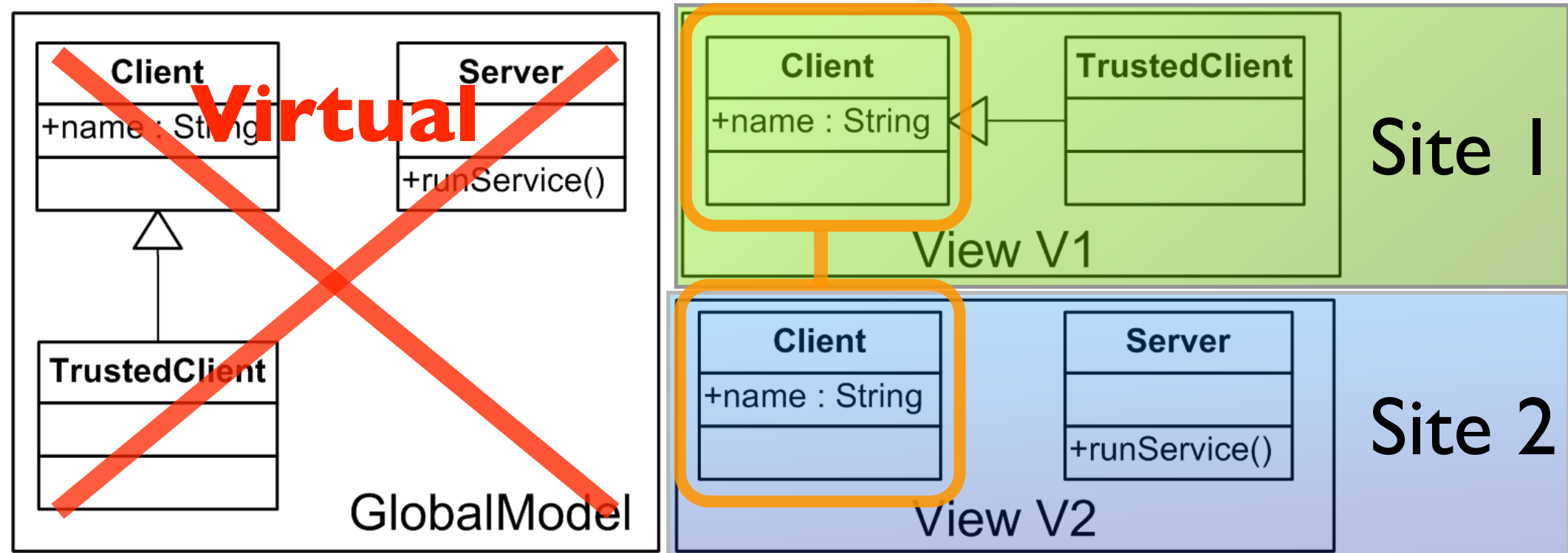


Context:DPraxis

We believe that P2P can tackle modeling scaling issues by providing only the needed information to each developer. DPraxis then shares the common subparts.

Common SubParts

Sharing



Filtering with Praxis Rules Impact Analysis

Praxis Rule

Cycle(A) :- Path(A,A)

Path(A,B) :-

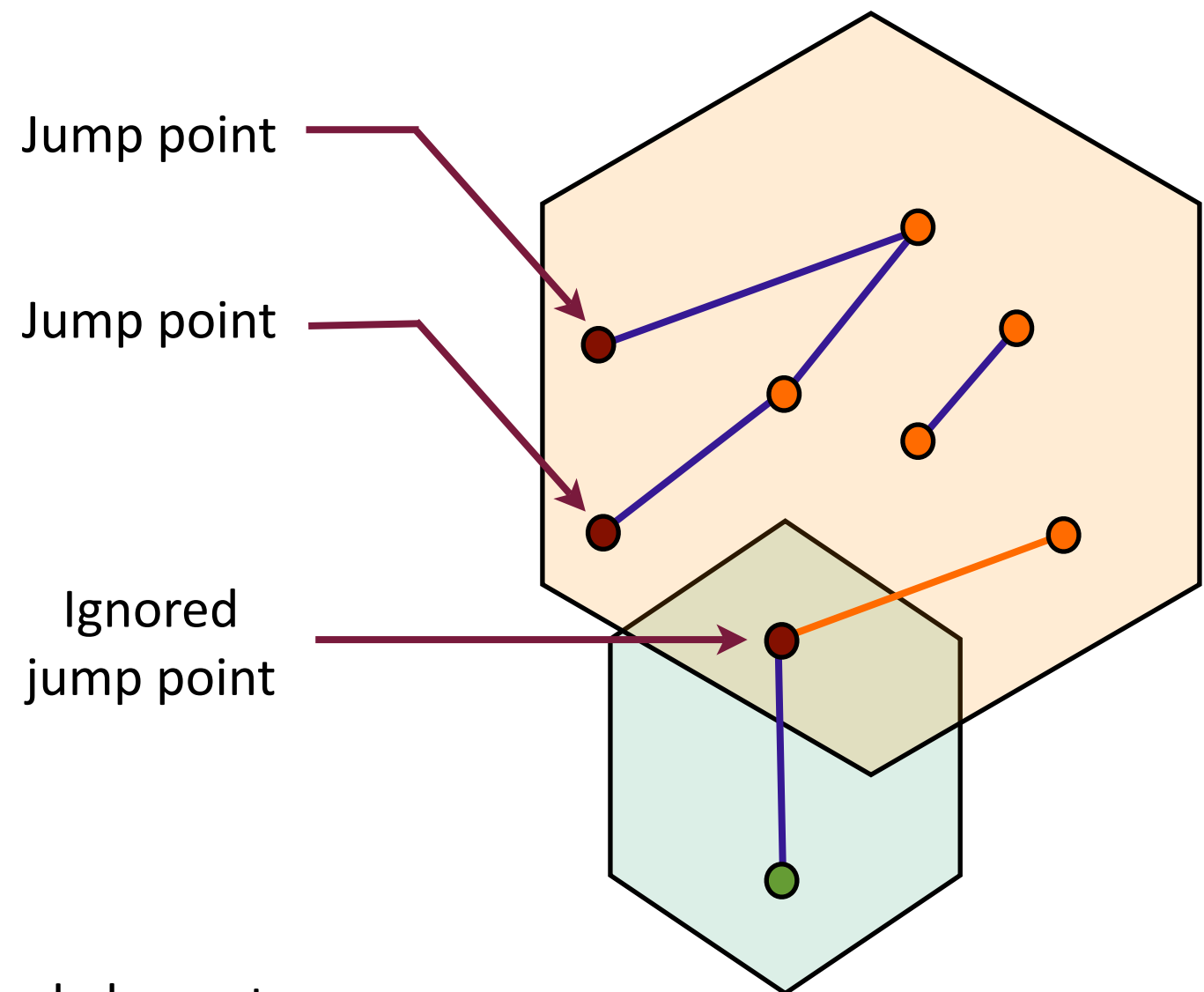
LastAddReference(A,ownedElement,B) or
Path(A,X) and Path(X,B)

