An Algebraic Framework for Merging Incomplete and Inconsistent Views

Mehrdad Sabetzadeh   **Steve Easterbrook**

Department of Computer Science
University of Toronto, Canada.

Email: {mehrdad,sme}@cs.toronto.edu

July 6, 2005

Motivation

- **What is view merging?**
  - Putting a set of views together to produce a new view encompassing all the given views
- **Why is view merging important?**
  - Creating a unified perspective
  - Exploring interactions between views
  - Performing various kinds of analyses

Applications of View Merging

- **Requirements Engineering**
  - Merging goal models
  - Merging behavioral models
- **Databases**
  - Schema integration
- **The Semantic Web**
  - Ontology integration

Challenges

- **Scalability**
  - . . . both cognitive and computational
- **Inconsistency**
  - Vocabulary differences; structural discrepancies; conceptual disagreements
- **Partiality**
  - Views are usually incomplete; stakeholders often have varying degrees of certainty about their decisions
- **Genericity**
  - . . . substantiated by the interdisciplinary nature of the view merging problem

Characteristics of Our Solution

- . . . draws on the ViewPoints framework (**cognitive scalability**, **partiality**, **inconsistency**)
- . . . provides a polytime merge algorithm (**computational scalability**)
- . . . allows for modeling of stakeholders' beliefs using annotations (**partiality**, **inconsistency**)
- . . . provides a generic approach for capturing evolution and describing view relationships (**genericity**)

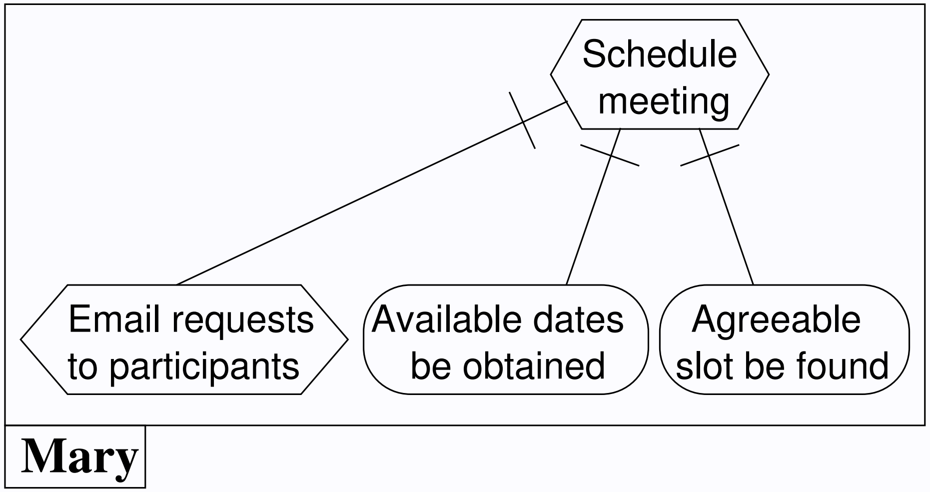
Outline

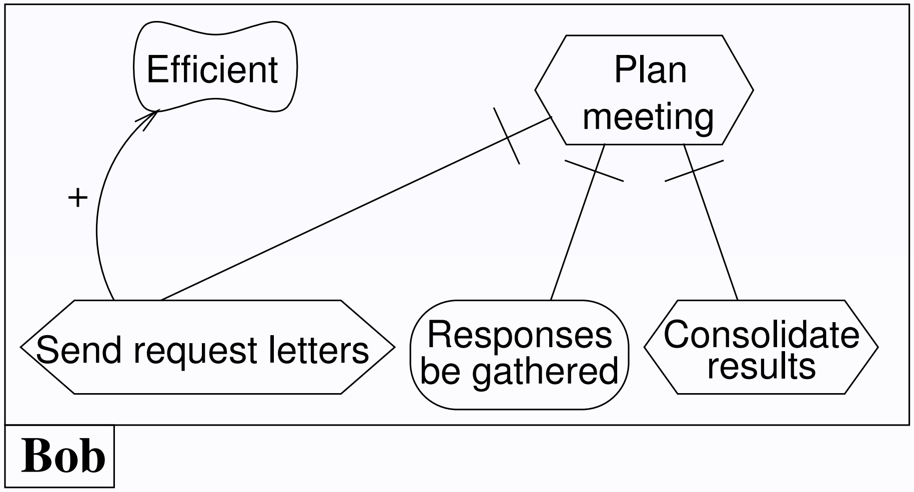
- ➜ Example: Merging i^* views
- ➜ Overview of the approach
- ➜ Core concepts: Views, Mappings, Diagrams
- ➜ The merge algorithm
- ➜ Example, revisited
- ➜ Adding support for traceability
- ➜ Recap
- ➜ Related work
- ➜ Future work

Example: A Meeting Scheduler

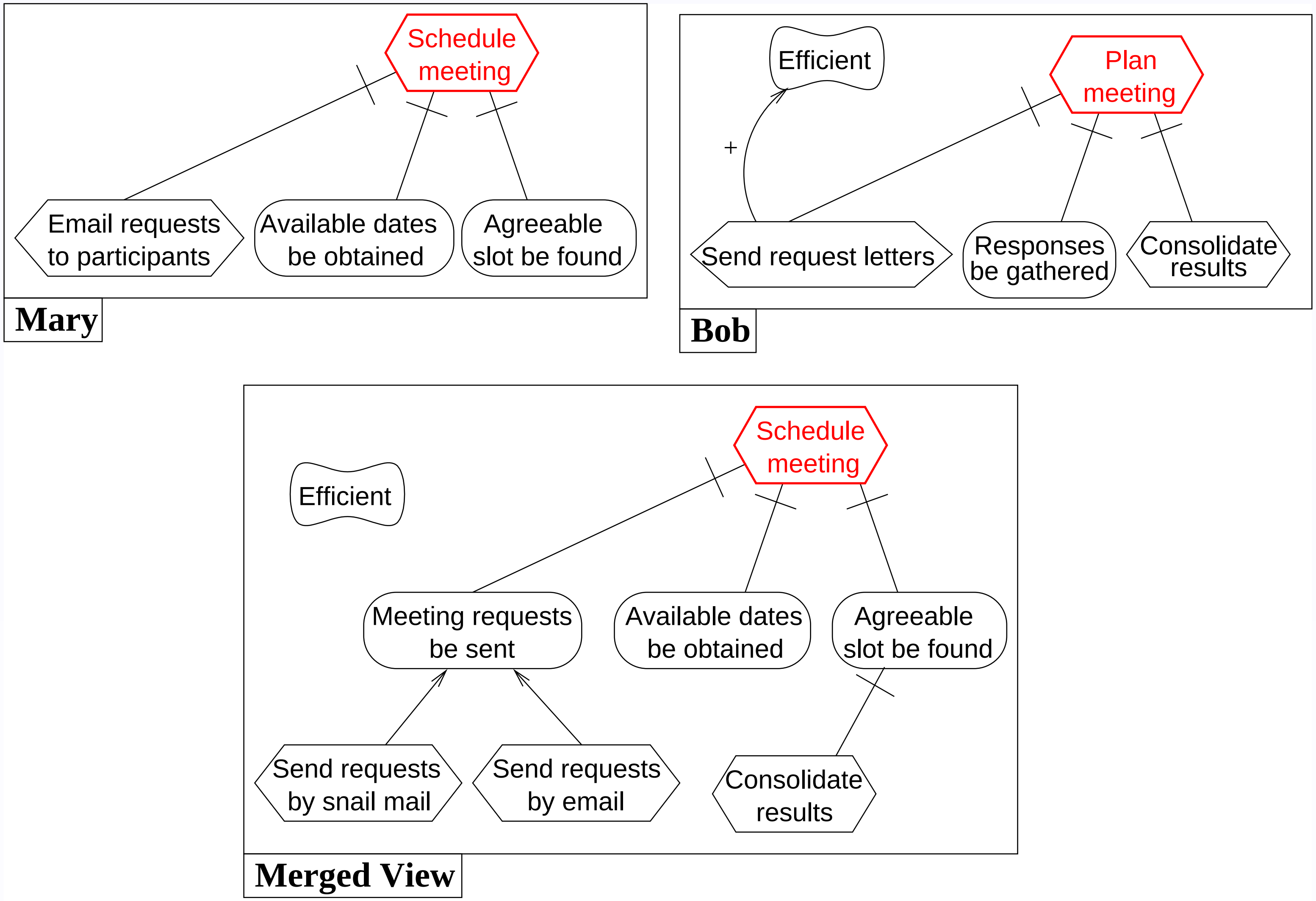
➜ **Problem**: Bob and Mary want to elaborate their requirements with the help of an analyst, Sam

➜ **Initial views of Bob and Mary**:



