

Requirements Engineering II – Assignment 2

Requirements Engineering with Viewpoints

Arthur T.
Jeton M.
Michael S.
Rita P.
Robert R.

Relation to the session's topic

- Conflicts and inconsistencies in requirements occur very often.
- Different participants: different views and different knowledge.
- → How to identify, document and handle conflicts and inconsistencies?

Literature

- Using ViewPoints for inconsistency management
[Steven Easterbrook, Bashar Nuseibeh, 1996]
- View merging in the presence of incompleteness and inconsistency
[Mehrdad Sabetzadeh, Steve Easterbrook, 2006]

Central topics

- How to :
 - allow co-evolution of multiple viewpoints?
 - connect viewpoints?
 - trace changes in viewpoints?
 - detect inconsistencies?
 - manage and trace inconsistency?

Inconsistence in viewpoints

- Why is allowing inconsistencies (temporally) to exist desirable ?
 - Stimulus for eliciting missing information
 - Don't force change without necessary information's
 - Don't prevent further work

Using ViewPoints for inconsistency management

[Steven Easterbrook, Bashar Nuseibeh, 1996]

Central statements

- In evolving specifications often multiple inconsistent views on system
- Detecting and resolving only a part of solution
- Resolved inconsistency may not stay resolved
- Good inconsistency management leads to more robust specifications
- Inconsistencies provide important clues about missing information
- It is often desirable to tolerate and even encourage inconsistency

ViewPoints framework

- Paper describes general framework for detecting inconsistencies
- Based on formal consistency rules
- Consistency checking separated from resolution
- ViewPoints responsible for checking their own constraints
- ViewPoints have a work records

Example (1)

A (owner=Anne; domain=telephone/calling)

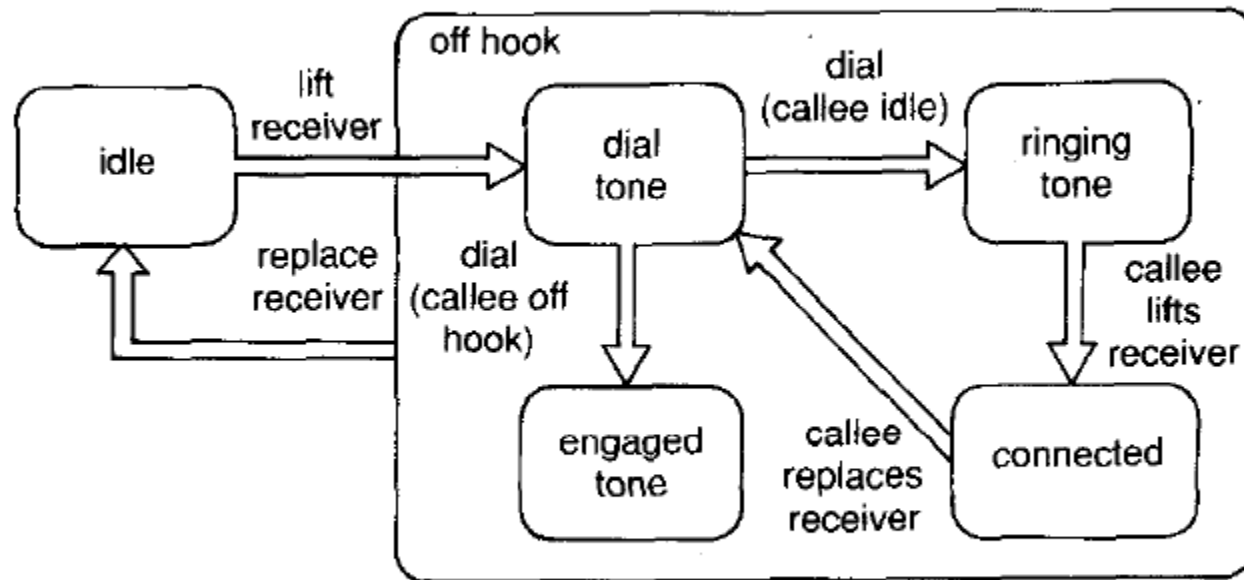


Fig. 1 Anne's initial ViewPoint specification for making a call

Example (2)

B (owner=Bob; domain=telephone/incoming call)