Unitary actions classes that have an
impact on the rule Cycle

AddReference(_,ownedElement,_)
RemReference(_,ownedElement,_)

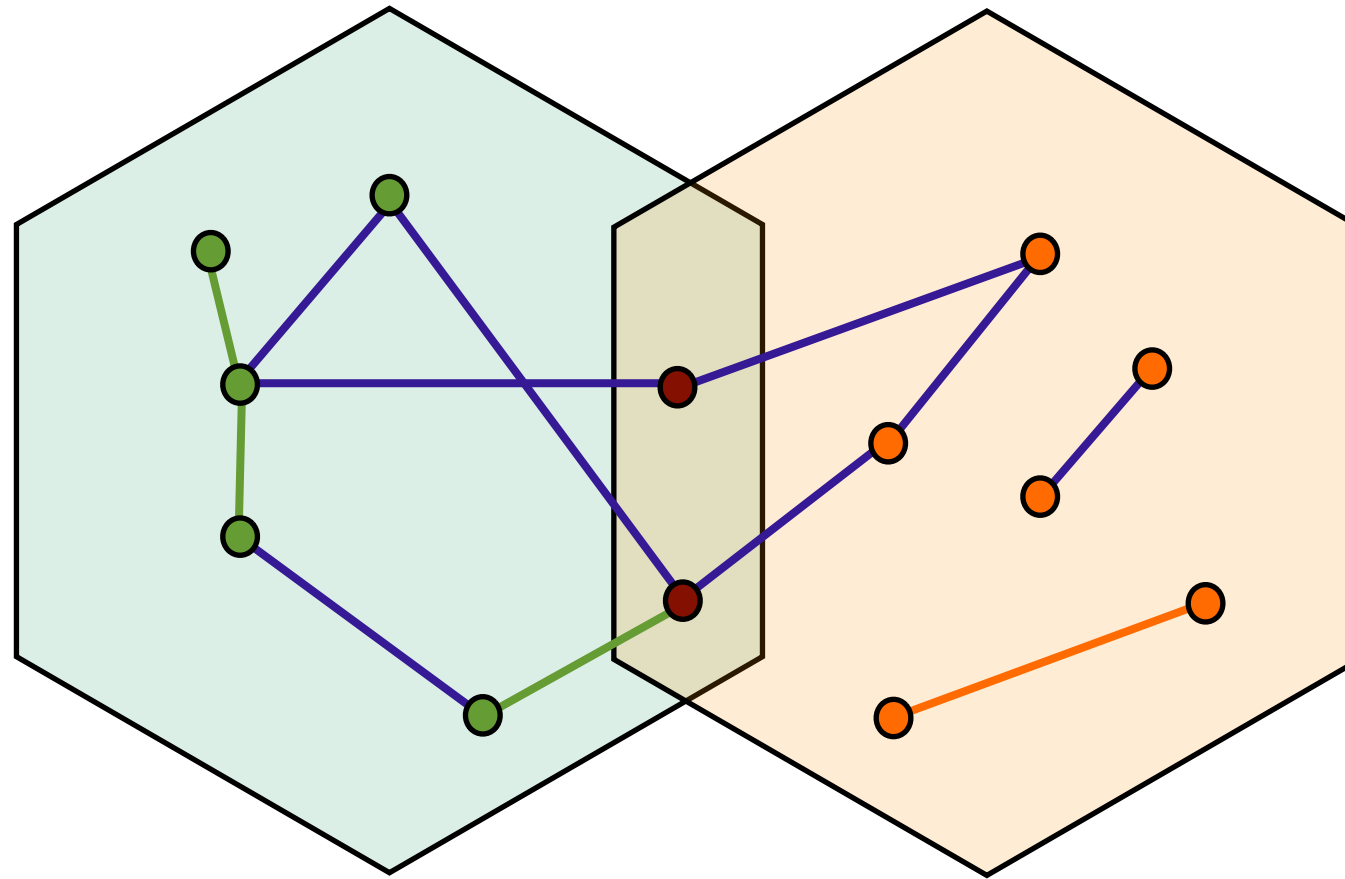
Determining the Jump points

Jump point: Model element that is replicated and that owns at least one reference that can be crossed by the considered inconsistency rule.