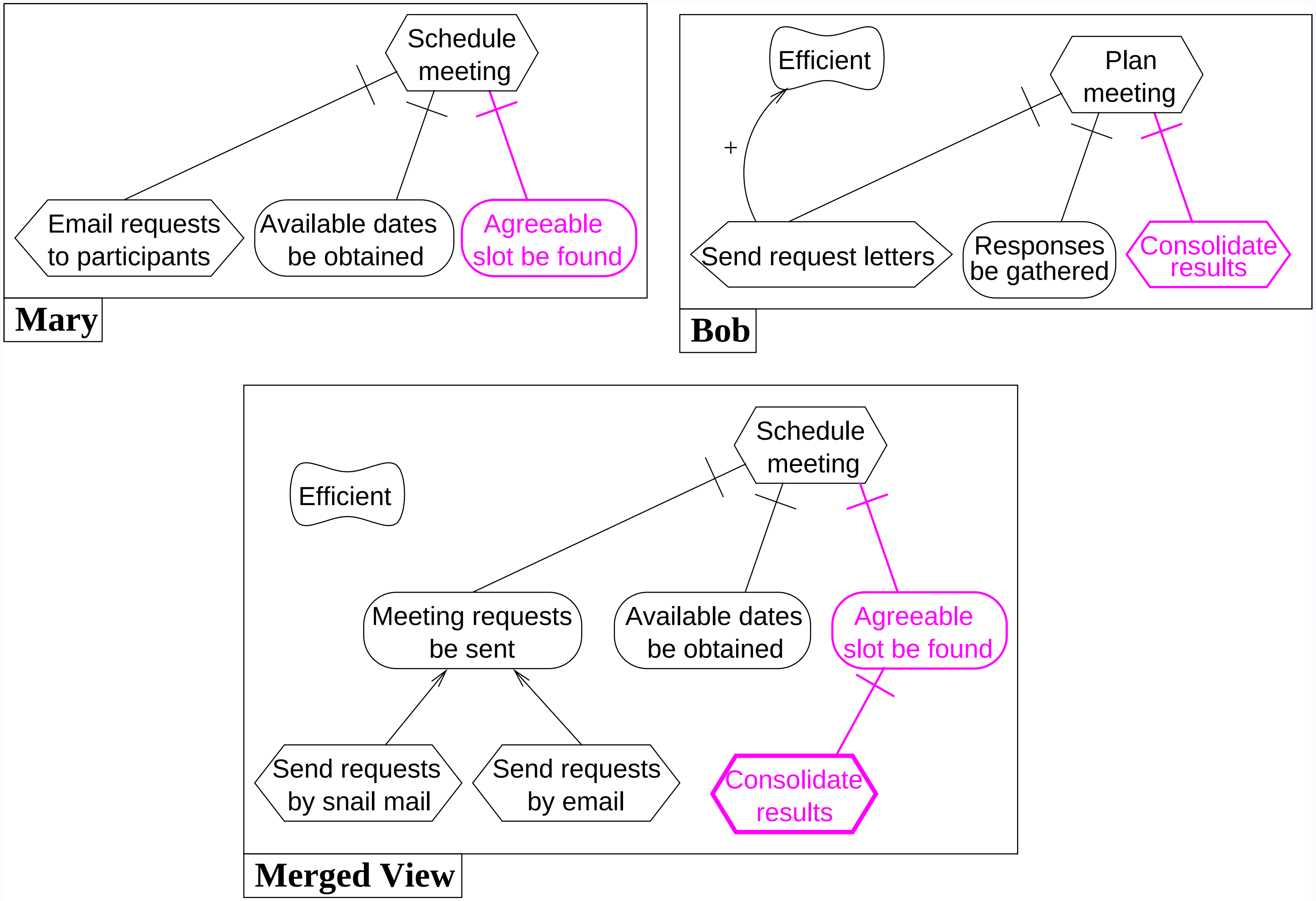


Different Aspects of the Problem



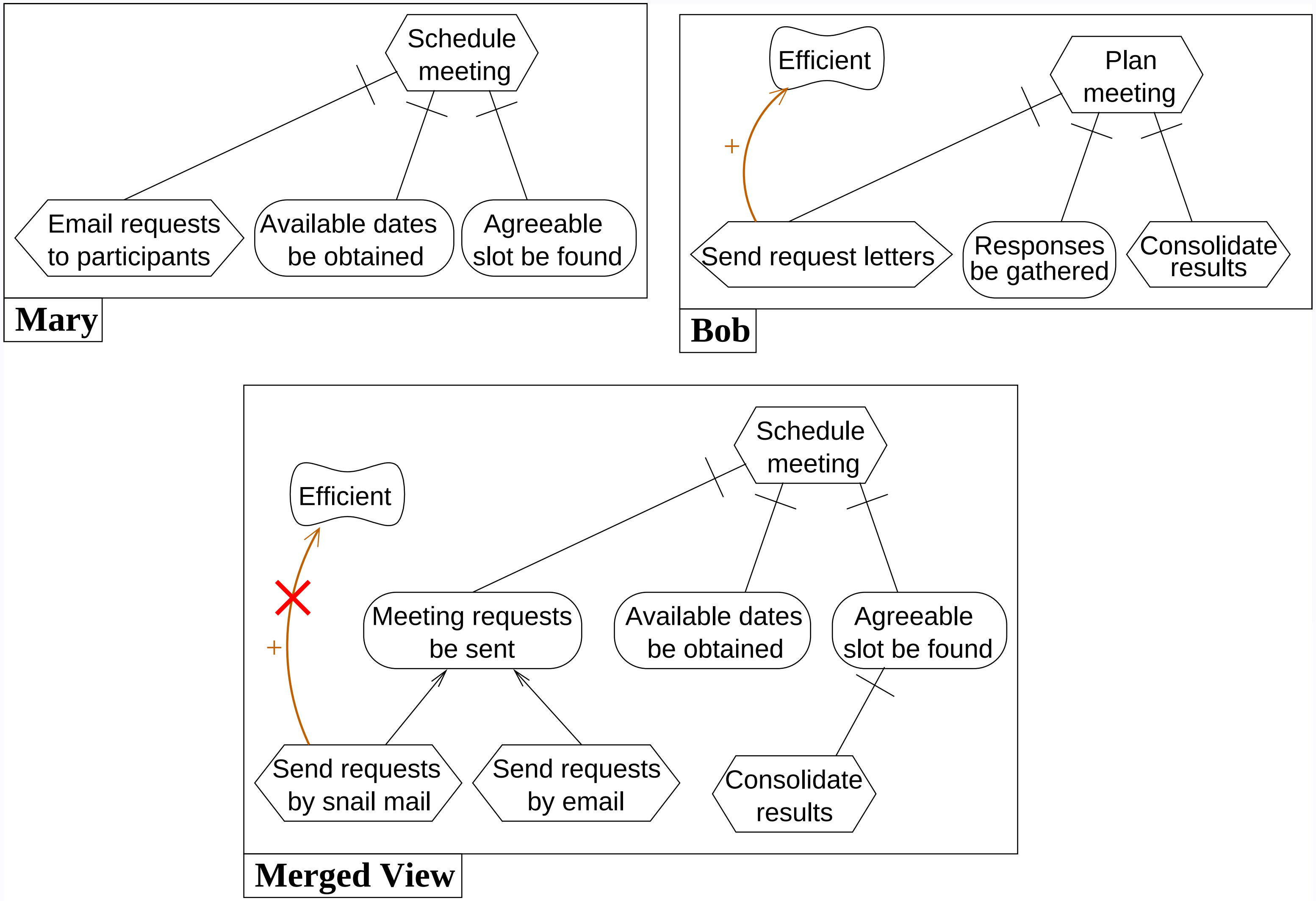
Different vocabularies

Different Aspects of the Problem



Structural differences

Different Aspects of the Problem



Conceptual differences

Different Aspects of . . . – Cnt'd

➜ **Also need to be able to:**

- . . . differentiate between the assumptions made and generated merges
- . . . record stakeholders' decisions about view elements
  - **. . . and how each decision can evolve**
- . . . keep track of how each stakeholder's vocabulary is adapted into the merged view
- . . . distinguish between the contributions of different stakeholders to the merged view

Overview of the Approach

➜ **Roadmap**

- Defining a representation formalism ($\rightarrow$ **Views**)
- Defining how a view can be mapped onto another ($\rightarrow$ **Mappings**)
- Specifying how view relationships can be hypothesized ($\rightarrow$ **Interconnection Diagrams**)
- Designing a view merging algorithm

➜ **Assumptions**

- Specification notations are graph-based
- It is possible to be precise about the areas of uncertainty and disagreement

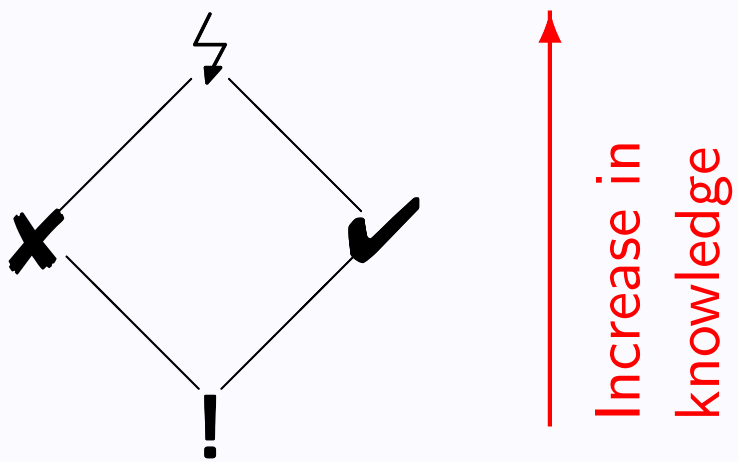
View Delineation

- **How are views represented?**
  - Each view is represented as a directed graph
    - **. . . parametrized by a meta-model**
  - View elements are *annotated*
  - An annotation . . .
    - **. . . is a value drawn from a fixed partial order**
      - Partial orders provide a flexible framework for modeling incompleteness and inconsistency [Belnap, Ginsberg]
    - **. . . describes how much knowledge is available about the element it is attached to**