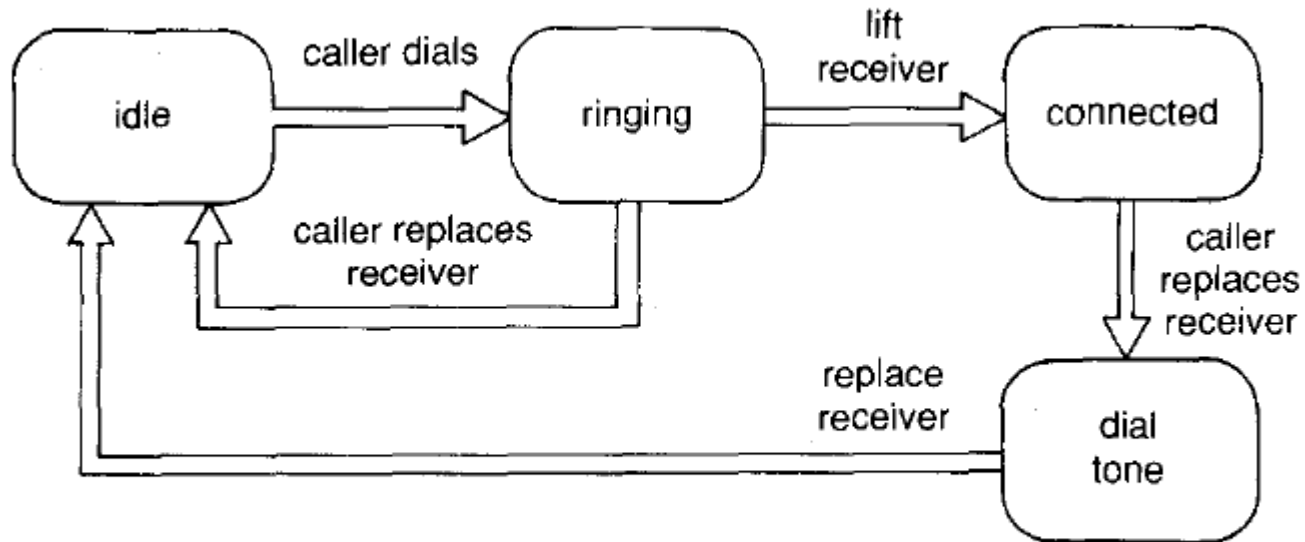


Fig. 2 Bob's initial ViewPoint specification for receiving a call

Example (3)

$$R_1 : \forall VP_D(STD, D_S) \\ \{VP_S.transition(X, Y) \wedge VP_D.state(X) \wedge VP_D. \\ state(Y) \rightarrow VP_D.transition(X, Y)\}$$

Formal description of a rule

$$\{\} \Rightarrow [VP_S, R_1] \\ \{\mathcal{R}_1(transition(X, Y), VP_D.transition(X, Y))\} \cup \\ \{missing(transition(X, Y), VP_D.transition(X, Y), R_1)\}$$

Precondition and postconditions

$$missing(transition(off\ hook, idle), \\ B.transition(connected, idle), R_1)$$

Result of applying rule 1

Example (4)

B (owner=Bob; domain=telephone/incoming call)

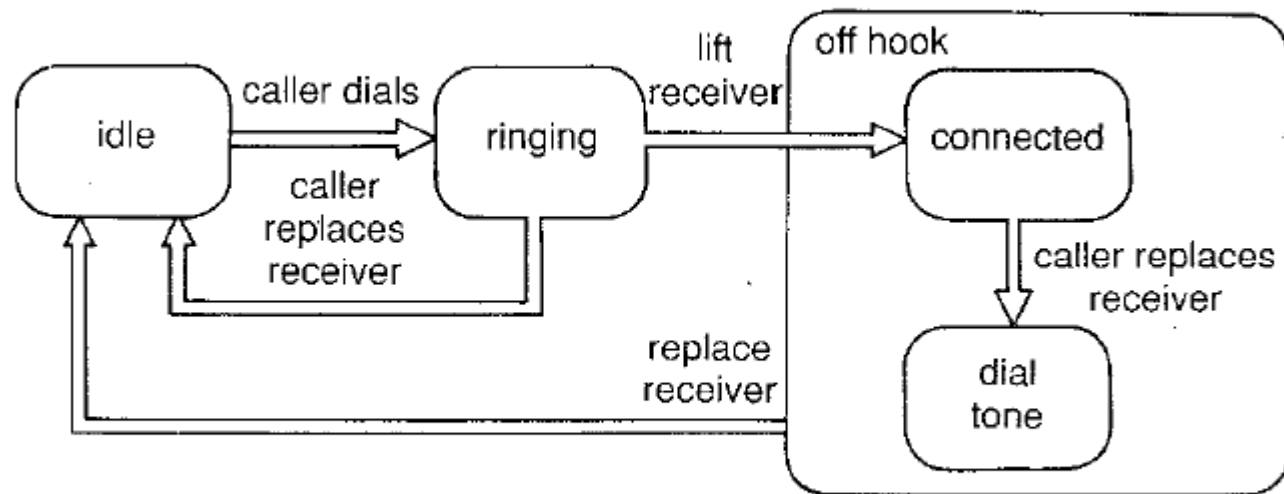


Fig. 3 Bob's ViewPoint specification after resolution

View merging in the presence of incompleteness and inconsistency

[Mehrdad Sabetzadeh, Steve Easterbrook, 2006]