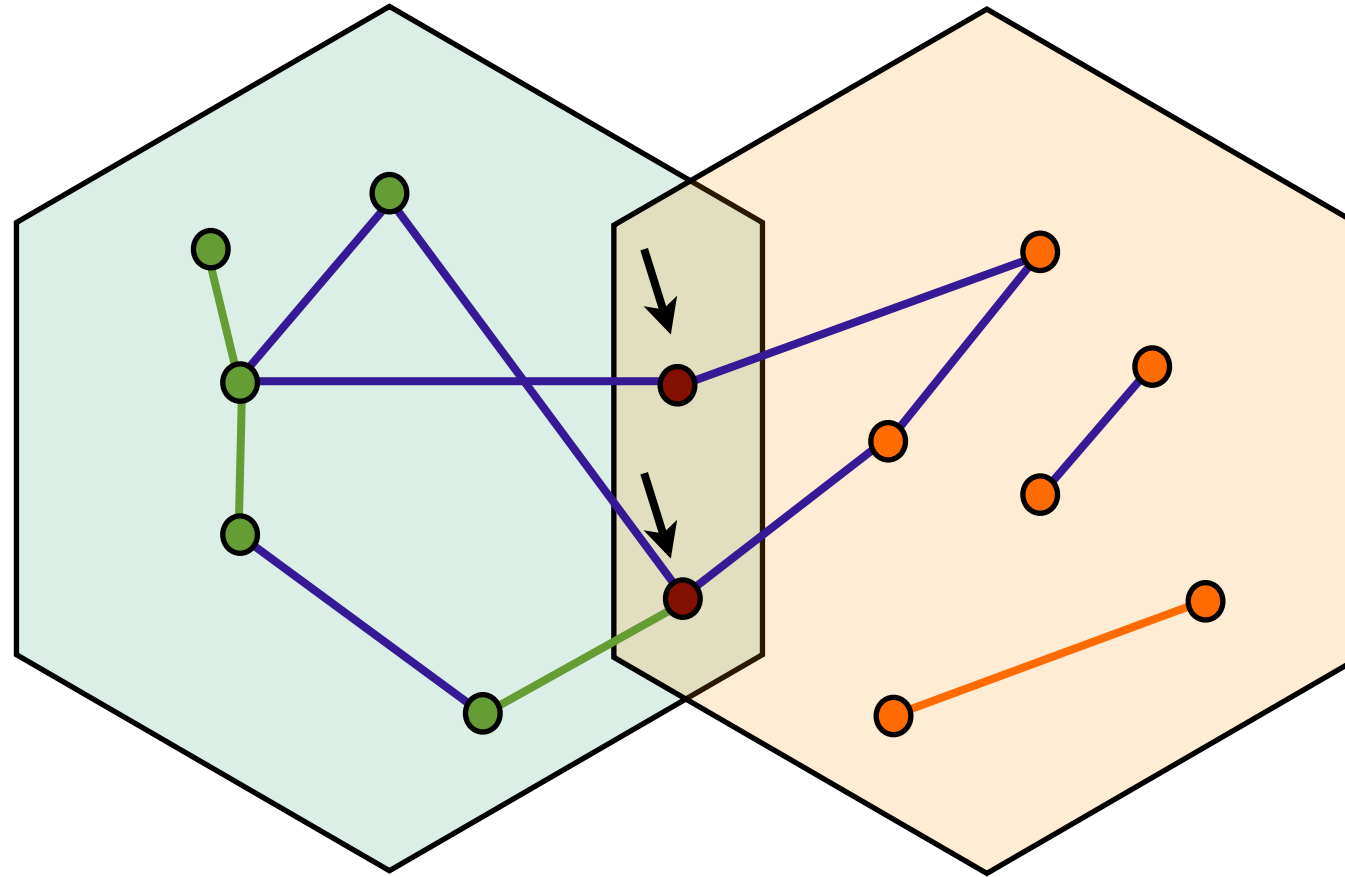
- We use DPraxis to get the list of replicated elements.
- We use the impact analysis to find if there are any references for these elements that can be crossed by the inconsistency rule.