View Annotations

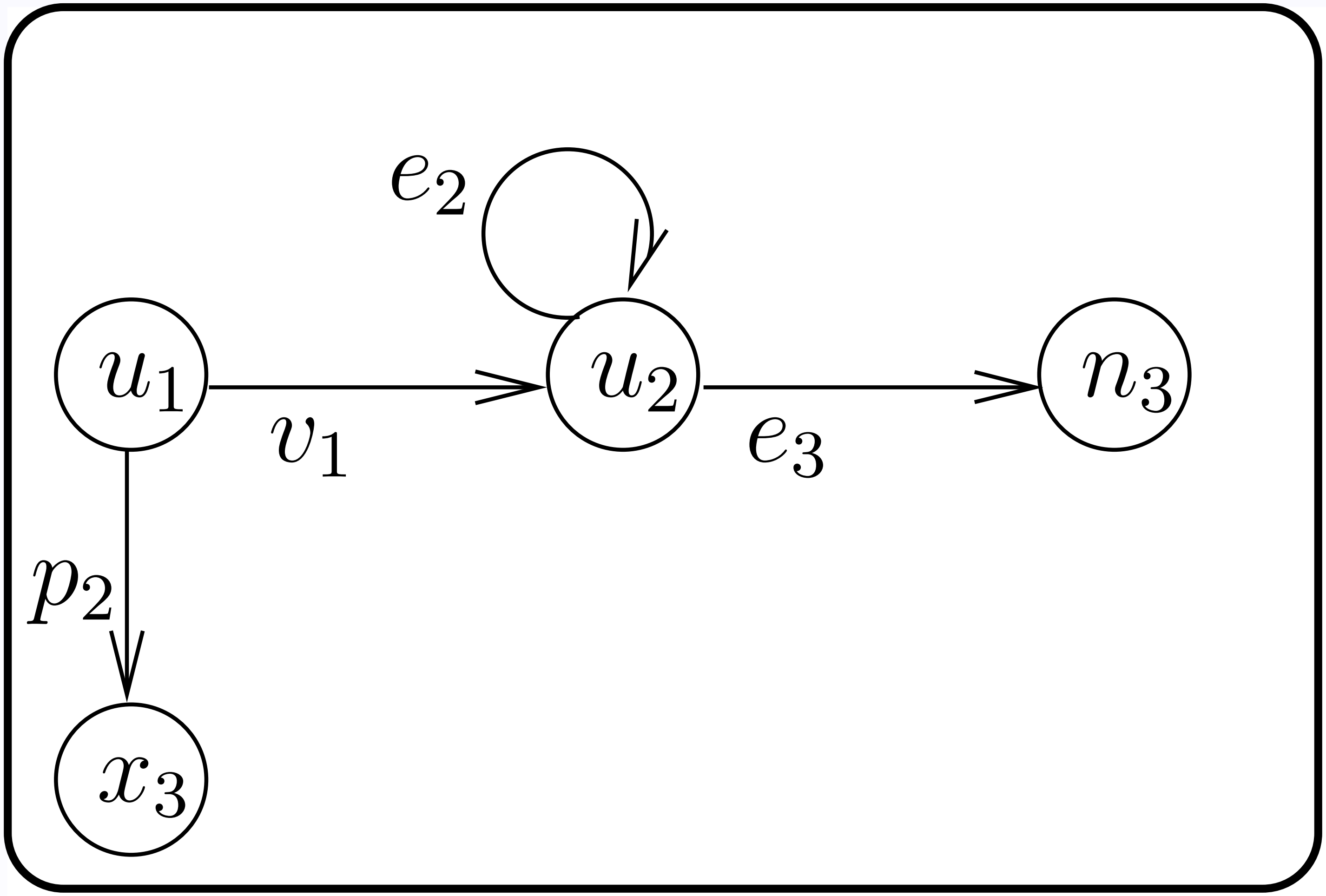
➜ We use a variant of Belnap's lattice

▷ **In principle, we can use any knowledge order (more on this later)**



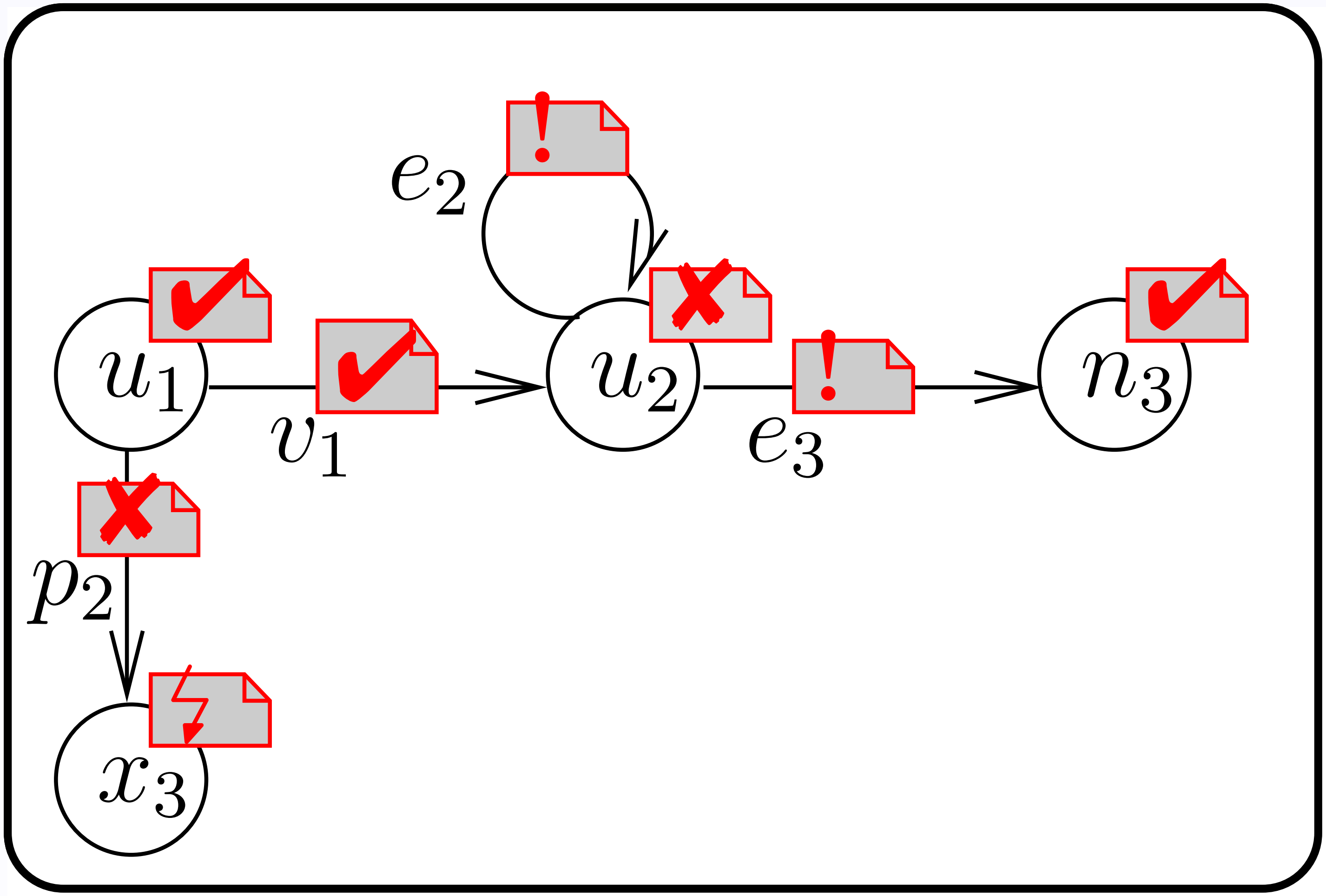
- **!**: *proposed* but not certain to be well-conceived
- ✘: known to be ill-conceived (*repudiated*)
- ✔: known to be well-conceived (*affirmed*)
- ϟ: both repudiated and affirmed, hence *disputed*

Annotations – Example



Underlying graph

Annotations – Example



... with annotations added in

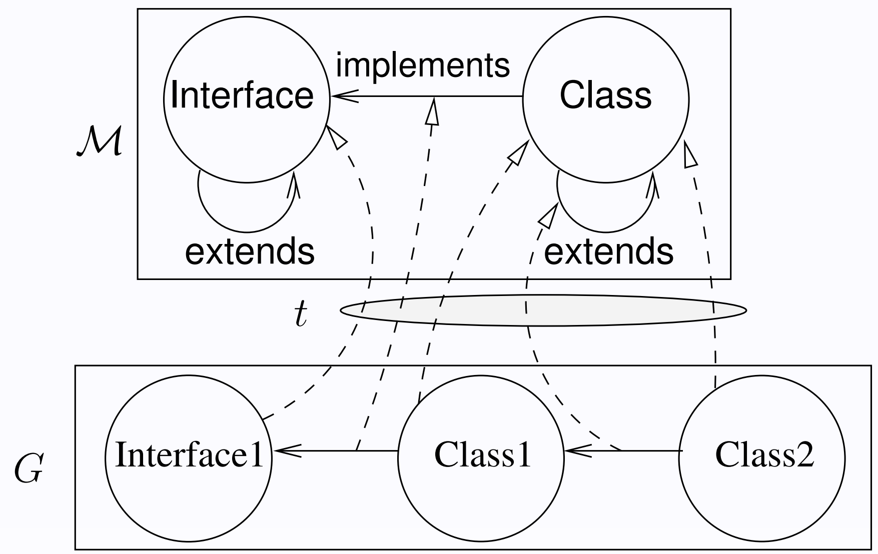
Typed Views

➜ Typing

- View elements are usually typed; e.g. in i^*:
  - **Nodes: Task, Goal, . . .**
  - **Edges: Means-Ends, Decomposition, . . .**
- Need a mechanism for assigning types to view elements
- Typing can be enforced by a meta-model
  - **Each view has a typing map to the meta-model**
  - **The meta-model itself is also represented as a directed graph**

Typing – Example

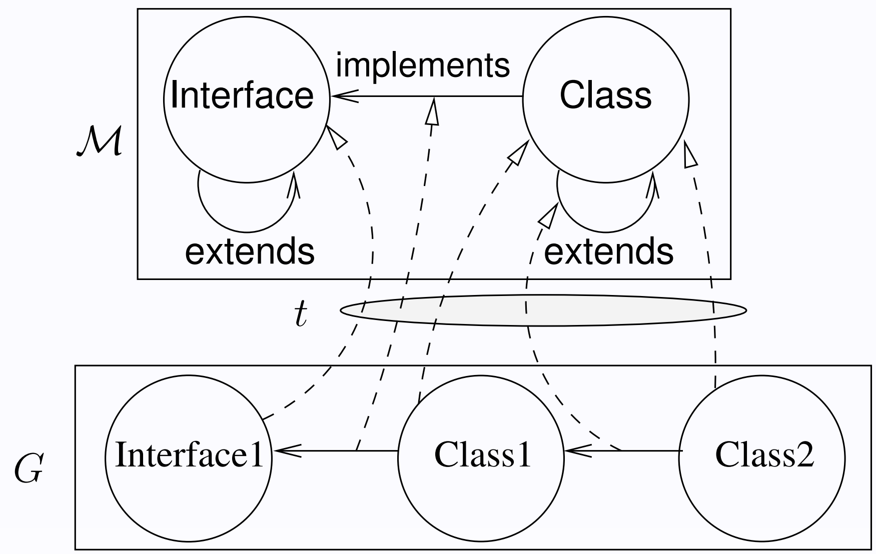
➜ Example: Class diagrams



✎ **Note:** Annotation is orthogonal to typing

Typing – Example

➜ Example: Class diagrams



✎ **Note:** Annotation is orthogonal to typing

➜ What is the meta-model for i^*?

↪ A detailed discussion can be found in the paper

View Mapping

- **Where are we?**
  - Basic constructs in the framework:
    - ✔ **Views**
    - ➔ **Mappings**
    - ☐ **Interconnection Diagrams**

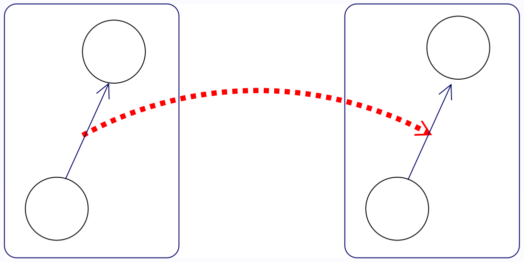
- **What is a view mapping?**
  - A mapping is a function describing how a view is *embedded* into another
    - **Mappings preserve graph structure and typing**
    - **Mappings respect view annotations: knowledge cannot decrease along a mapping**

View Mapping - Cnt'd

- **Preservation of structure:**
  - Nodes are mapped only to nodes and edges are mapped only to edges
  - If an edge e is mapped to an edge e', then the source and the target of e must be resp. mapped to those of e'

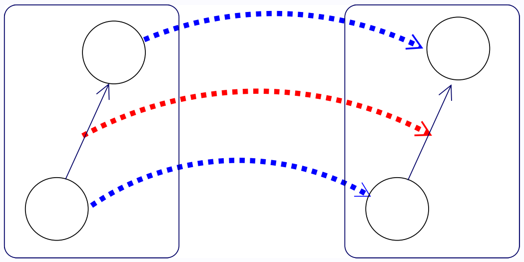
View Mapping - Cnt'd

- **Preservation of structure:**
  - Nodes are mapped only to nodes and edges are mapped only to edges
  - If an edge e is mapped to an edge e', then the source and the target of e must be resp. mapped to those of e'



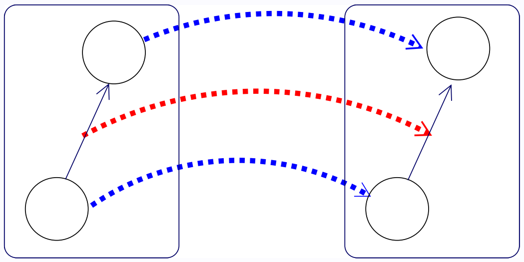
View Mapping - Cnt'd

- **Preservation of structure:**
  - Nodes are mapped only to nodes and edges are mapped only to edges
  - If an edge e is mapped to an edge e', then the source and the target of e must be resp. mapped to those of e'