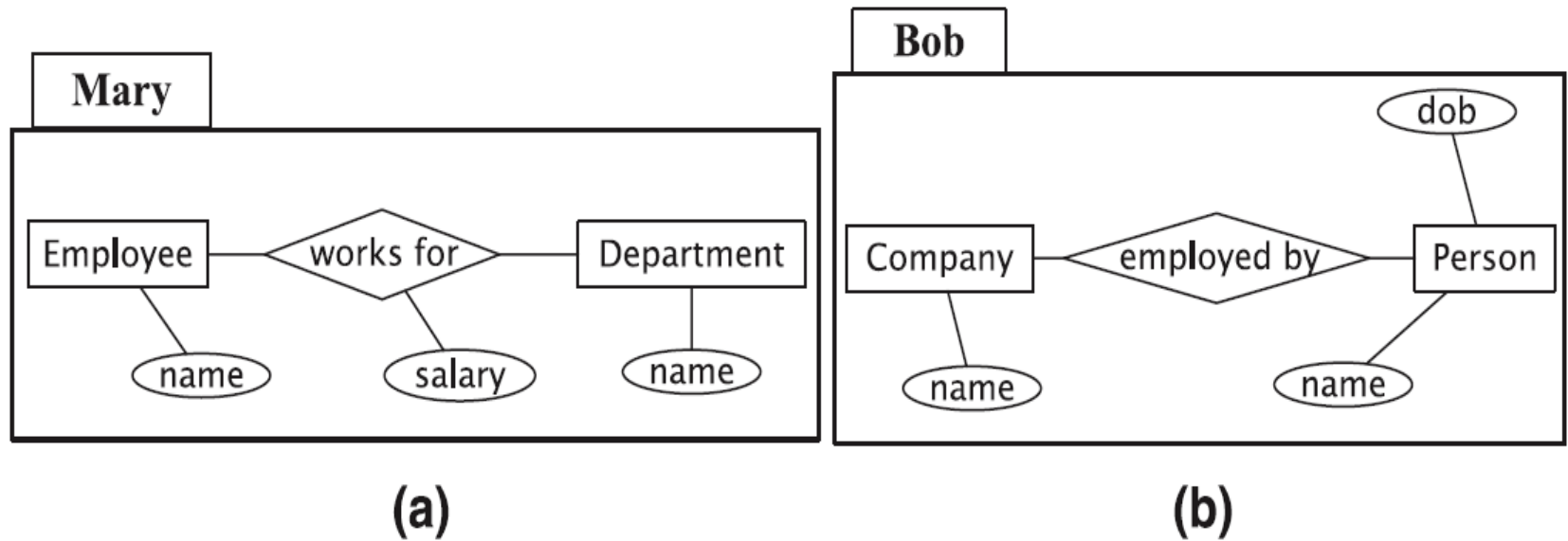
Summary

- Models play a key role in many aspects of requirements analysis and design
- For complex systems: models constructed and manipulated by distributed teams
- → building a consistent & unified view major challenge
- Views evolve over time
- → most of the time the current set of views are likely to be incomplete and inconsistent