Single-View Inconsistency detection



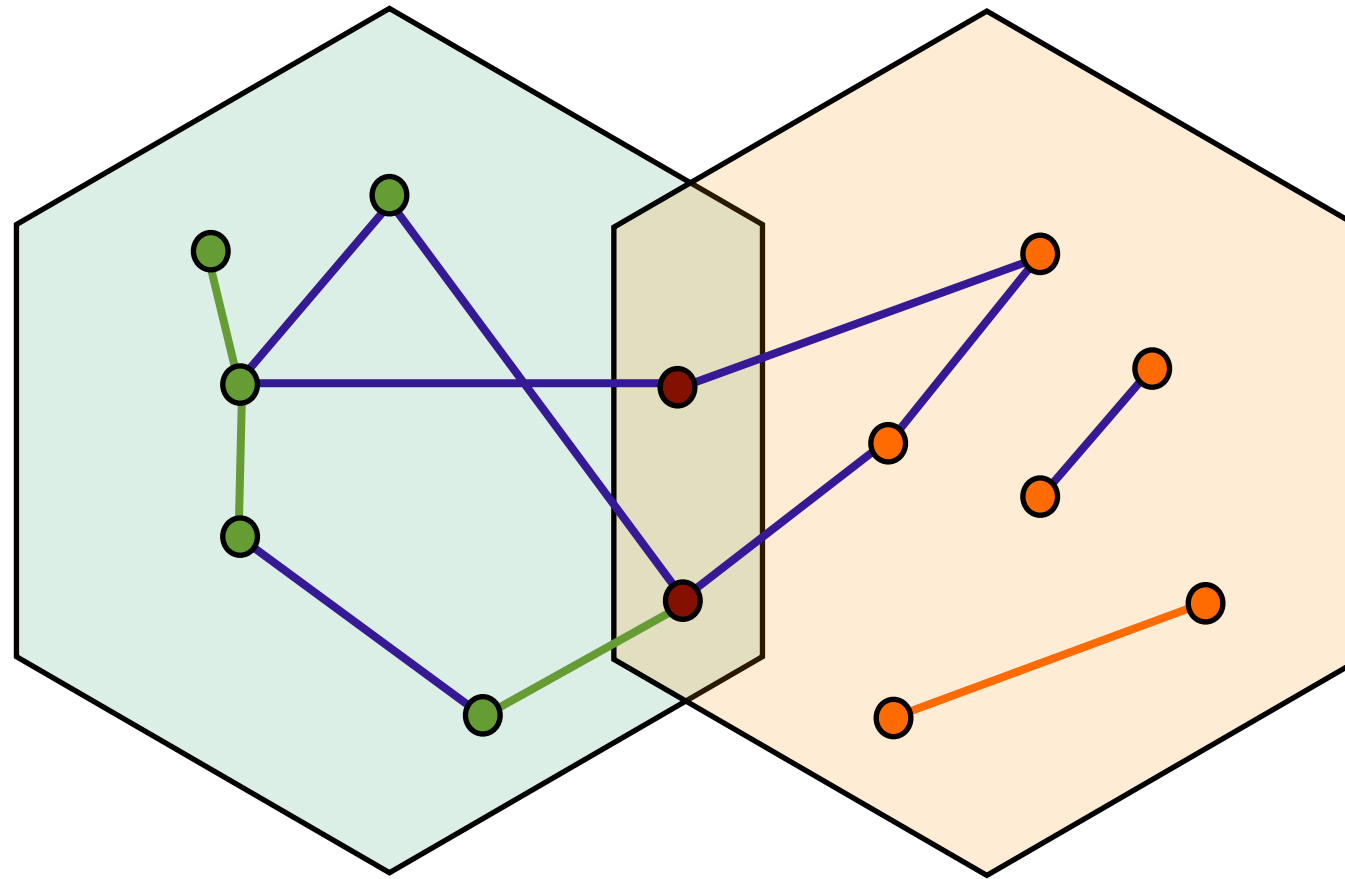
- Determine the Jump Points
- For each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



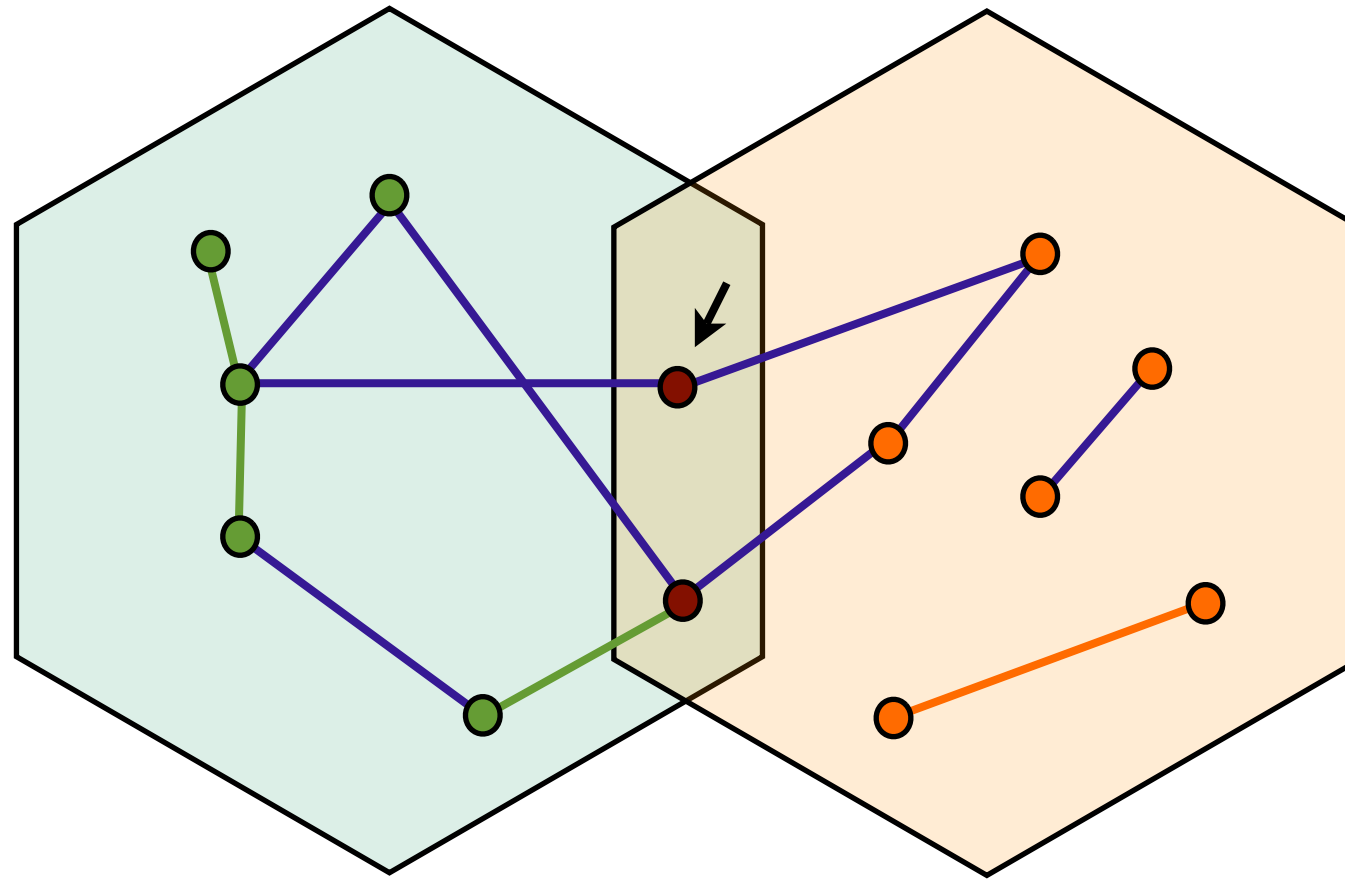
- Determine the Jump Points
- For each each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