View Mapping - Cnt'd

- **Preservation of structure:**
  - Nodes are mapped only to nodes and edges are mapped only to edges
  - If an edge e is mapped to an edge e', then the source and the target of e must be resp. mapped to those of e'

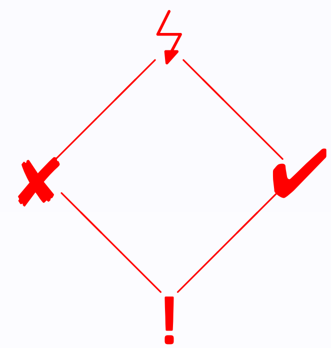


- **Preservation of types:**
  - If an element x is mapped to an element x', then the type of x and x' must be identical

View Mapping - Cnt'd

➜ **Respecting the annotations:**

- The value annotating the image of an element must be *at least as specific as* the value annotating the element
  - ... **i.e. knowledge cannot be lost along a mapping**
- **Examples**:
  - ✎ **Recall the annotation lattice:**



  - **a ! can remain as is, or evolve to any other value;**
  - **a ✔ can remain as is, or evolve to ↯ <u>only</u>;**
  - **a ✘ can remain as is, or evolve to ↯ <u>only</u>**

Capturing View Interrelations

➜ **Where are we?**

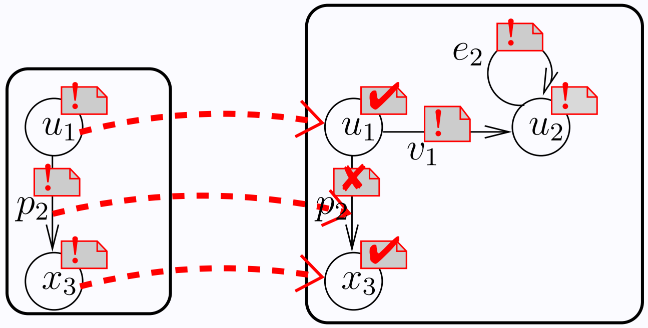
- Basic constructs in the framework:
  - ✔ **Views**
  - ✔ **Mappings**
  - ➔ **Interconnection Diagrams**

➜ **What is an interconnection diagram?**

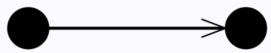
- An interconnection diagram is a graph whose nodes are views and whose edges are mappings
- . . . and describes a hypothesis about relationships between views

View Interrelations – Examples

➜ Evolution

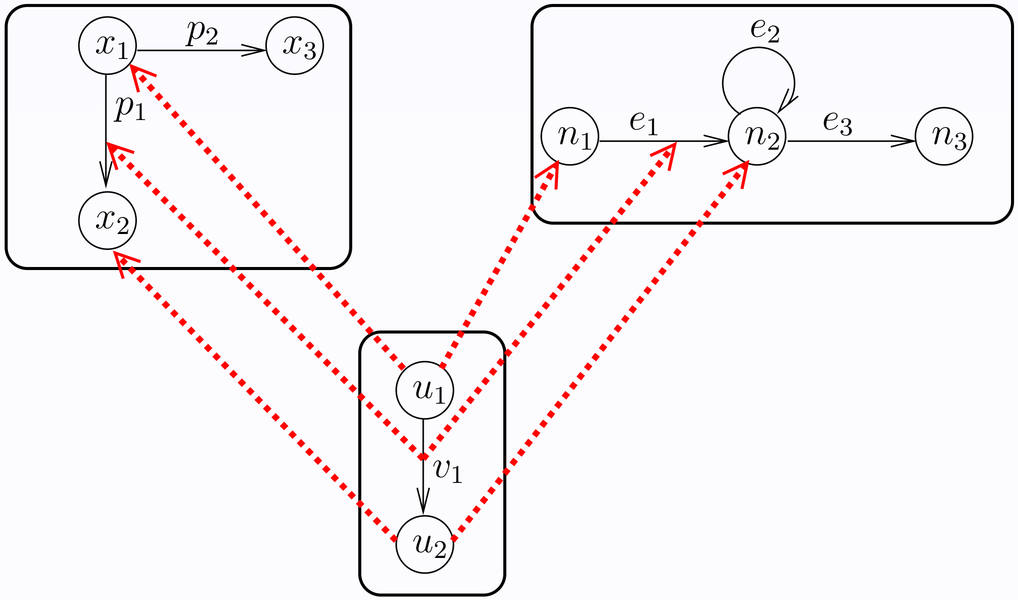


Pattern:

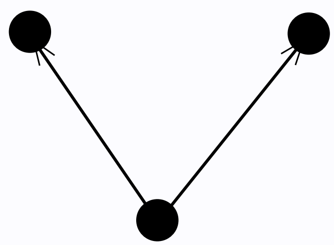


➜ Correspondence identification

▷ Annotations have been omitted for simplicity



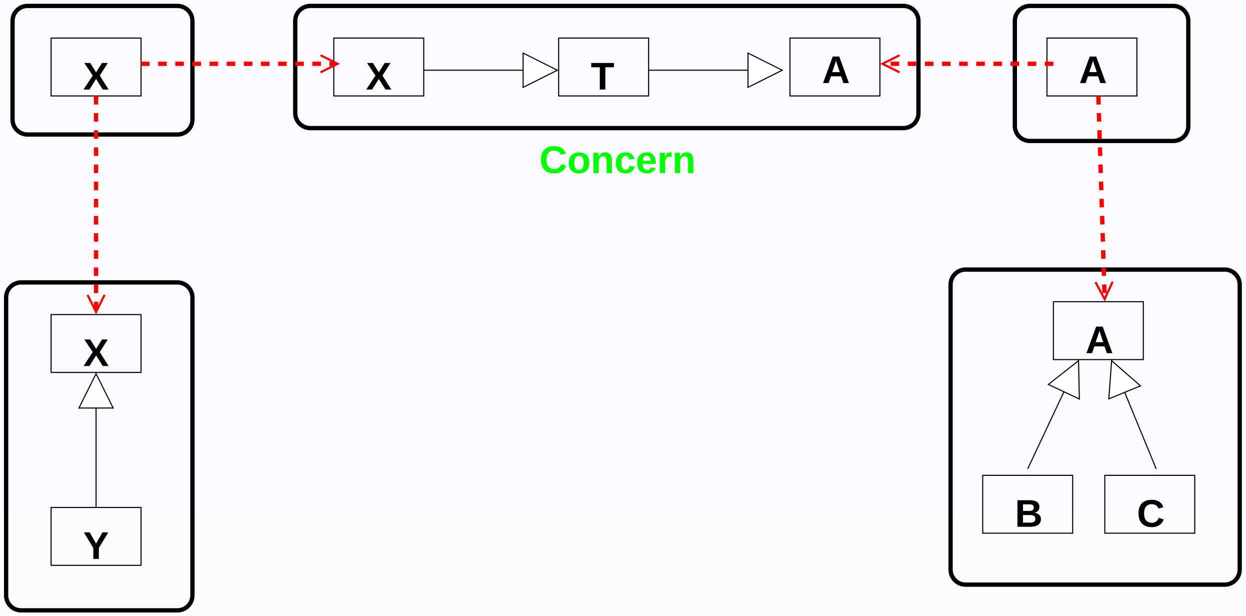
Pattern:



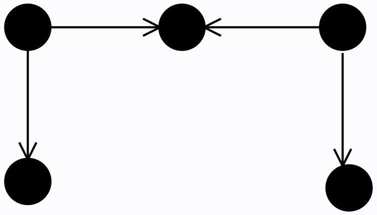
Examples – Cnt'd

➜ Incorporating a concern

▷ Annotations have been omitted for simplicity



Pattern:



View Merging

➜ **Where are we?**

✔ Basic constructs:

- **Views, Mappings, Interconnection Diagrams**

➜ View merging

➜ **The merge process**

- **Input**: An interconnection diagram D
- **Output**: A merged view combining all the views in D w.r.t. to their relationships as described by the mappings in D
  - **. . . we also get a family of mappings showing how each view in D gets adapted into the merged view**

The Merge Algorithm

➜ **Intuition:**

1. Assume all view elements are distinct
2. Unify elements deemed equal by the interconnections

✎ View nodes and edges are merged component-wise

➪ . . . **hence, we only need an algorithm for merging annotated sets**

The Merge Algorithm – Cnt'd

➜ **Merging annotated sets:**

1. Disregard the annotations
2. Merge the resulting plain sets
3. Compute an annot. for each merged set element

➜ **Merging sets:**

1. Get the disjoint union of all sets
   ➥ **Treat the result as an initially discrete graph G**
2. Connect pairs of G-nodes related by a mapping
3. Find the connected components of G
   ➥ **. . . each connected component is an element of the merged set**

The Merge Algorithm – Cnt'd

➜ Set merging – an example:

$$A = \{x, y, w\} \qquad B = \{x, y, t\}$$

$$C = \{z, w\}$$

The given interconnection diagram

The Merge Algorithm – Cnt'd

➜ Set merging – an example:

$$A = \{x, y, w\} \qquad B = \{x, y, t\}$$

$$C = \{z, w\}$$

w_C

x_A y_A w_A x_B y_B t_B

z_C

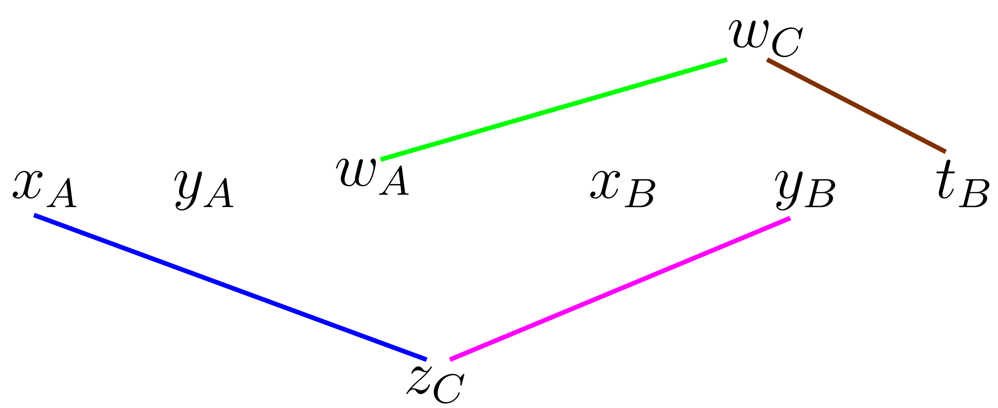
Computing the disjoint union

The Merge Algorithm – Cnt'd

➜ **Set merging – an example:**

$$A = \{x, y, w\} \qquad B = \{x, y, t\}$$

$$C = \{z, w\}$$



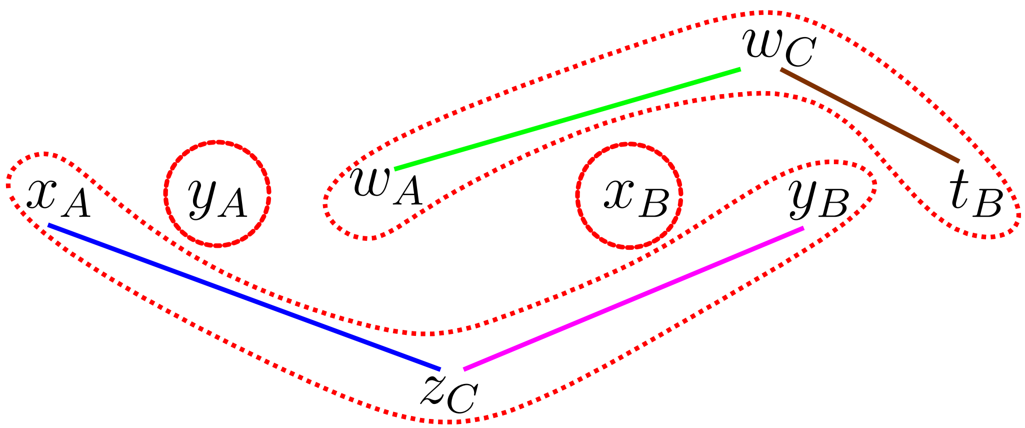
Connecting up the related pairs

The Merge Algorithm – Cnt'd

➜ **Set merging – an example:**

$$A = \{x, y, w\} \qquad B = \{x, y, t\}$$

$$C = \{z, w\}$$



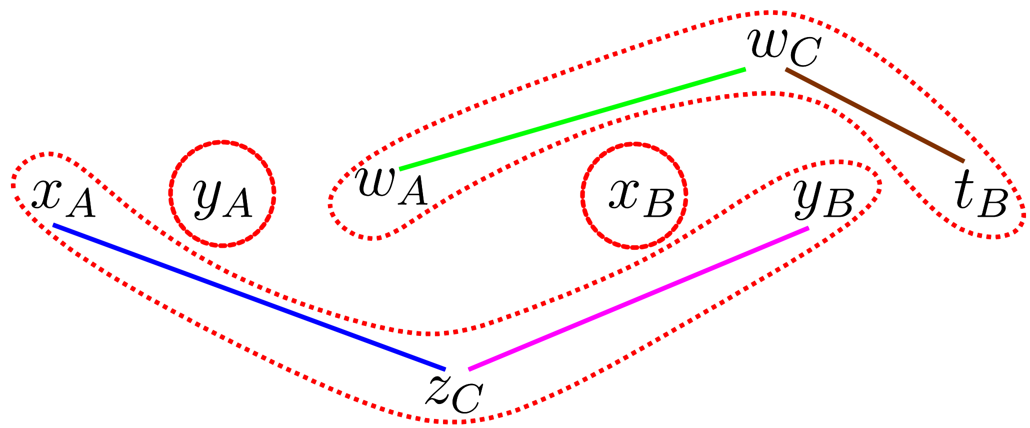
Finding the connected components

The Merge Algorithm – Cnt'd

➜ **Set merging – an example:**

$$A = \{x, y, w\} \qquad B = \{x, y, t\}$$

$$C = \{z, w\}$$



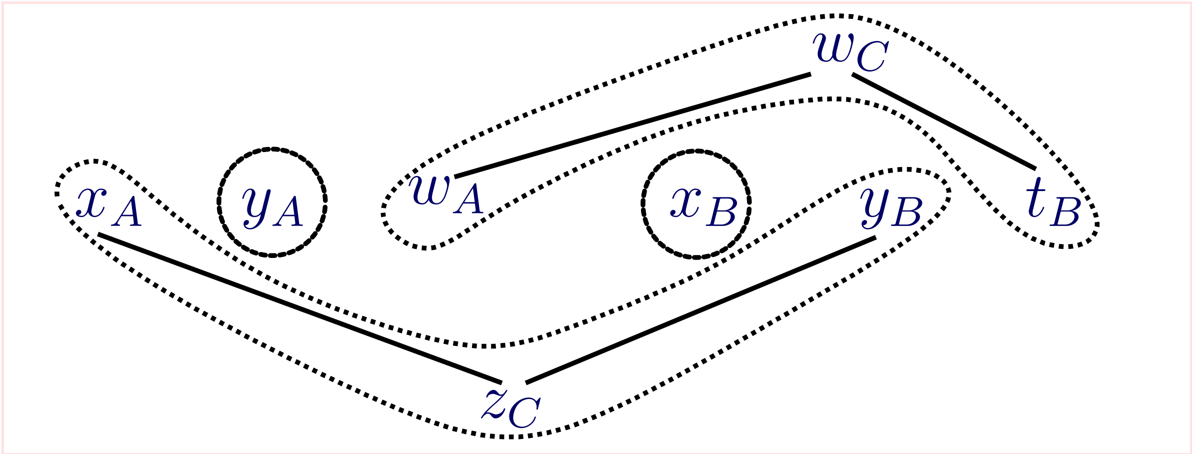
$$\{\underline{\{x_A, y_B, z_C\}}, \underline{\{y_A\}}, \underline{\{w_A, t_B, w_C\}}, \underline{\{x_B\}}\}$$

The merged set

The Merge Algorithm – Cnt'd

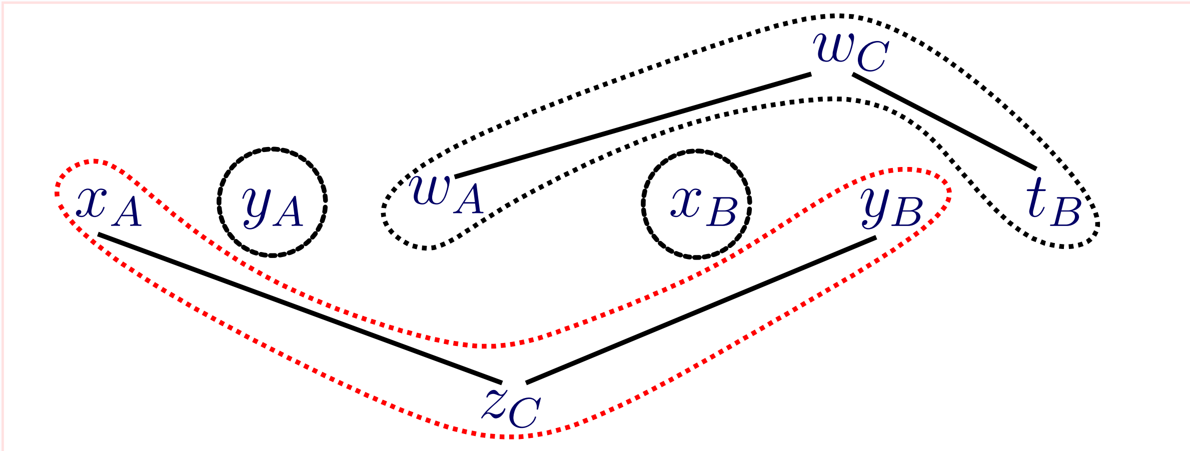
➜ **Computing the annotations:**

- Take the *least upper bound* of the annotations of the elements in each connected component
  - **l.u.b. denotes the least specific (and yet admissible) knowledge degree**
  - **. . . for the l.u.b. to exist for all cases, we require that the annotation poset be a *complete lattice***
- **Example**:



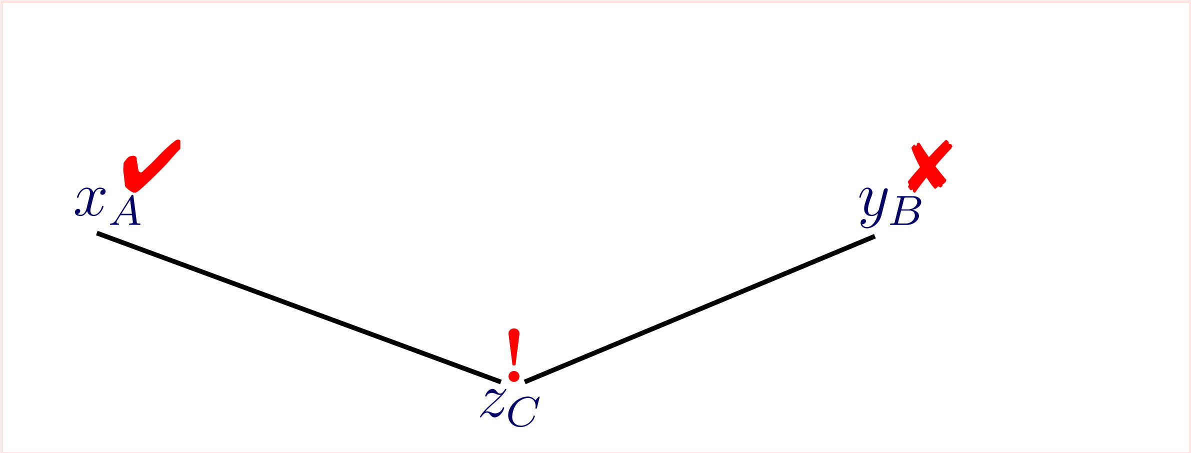
The Merge Algorithm – Cnt'd

- **Computing the annotations:**
  - Take the *least upper bound* of the annotations of the elements in each connected component
    - **l.u.b. denotes the least specific (and yet admissible) knowledge degree**
    - **. . . for the l.u.b. to exist for all cases, we require that the annotation poset be a *complete lattice***
  - **Example**:



The Merge Algorithm – Cnt'd

- **Computing the annotations:**
  - Take the *least upper bound* of the annotations of the elements in each connected component
    - **l.u.b. denotes the least specific (and yet admissible) knowledge degree**
    - **. . . for the l.u.b. to exist for all cases, we require that the annotation poset be a *complete lattice***
  - **Example**:

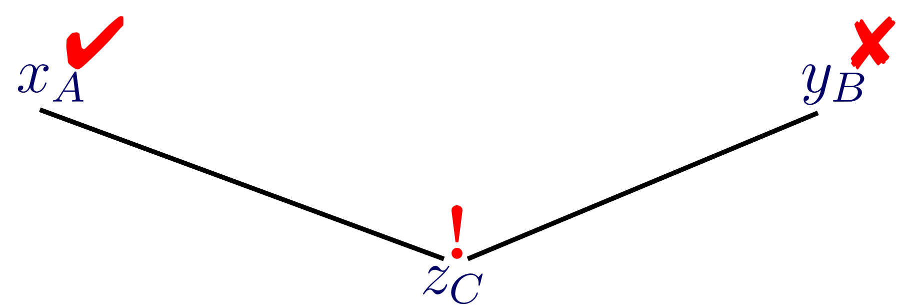


The Merge Algorithm – Cnt'd

➜ **Computing the annotations:**

- Take the *least upper bound* of the annotations of the elements in each connected component
  - **l.u.b. denotes the least specific (and yet admissible) knowledge degree**
  - **. . . for the l.u.b. to exist for all cases, we require that the annotation poset be a** *complete lattice*
- **Example**:

$$\mathrm{lub}\{!, ✔, ✘\} = ↯$$

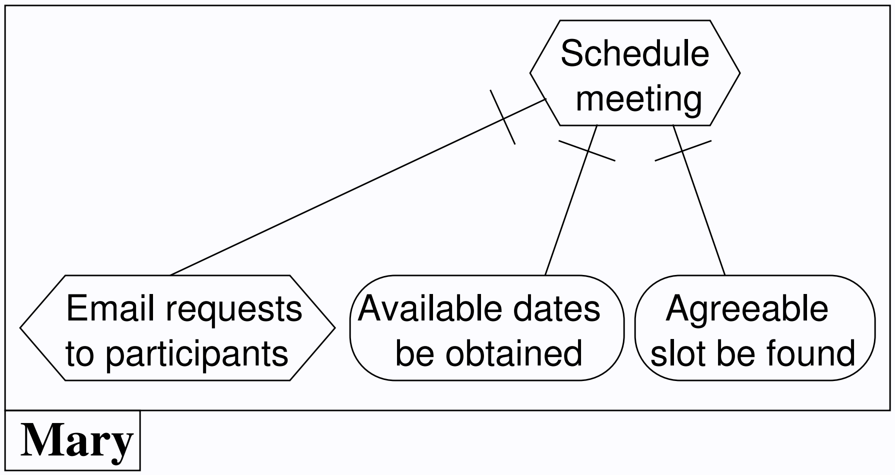


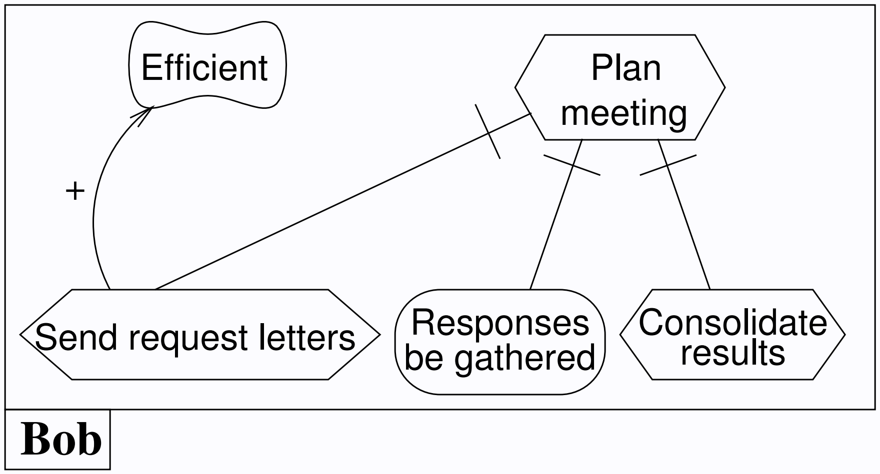
Example – Revisited

➜ Where are we?

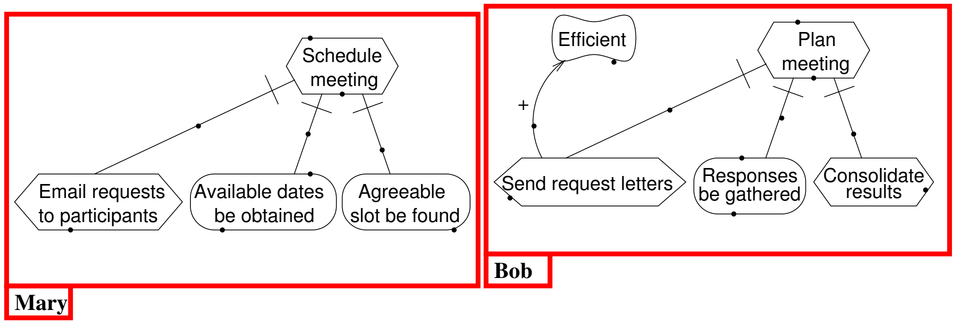
- ✔ Basic constructs: Views, Mappings, Diagrams
- ✔ The merge algorithm
- ➜ Merging i^* views – an example