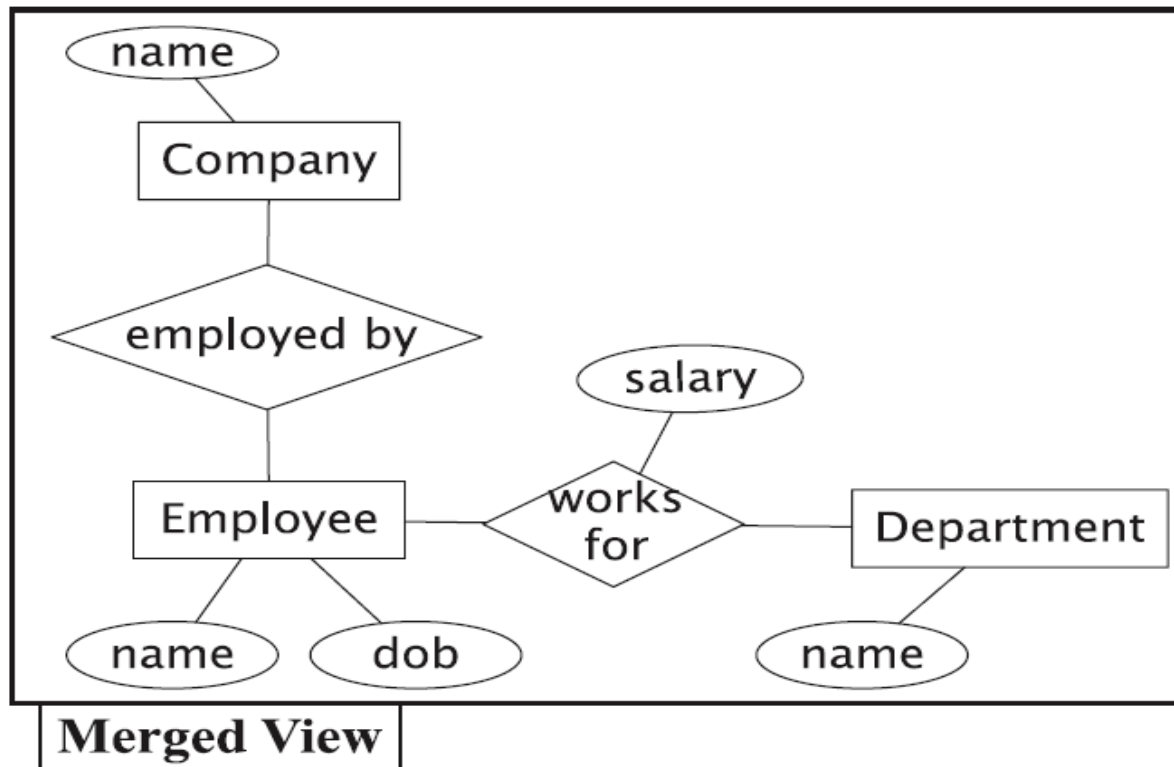
Summary (2)

- Framework for merging multiple views
 - tolerates inconsistency between views
- Adaptable to any graph-based modeling language
- Use of unification graph for connecting views
- Use of unification graph to provide traceability
- Use of annotated graphs for modelling incompleteness and inconsistency

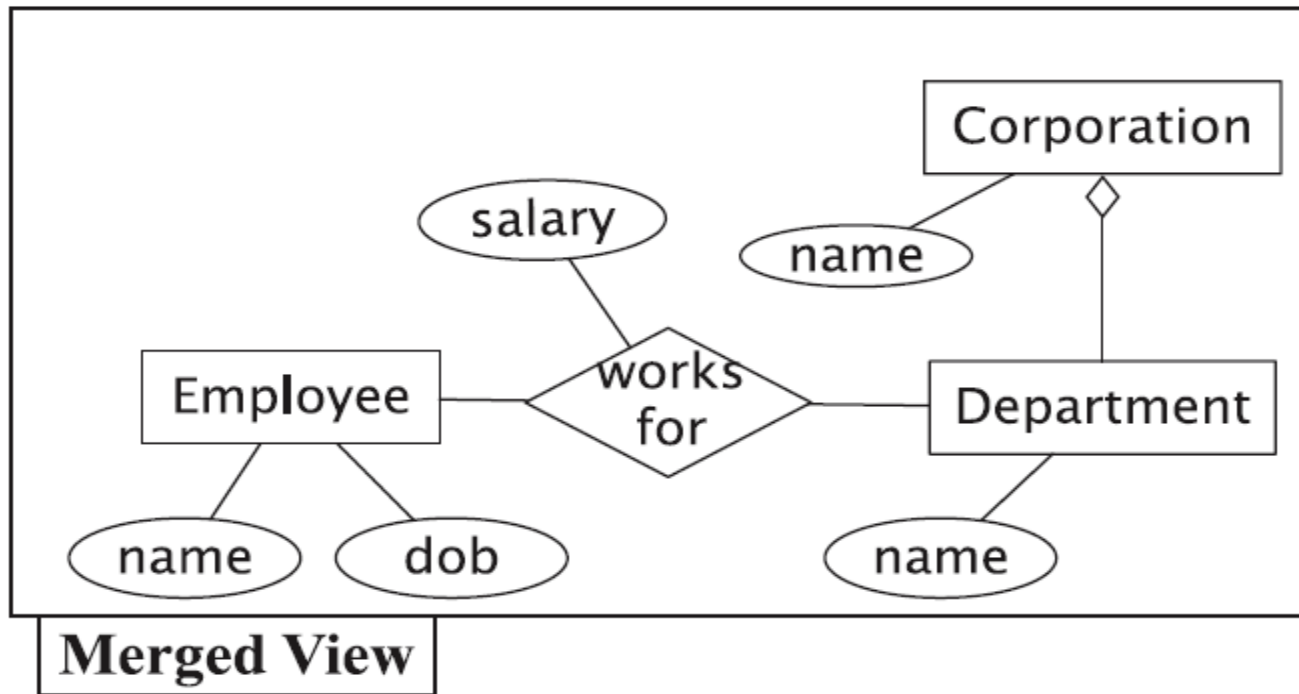
Merge example (1)