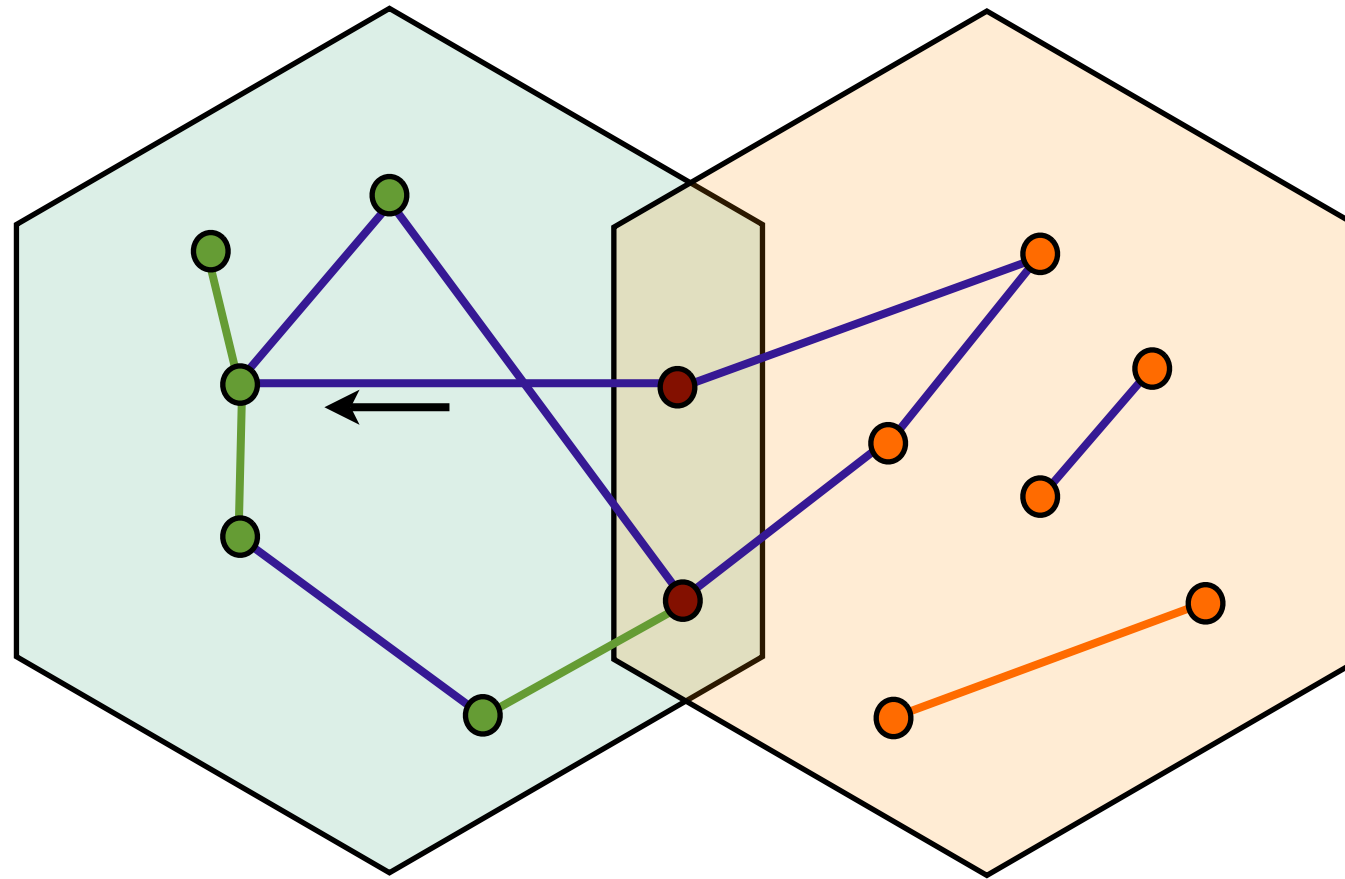
- Determine the Jump Points
- For each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