➜ Initial views of stakeholders



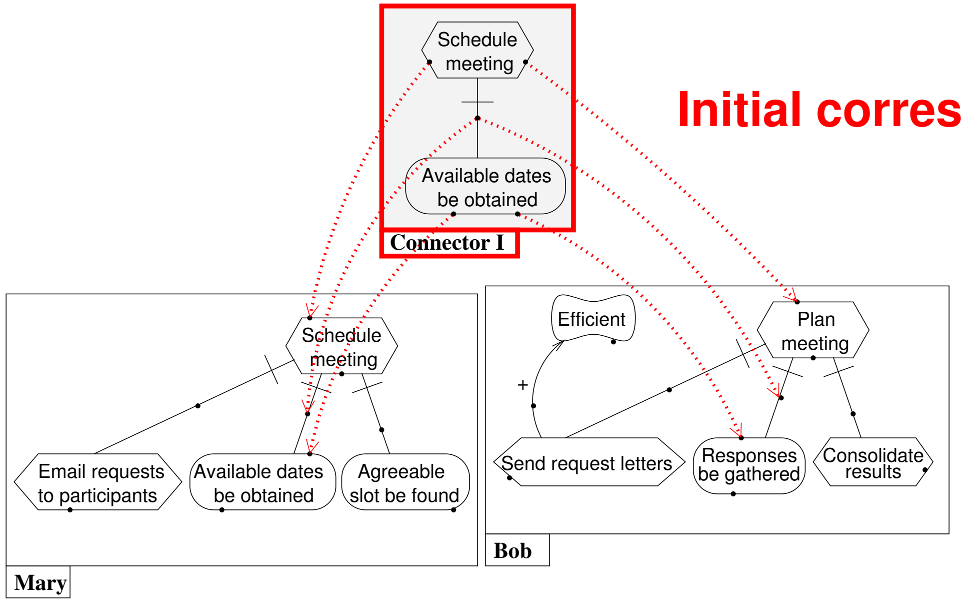


View Interconnections



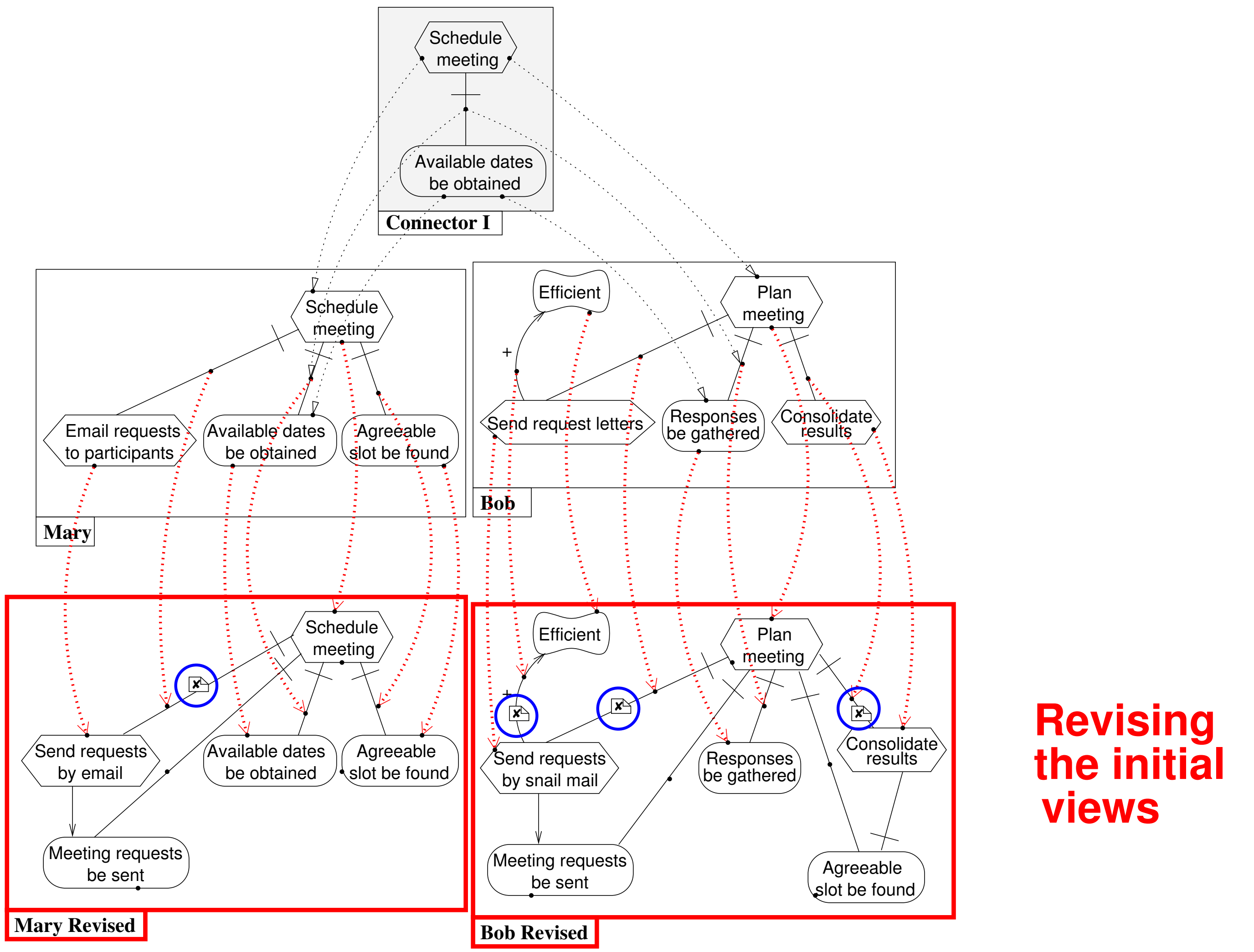
Initial views

View Interconnections



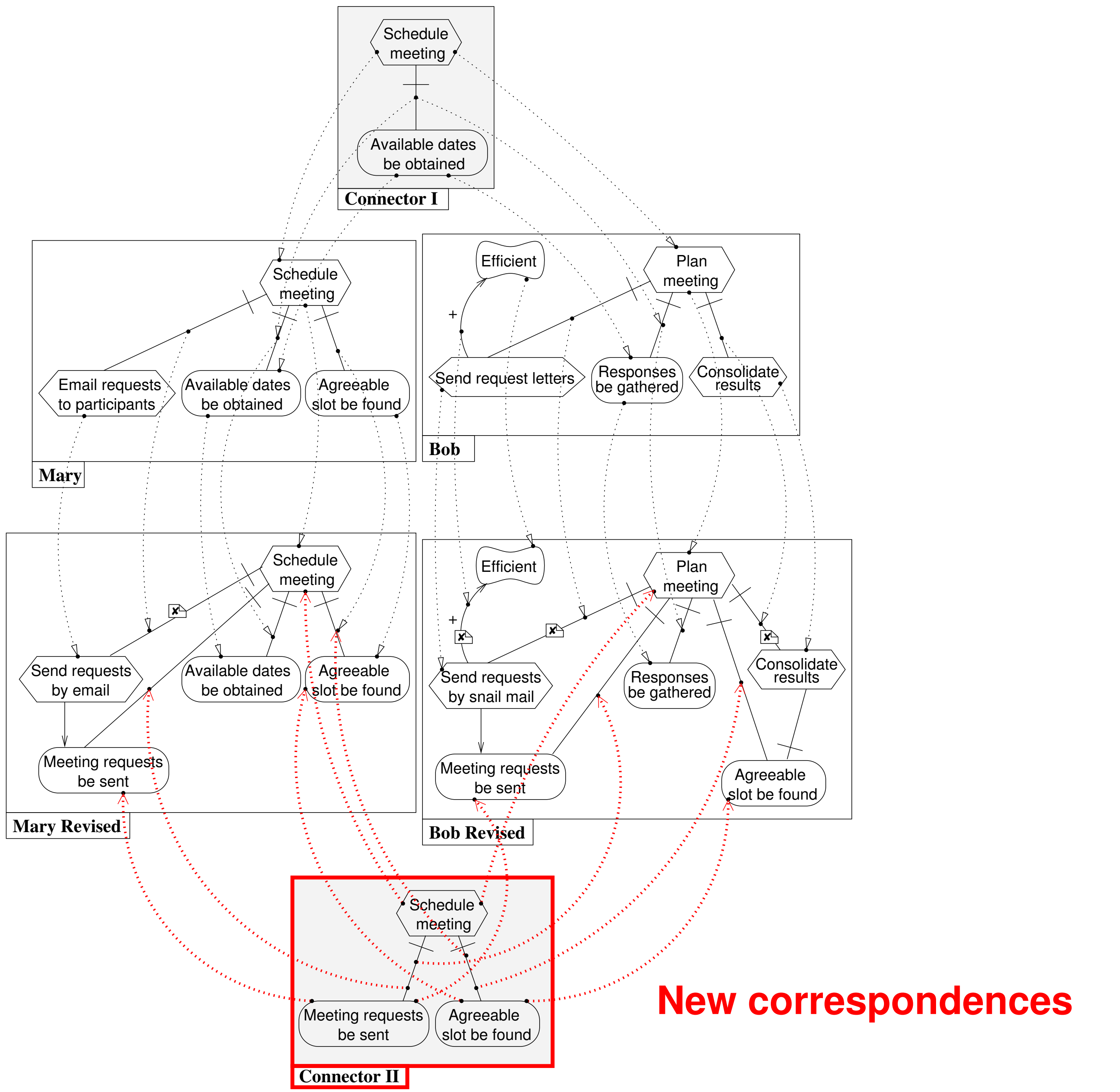
Initial correspondences

View Interconnections



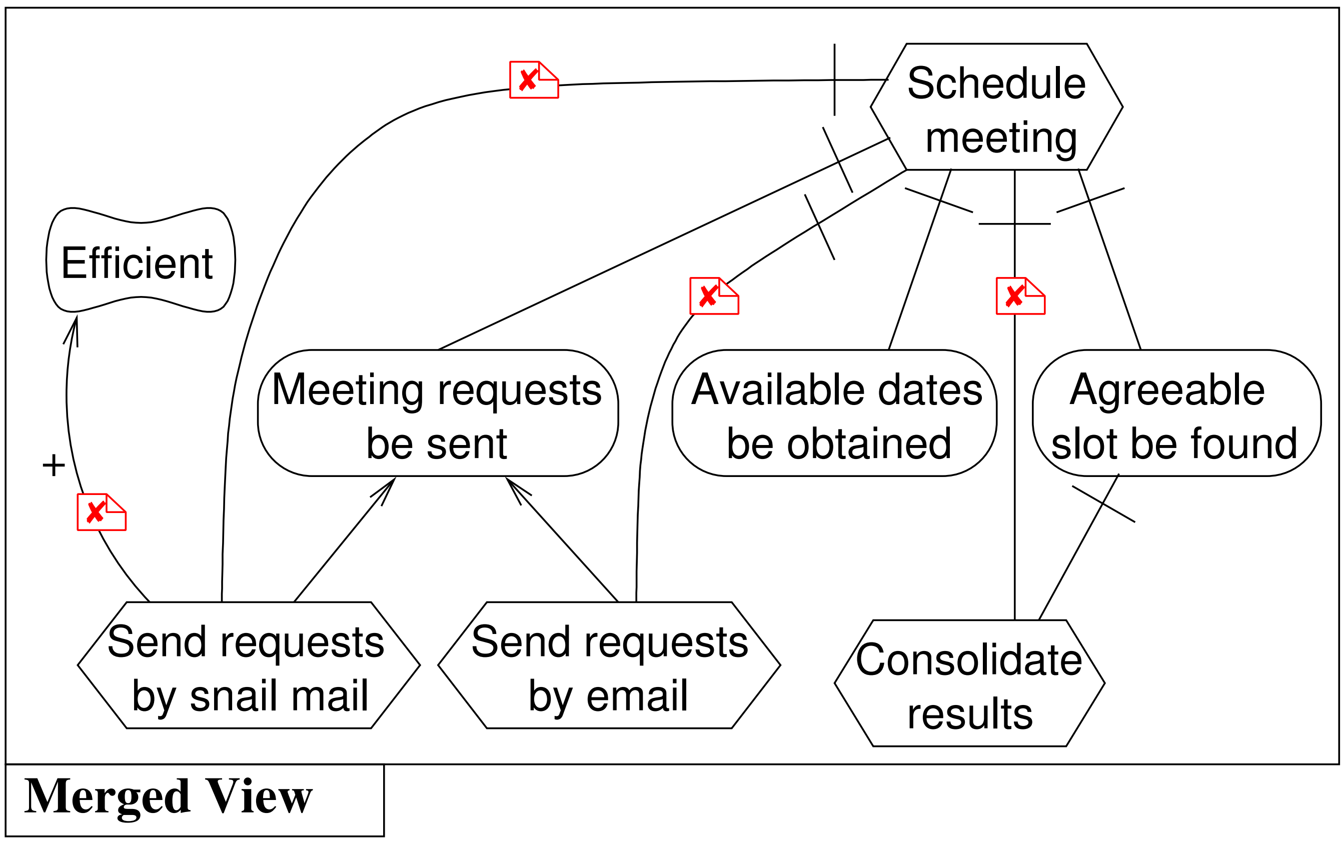
Revising the initial views

View Interconnections



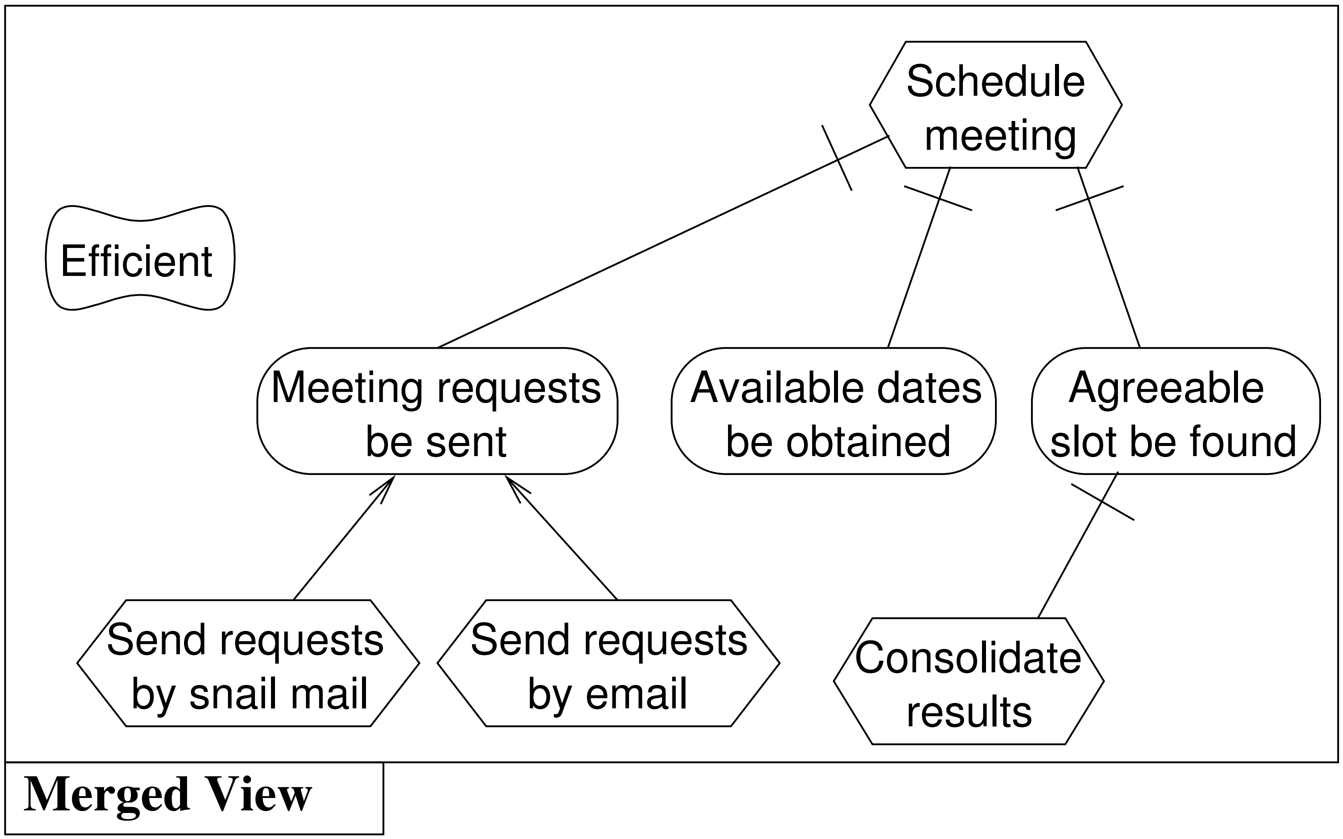
New correspondences

The Merged View



Result of the merge operation

The Merged View



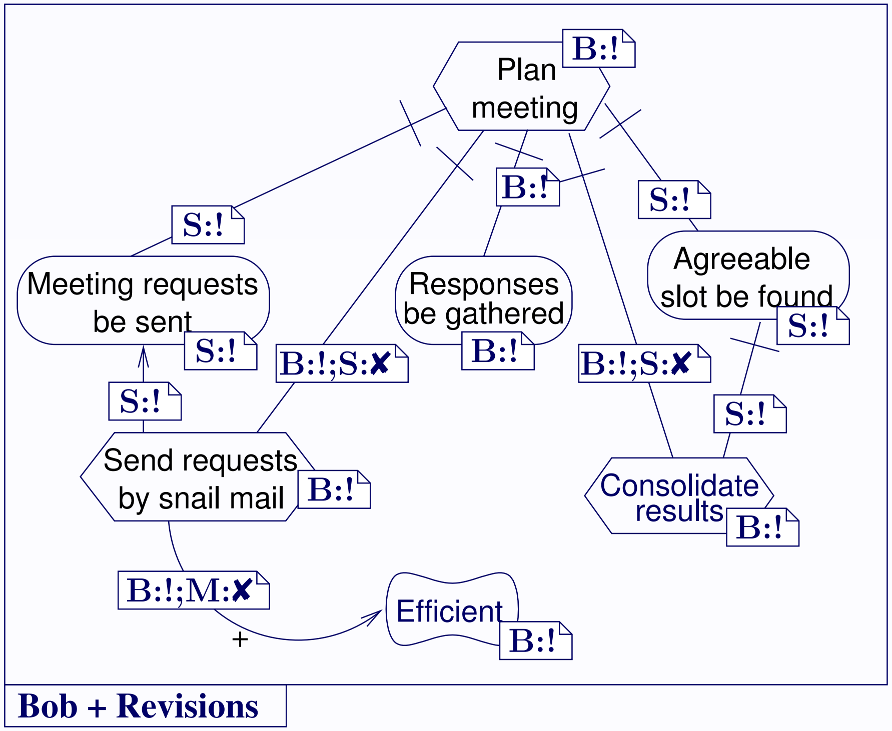
... with repudiated elements removed

Trouble in Paradise!