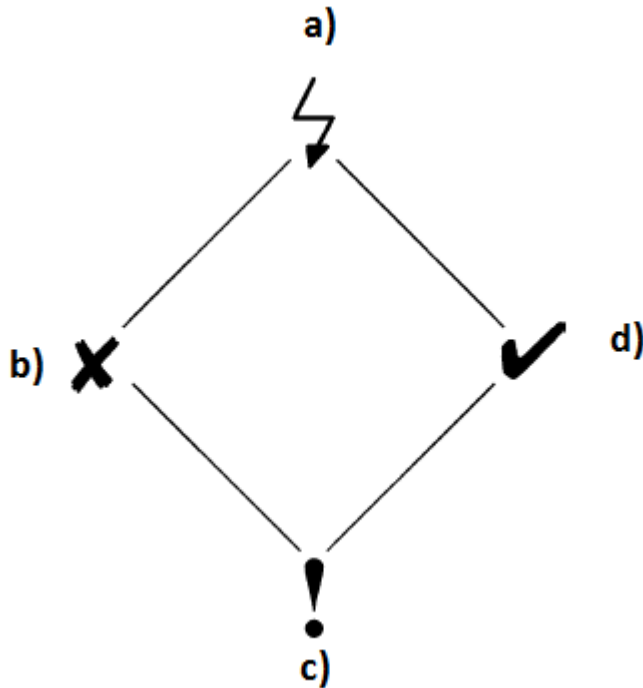
Merge example (2)



Merge example (3)