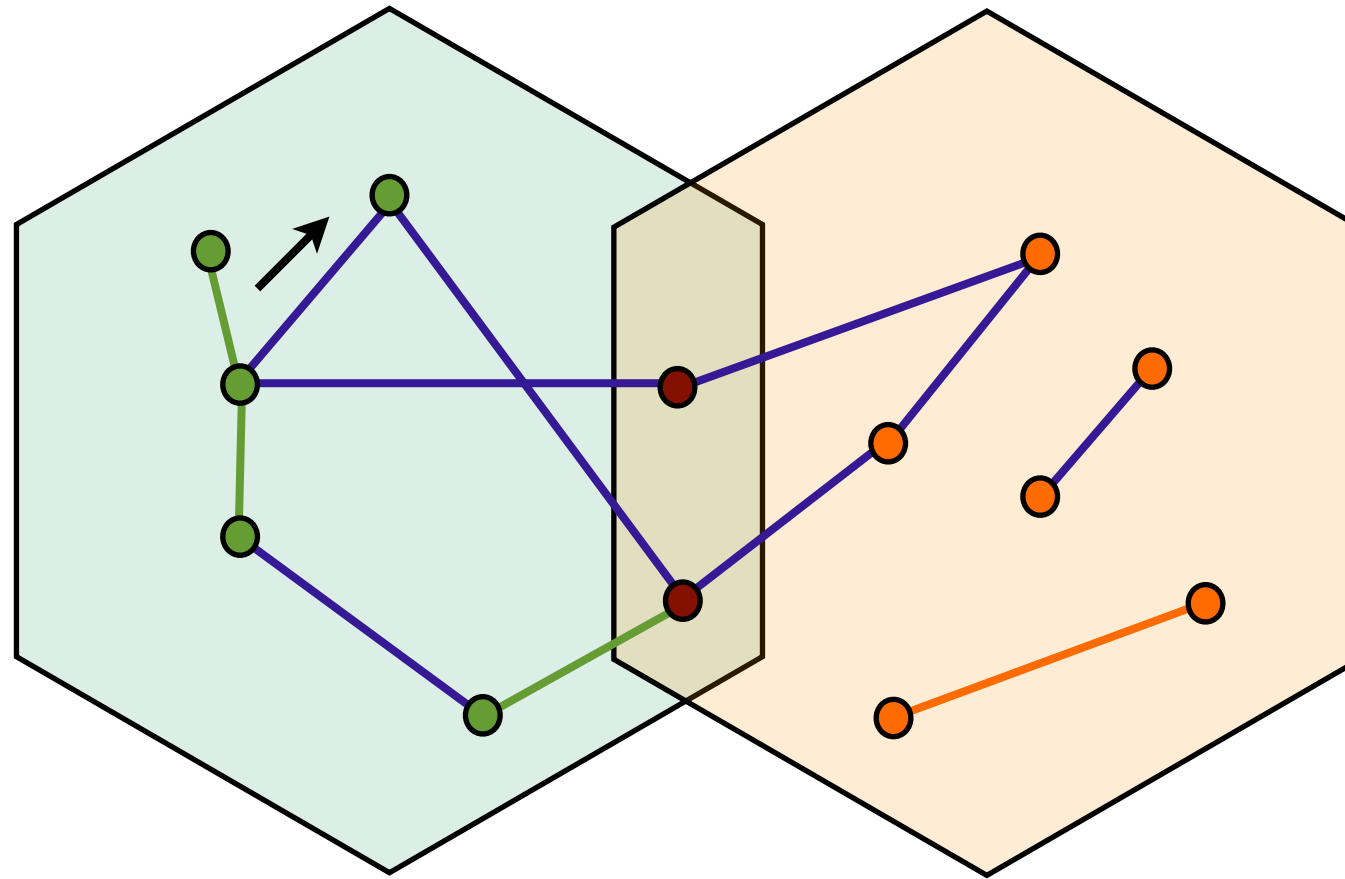
- Determine the Jump Points
- For each each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