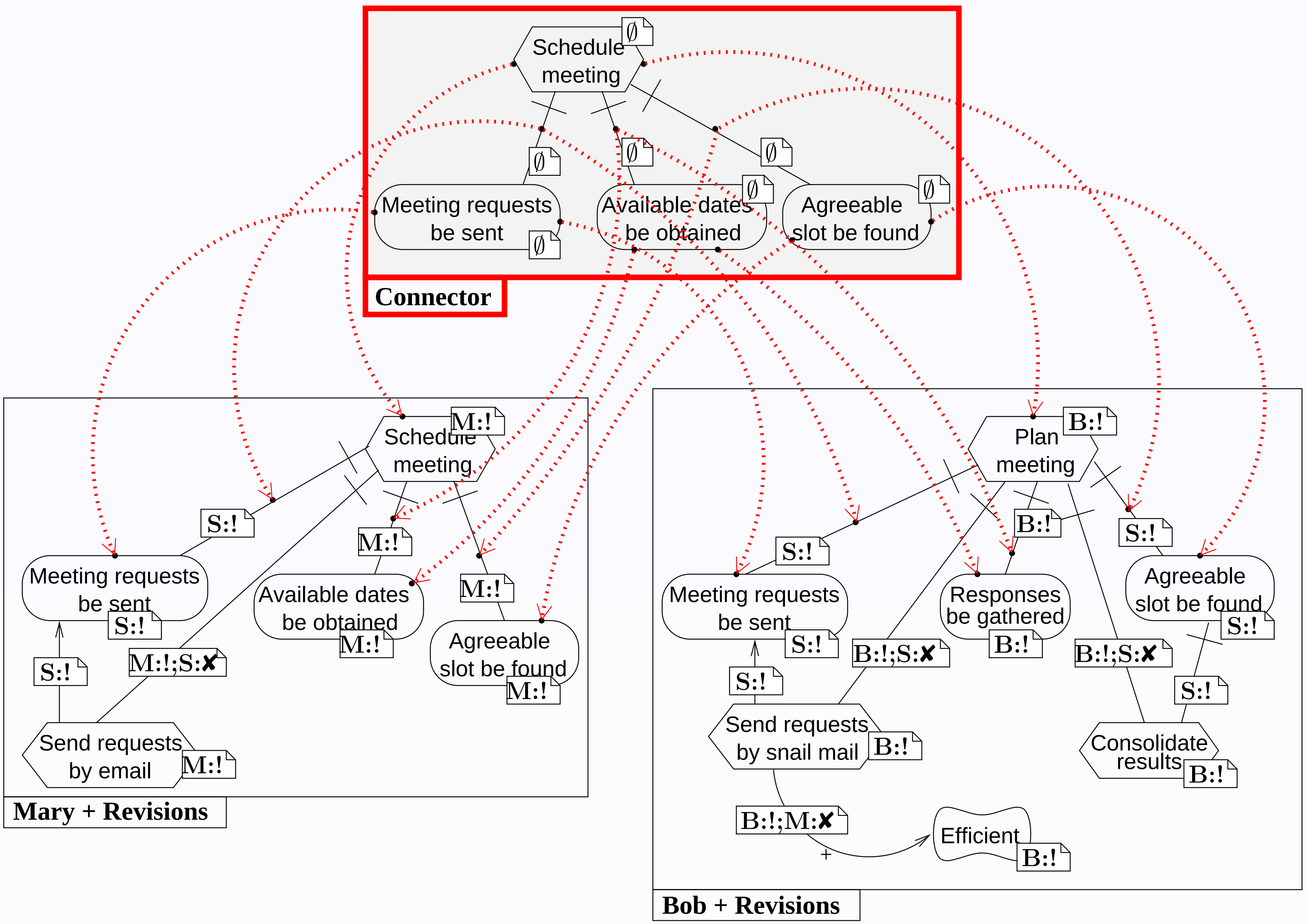
- **Problems with the annotation scheme**
  - Merges do not reflect the decisions of individual stakeholders
    - **. . . hence, we cannot differentiate between the conceptual contributions of different parties**
      - . . . what if we want to attach a priority or credibility factor to each stakeholder?
  - Direct manipulation of already-existing views may not be possible due to traceability reasons
- **Solution:**
  - Using a more elaborate annotation scheme . . .

Stakeholder Traceability

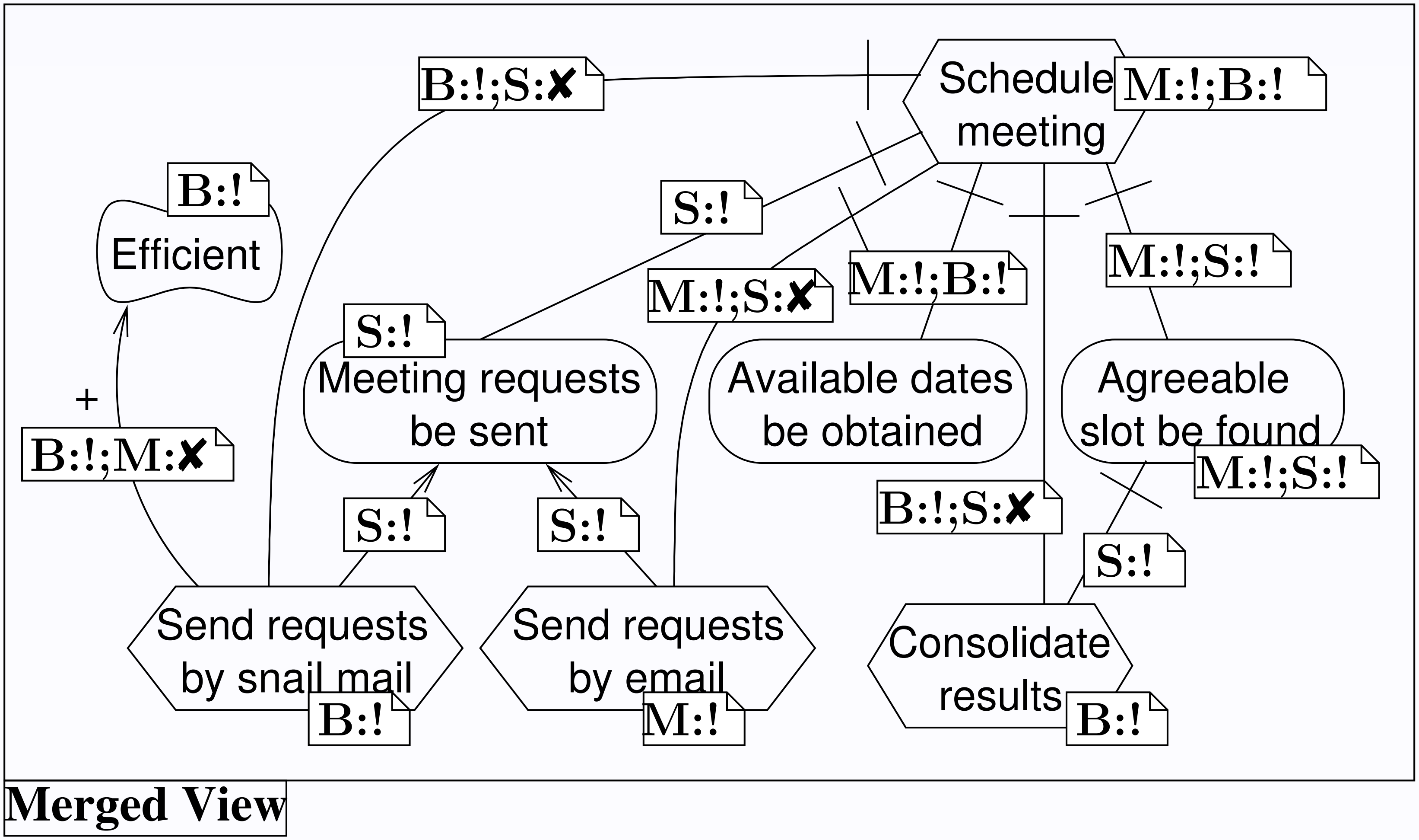
- **New annotation scheme:**
  - Attaching multiple annotations to view elements
    - **. . . one value for each involved stakeholder**
    - **View mappings must respect the annotations for every individual stakeholder**
  - **Example**: Sam and Mary revising Bob's view



New Interconnections



The New Merged View



The annotation of each element reflects the conceptual contributions made to it by the involved stakeholders

Recap

- **Summary**
  - **A formal framework for merging incomplete and inconsistent views**
    - **independent of any particular modeling notation**
    - **customizable to many graph-based notations**
    - **supports explicit mapping of views**
      - . . . hence, supports exploration of alternatives
  - **Limitations and weaknesses**
    - **based purely on syntactic mappings**
    - **still not clear how the identification of interconnections can be automated**

Related Work

➜ Requirements Engineering:

- [Easterbrook–Chechik] proposed using lattices for modeling inconsistency in views
  - ... **focuses on support for model-checking**
  - ... **is tailored to 3-way merges**
- [Ehrig et al] use category theory for consolidating viewpoints
  - ... **focuses on graph transformation systems**
  - ... **does not support inconsistency**
- [Uchitel–Chechik] have proposed a framework for merging partial transition systems
  - ... **focuses on preservation of behavior**
  - ... **views cannot be merged if they are inconsistent**
  - ... **relationships are hypothesized via (bi)similarity**

Related Work – Cnt'd

- **Databases:**
  - Numerous approaches to schema merging have been proposed:
    - [Buneman et al, Pottinger–Bernstein, Melnik] to name a few
    - None of the above support incompleteness and inconsistency
    - And most are tailored to the 3-way-merge pattern
    - Also not clear how they generalize to notations other than ER diagrams

Future Work

- **Venues for future work**
  - Add model-based semantics for various modeling languages
    - **likely targets: class diagrams in OOD; relational schema in DB design**
  - Explore treatment of heterogeneous merges
    - **example: reason over multiple notations of UML**
  - Automating the identification of interconnections
  - Investigating how the framework can facilitate negotiation during requirements analysis

Thank You!

Questions?