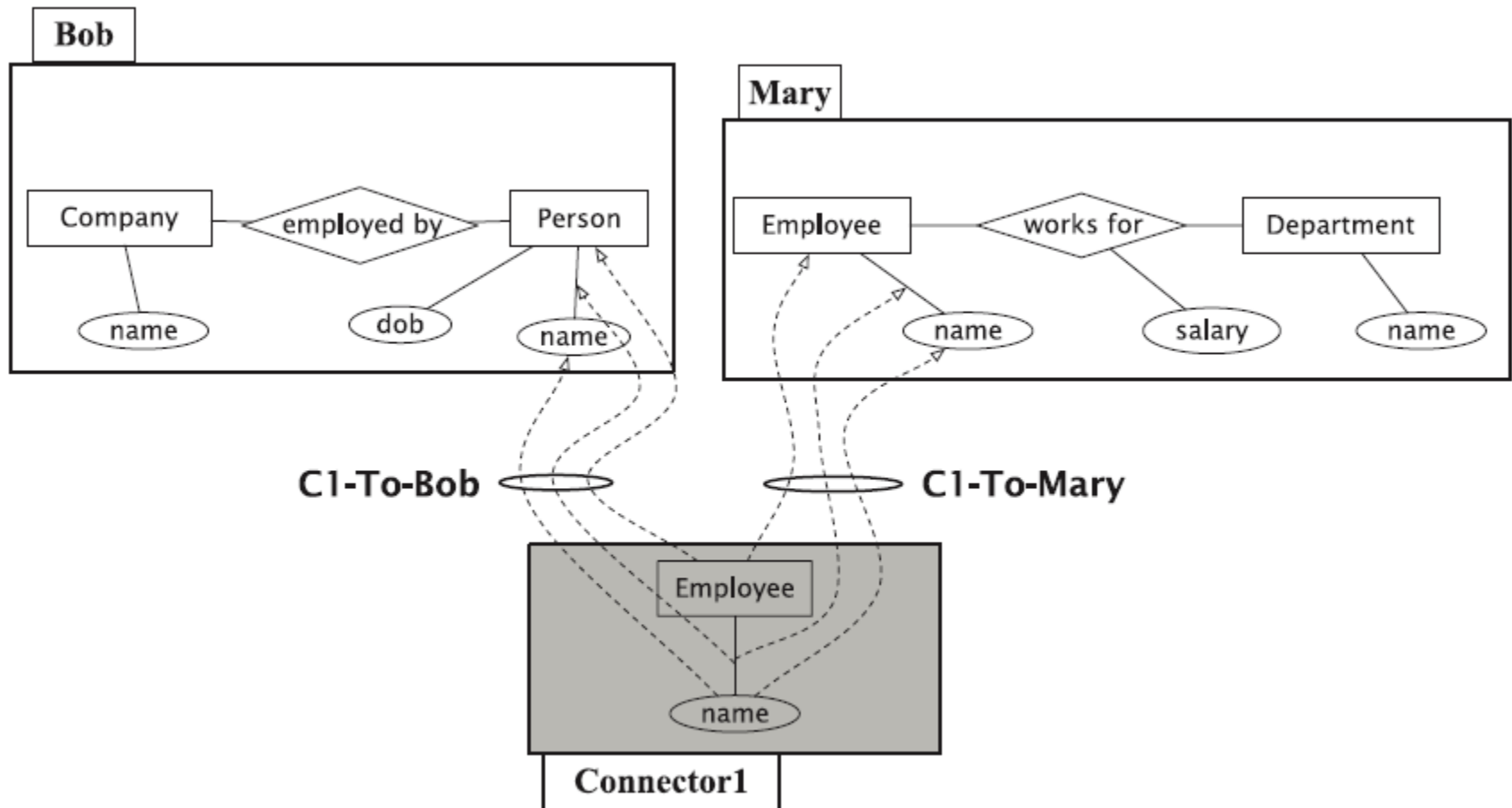
Annotations



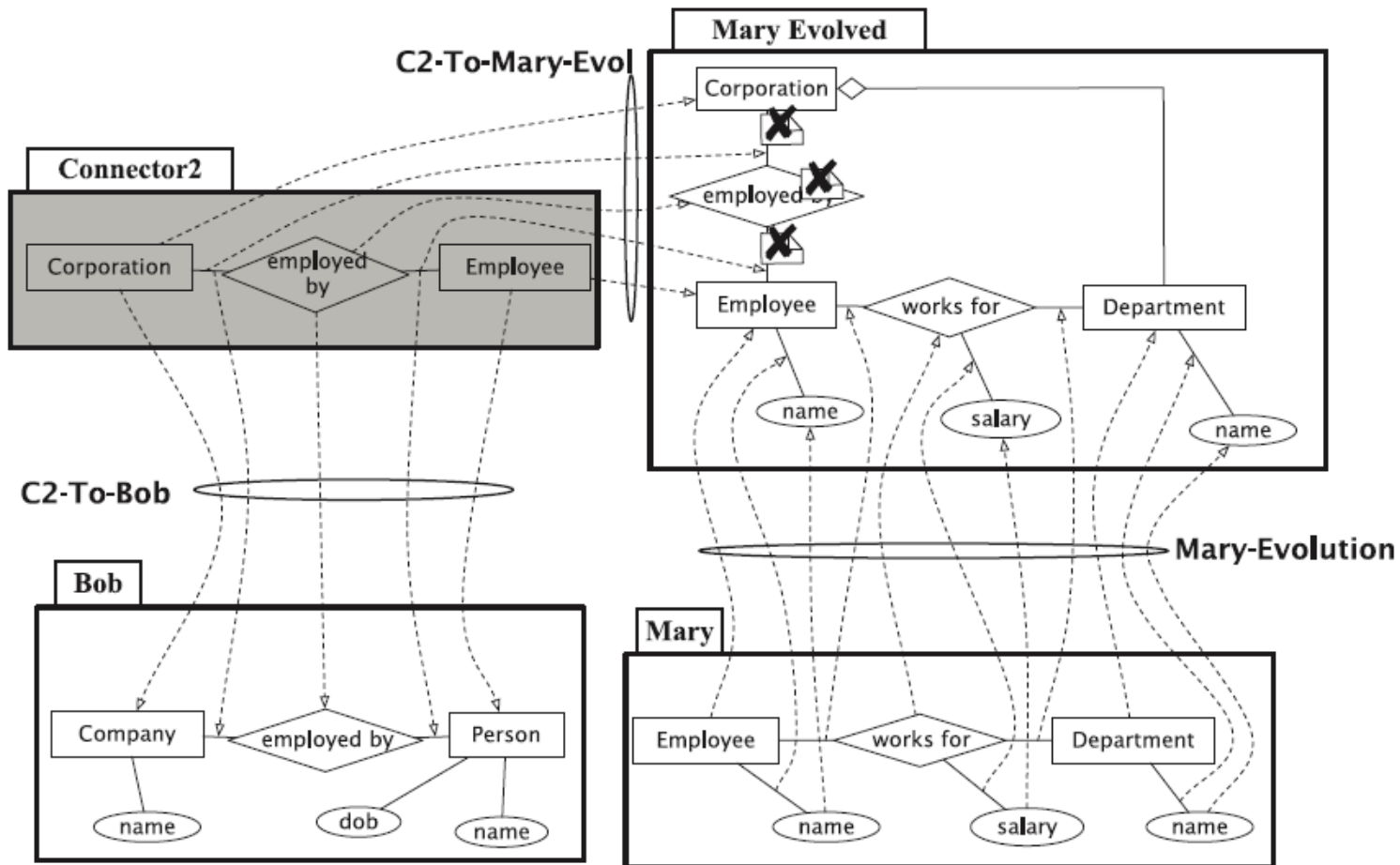
- a. Conflict
- b. Rejected
- c. Proposed
- d. Affirmed