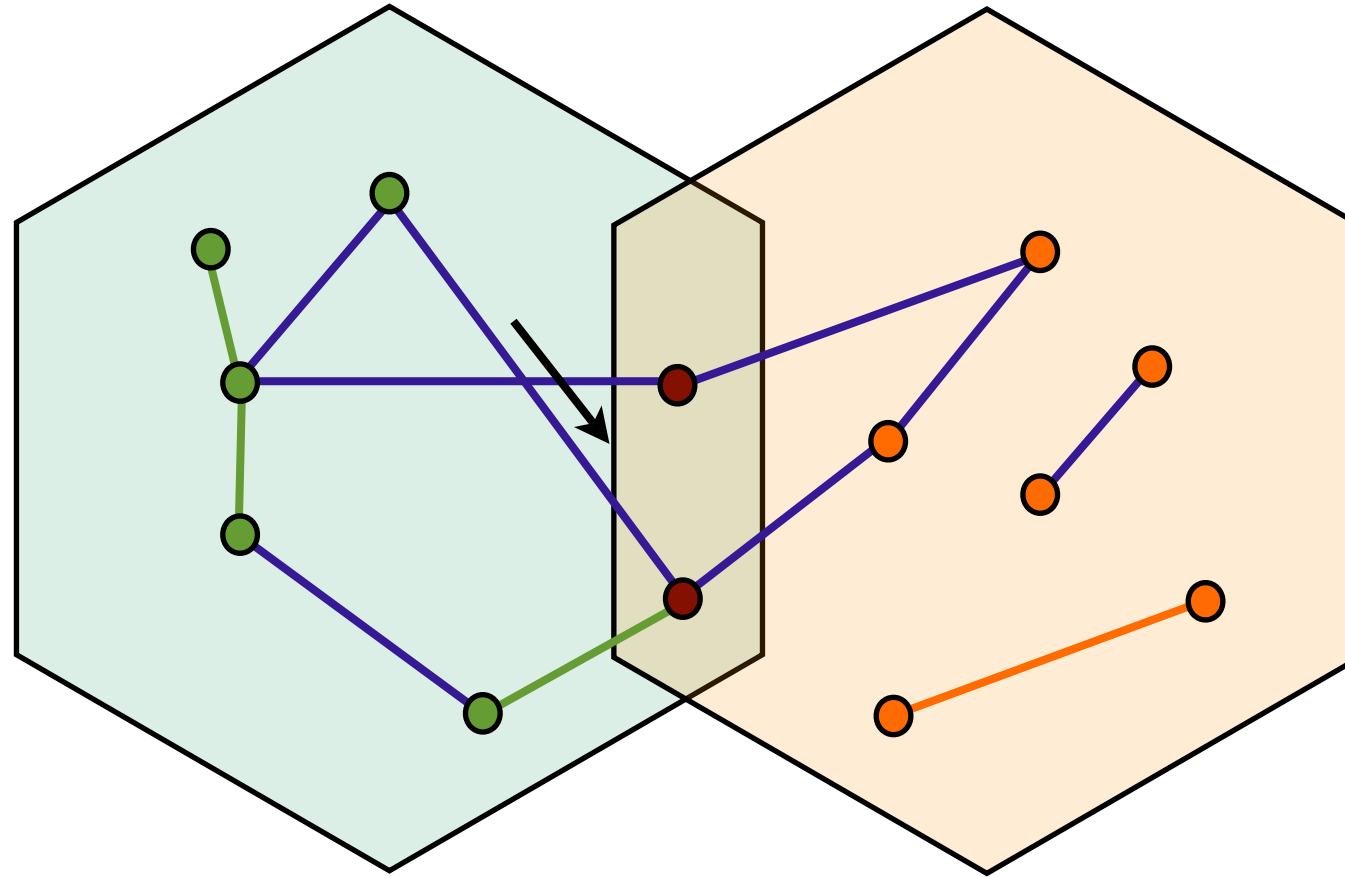
- Determine the Jump Points
- For each each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



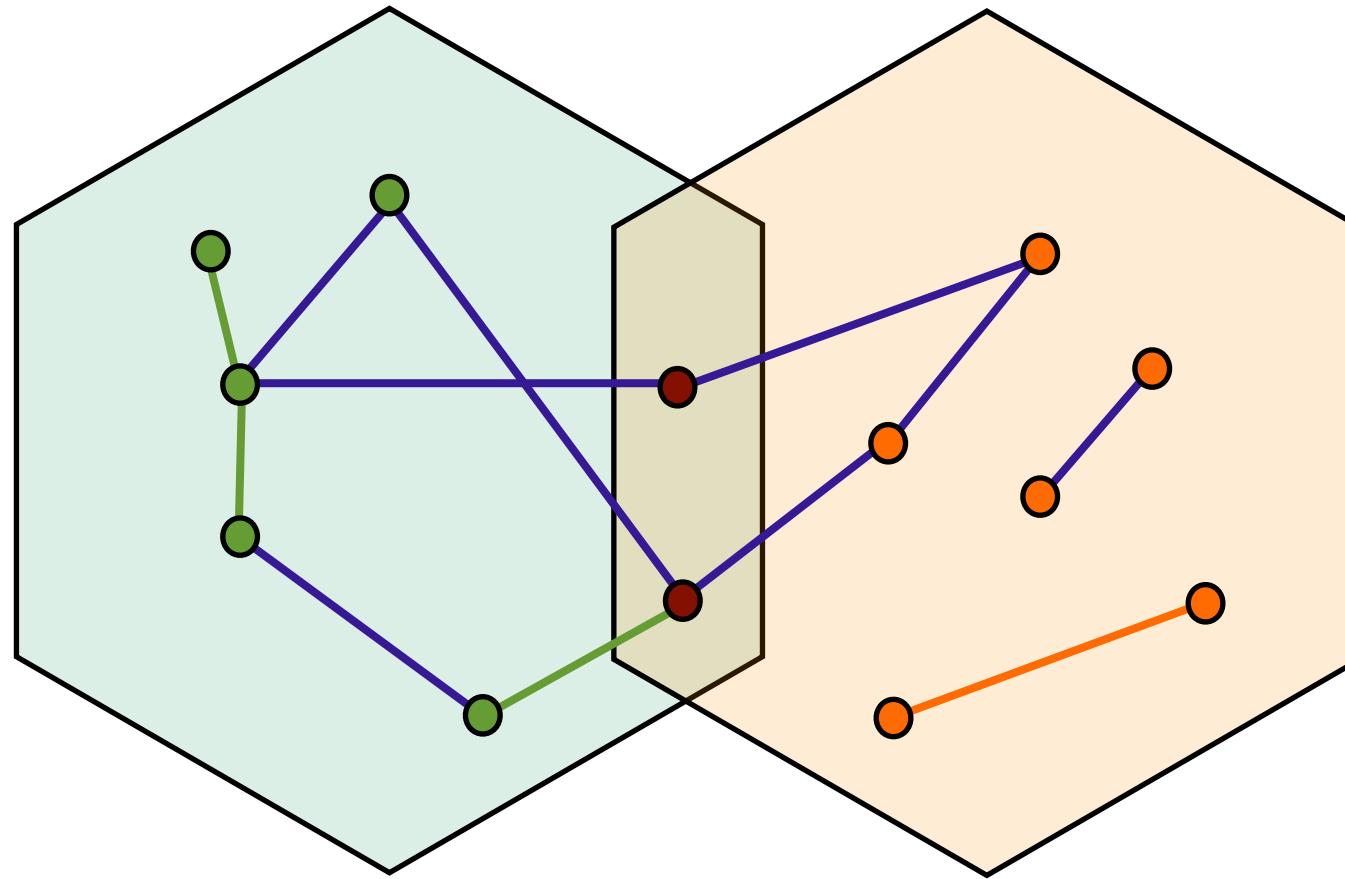
- Determine the Jump Points
- For each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