- Shall indicate inconsistencies and give information about how sure the source is about the corresponding element

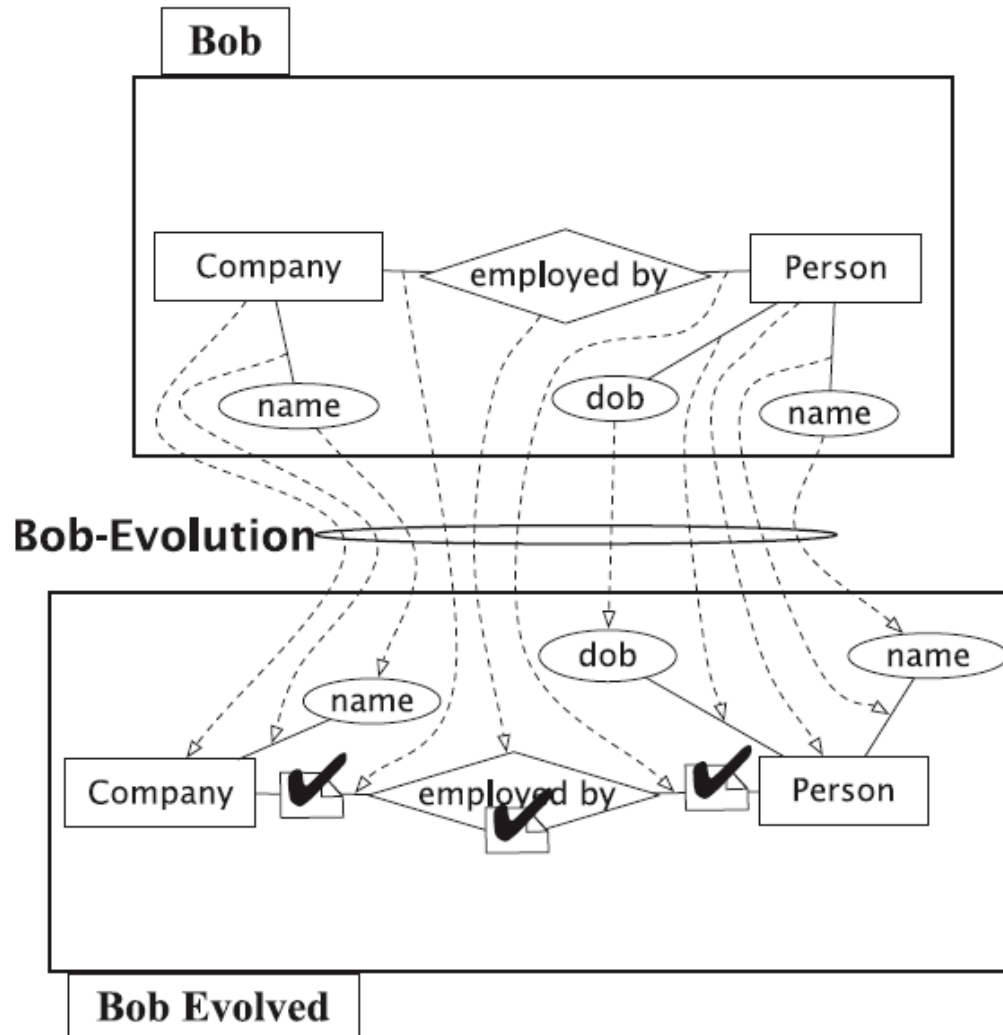
Interconnections (1)



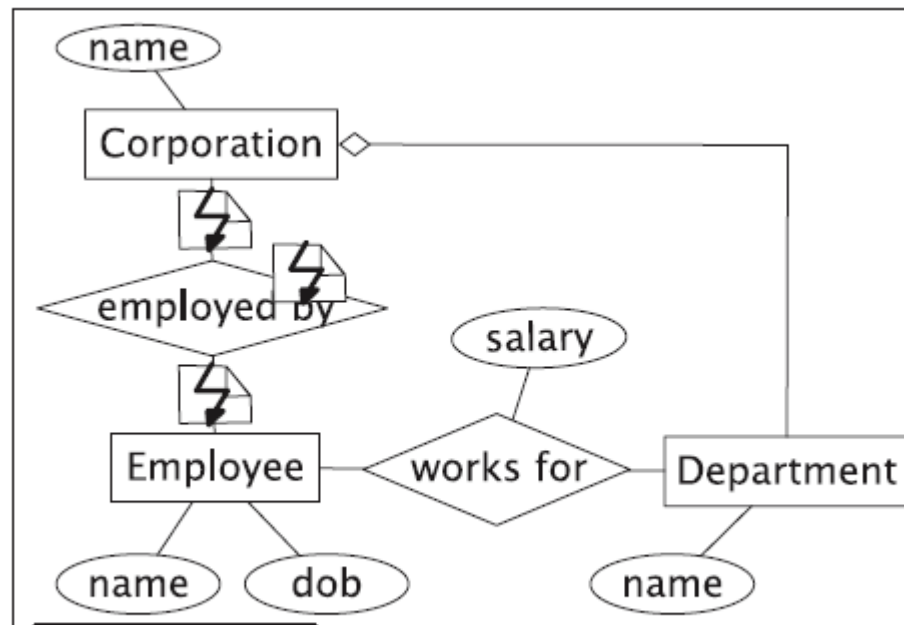
Interconnections (2)



Interconnections (3)

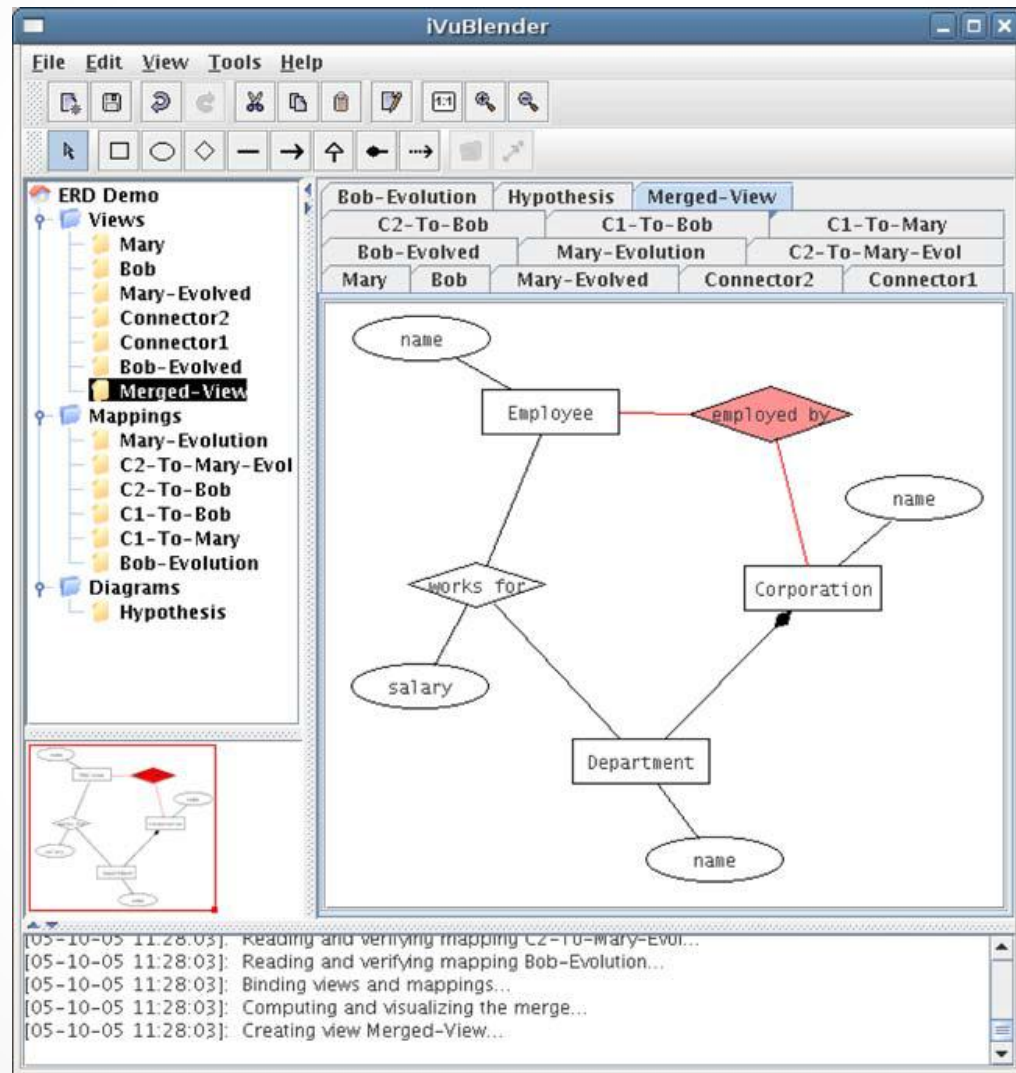


Interconnections (4)



Merged View

Tool support



Questions?