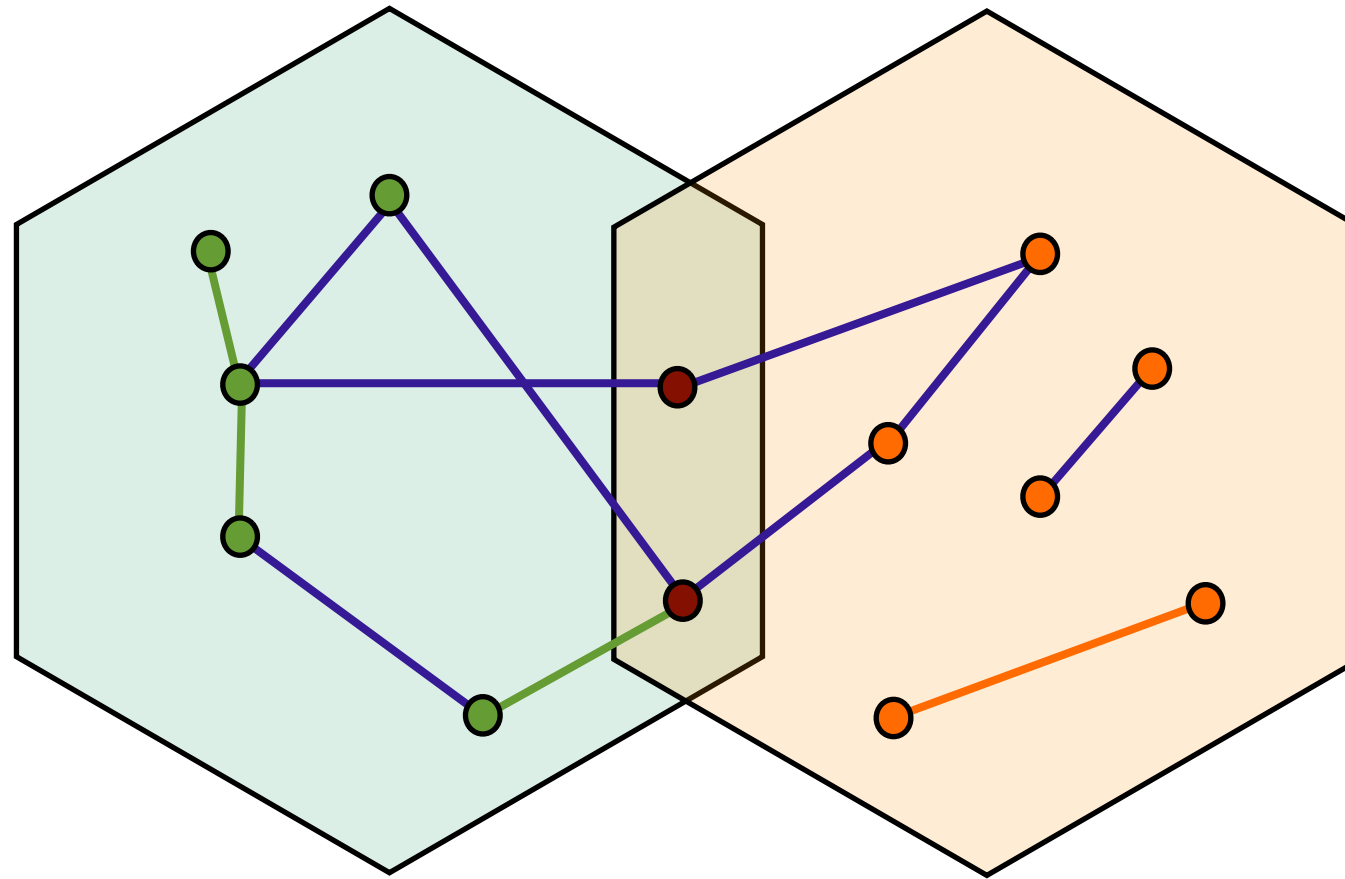
- Determine the Jump Points
- For each each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



- Determine the Jump Points
- For each each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Single-View Inconsistency detection



- Determine the Jump Points
- For each view that owns a replica of a Jump point:
 - A request for the interesting unitary actions is made by asking for a closure over the considered relations and elements.
 - A recursive call is possible when a new Jump Point is hit.
- An inconsistency detection is executed on the local view plus the gathered elements.

Video Demo

